Spring Cloud基于Spring Boot,为我们提供了配置管理,服务发现,断路器,代理服务等我们在做分布式开发时常用问题的解决方案.

基于Spring Cloud开发的程序特别适合在Docker或者其他专业PaaS(平台即服务,如Cloud Foundry)部署,所以又称为原生云应用(Cloud Native Application)

1. Spring Cloud快速入门

1.1配置服务

Spring Cloud提供了Config Server,它有分布式系统开发中外部配置的功能.通过Config Server,我们可以集中存储所有应用的配置文件.

Config Server支持在git或者文件系统中放置配置文件,可以使用如下格式来区分不同应用的不同配置文件

/{application}/{profile}[/{label}]
/{application}-{profile}.yml
/{label}/{application}-{profile}.yml
/{application}-{profile}.properties
/{label}/{application}-{profile}.properties

Spring Cloud 提供了注解@EnableConfigServer来开启配置服务

1.2 服务发现

Spring Cloud通过Netflix OSS的Eureka开实现服务发现,服务发现的主要目的是为了让每个服务之间可以互相通信.Eureka Server为微服务注册中心

Spring Cloud使用注解的方式提供了Eureka服务端(@EnableEurekaServer)和客户端(@EnableEurekaClient)

1.3 路由网关

路由网关的主要目的是为了让所有的微服务对外只有一个接口,我们只需要访问一个网关地址,即可由路由网关将我们的请求代理到不同的服务中.

Spring Cloud是通过Zuul来实现的,支持自动路由映射到在Eurake Server上注册的服务.Spring Cloud提供了注解@EnableZuulProxy来开启路由代理.

1.4 负载均衡

Spring Cloud提供了Ribbon和Feign作为客户端的负载均衡.在Spring Cloud下,使用Ribbon直接注入一个RestTemplate对象即可,此RestTemplate已做好负载均衡的配置;而用Feign只需定义个注解,有@FeignClient注解的接口,然后使用@RequestMapping注解在方法上映射远程的REST服务,此方法是做好负载均衡配置的.

1.5 断路器

断路器(Circuit Breaker),主要是为了解决当某个方法调用失败的时候,调用后备方法来替代失败的方法,以达到容错,阻止级联错误等功能.

Spring Cloud使用@EnableCircuitBreaker来启用断路器支持,使用@HystrixCommand的fallbackMethod来指定后备方法.

Spring Cloud还给我们提供了一个控制台来监控断路器的运行情况.通过@EnableHystrixDashboard注解开启.

2.实战

主要由6部分微服务组成:

config:配置服务器,本例为person-service和some-service提供外部配置.

discovery:Eureka Server为微服务提供注册.

person:为UI模块提供保存person的Rest服务

some:为UI模块返回一段字符串

UI:作为应用网关,提供外部访问的唯一入口.使用Feign消费person服务,Ribbon消费some服务,且都提供断路器功能.

monitor:监控UI模块中的断路器

2.1 项目构建

使用<modules>标签来实现模块化,

使用spring-cloud-starter-parent替代spring-boot-starter-parent,其具备spring-boot-starter-parent的同样功能并附加了Spring Cloud的依赖

在此pom文件里添加的dependency对所有的子模块都是有效的,即在子模块不用额外添加这些依赖.

2.2 服务发现-Discovery(Eureka Sserver)

1. 依赖

服务发现依赖于Eureka Server,所以本模块加上如下依赖即可:

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

启动类:

```
@SpringBootApplication
@EnableEurekaServer //1 开启对Eureka Server的支持
public class DiscoveryApplication {
   public static void main(String[] args) {
      SpringApplication.run(DiscoveryApplication.class, args);
   }
}
```

application.yml:

```
server:
  port: 8761 #Eureka Server服务的端口号为8761
```

```
eureka:
  instance:
    hostname: localhost  #当前Euraka Server的hostname为localhost
  client:
    register-with-eureka: false
    fetch-registry: false  #当前服务不需要到Eureka Server上注册
```

2.3 配置-Config(Config Server)

1.依赖

Spring Cloud为我们提供了作为配置服务的依赖spring-cloud-config-server,以及作为eureka客户端的依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

启动类:

```
@SpringBootApplication
@EnableConfigServer //1 开启配置服务器的支持
@EnableEurekaClient //2  开启作为EurekaServer的客户端
public class ConfigApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigApplication.class, args);
    }
}
@EnableDiscoveryClient -> 基于spring-cloud-commons,支持eureka,consul,zk等
@EnableEurekaClient仅仅支持Eureka Server
```

bootstrap.yml:

Spring Cloud应用提供使用bootstrap.yml(bootstrap.properties)负责从外部资源加载配置属性.

```
spring:
  application:
    name: config #1 在Eureka Server注册的服务名为config
  profiles:
    active: native #2 配置服务器使用本地配置,默认为git

eureka:
  instance:
    non-secure-port: ${server.port:8080} #3 非ssl端口,若环境变量中server.port有值,则使用环境变量的值,否则使用8080
    metadata-map:
      instanceId: ${spring.application.name}:${random.value} #4 配置在Eureka Server的
```

```
实例ID
  client:
    service-url:
      defaultZone: http://${eureka.host:localhost}:${eureka.port:8761}/eureka/
#5Eureka客户端设置Eureka Server的地址
```

application.yml

```
spring:
  cloud:
    config:
      server:
        native:
          search-locations: classpath:/config #1

server:
  port: 8888

#1 person和some的配置文件在/config下
```

2.4 服务模块-Person服务

1.依赖

需要对数据库进行操作,故添加spring-boot-starter-data-jpa依赖需要使用config Server的配置,需要spring-cloud-config-client依赖:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

配置bootstrap.yml:

```
spring:
  application:
    name: person
  cloud:
    config:
      enabled: true
```

```
    discovery:
      enabled: true
      service-id: CONFIG #1  指定config server的服务名,将会通过Eureka发现config
server
eureka:
  instance:
    non-secure-port: ${server.port:8082}
  client:
    service-url:
      defaultZone: http://${eureka.host:localhost}:${eureka.port:8761}/eureka/
```

application.yml

```
server:
  port: 8082

spring:
  jpa:
    hibernate:
      ddl-auto: update
```

2.5 服务模块-Some服务

如上

配置bootstrap.yml:

```
spring:
  application:
    name: some
  cloud:
    config:
      enabled: true
      discovery:
        enabled: true
        service-id: CONFIG
eureka:
  instance:
    non-secure-port: ${server.port:8083}
  client:
    service-url:
      defaultZone: http://${eureka.host:localhost}:${eureka.port:8761}/eureka/
```

2.6 界面模块-UI(Ribbon,Feign)

1.依赖

本模块会使用Ribbon,feign,zuul以及CircuitBreaker,所以需要添加相关依赖,本模块是一个具有页面的模块,所以需要通过webjar加载一些常用的脚本框架:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
```

```
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>
```

2.关键代码:

(1) 入口类:

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients //1 开启feign客户端支持
@EnableCircuitBreaker //2 开启circuitBreaker的支持
@EnableZuulProxy //3 开启网关代理的支持
public class UiApplication {
  public static void main(String[] args) {
      SpringApplication.run(UiApplication.class, args);
   }
}
```

(2) 使用feign调用person服务:

```
@FeignClient("person")
public interface PersonService {
   @RequestMapping(method = RequestMethod.POST, value = "/save",
           produces = MediaType.APPLICATION_JSON_VALUE, consumes =
MediaType.APPLICATION_JSON_VALUE)
     @ResponseBody
    List<Person> save(@RequestBody String name);
}
```

(3) 调用Person Server的断路器:

```
@Service
public class PersonHystrixService {

  @Autowired
  PersonService personService;

  @HystrixCommand(fallbackMethod = "fallbackSave") //1
  public List<Person> save(String name) {
    return personService.save(name);
  }

  public List<Person> fallbackSave(String name){
    List<Person> list = new ArrayList<>();
    Person p = new Person(name+"没有保存成功，Person Service 故障");
    list.add(p);
    return list;
  }
}

#1 使用@HystrixCommand的fallbackMethod参数指定,
   当本方法调用失败时,调用后备方法fallbackSave
```

(4) 使用Ribbon调用SomeService,并使用断路器:

```
@Service
public class SomeHystrixService {

  @Autowired
   RestTemplate restTemplate; //1

  @HystrixCommand(fallbackMethod = "fallbackSome") //2
  public String getSome() {
    return restTemplate.getForObject("http://some/getsome", String.class);
  }

  public String fallbackSome(){
    return "some service模块故障";
  }
}

#1 在Spring Boot下使用Ribbon,我们只需要注入一个RestTemplate即可,
   Spring Boot已为我们做好了配置
#2  使用@HystrixCommand的fallbackMethod参数指定,
   当本方法调用失败时,调用后备方法fallbackSave
```

3.配置

bootstrap.yml

```
spring:
  application:
```

```
    name: ui

eureka:
  instance:
    non-secure-port: ${server.port:80}
  client:
    service-url:
      defaultZone: http://${eureka.host:localhost}:${eureka.port:8761}/eureka/
```

application.yml:

```
server:
  port: 80
```

2.7 断路器监控-Monitor(DashBoard)

1. 依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-turbine</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2.主要代码

```
@SpringBootApplication
@EnableEurekaClient
@EnableHystrixDashboard
@EnableTurbine
public class MonitorApplication {

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

bootstrap.yml

```
spring:
  application:
    name: monitor

eureka:
  instance:
    nonSecurePort: ${server.port:8989}
  client:
```

```
    serviceUrl:
      defaultZone: http://${eureka.host:localhost}:${eureka.port:8761}/eureka/
```

application.yml

```
server:
  port: 8989
```

3.基于Docker部署

3.1 Dockerfile编写

以上6个服务的Dockerfile编写几乎完全一样,以config为例:

1.runboot.sh脚本编写

位于src/main/docker下:

```
sleep 10
java -Djava.security.edg=file:/dev/./urandom -jar /app/app.jar
```

根据启动顺序调整sleep的时间

2.Dockerfile编写

位于src/main/docker下

```
FROM java:8
VOLUME /tmp
RUN mkdir /app
ADD config-1.0.0-SNAPSHOT.jar /app/app.jar
ADD runboot.sh /app/
RUN bash -c 'touch /app/app.jar'
WORKDIR /app
RUN chmod a+x runboot.sh
EXPOSE 8888
CMD /app/runboot.sh
为不同的微服务只需修改
ADD config-1.0.0-SNAPSHOT.jar /app/app.jar
以及端口
EXPOSE 8888
```

3.Docker的maven插件

开发机器编译Docker镜像到服务器,使用docker-maven-plugin即可,在所有程序的pom.xml内增加:

```
<build>
    <plugins>
        <plugin>
            <groupId>io.fabric8</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>0.30.0</version>
            <configuration>
                <imageName>${project.name}:${project.version}</imageName>
```

```
            <dockerDirectory>${project.basedir}/src/main/docker</dockerDirectory>
            <skipDockerBuild>false</skipDockerBuild>
            <resources>
               <resource>
                  <directory>${project.build.directory}</directory>
                  <include>${project.build.finalName}</include>
               </resource>
            </resources>
         </configuration>
      </plugin>
   </plugins>
</build>
```

4.编译镜像

使用docker-maven-plugin,默认将Docker编译到localhost.如果时远程Linux服务器,请在环境变量中配置DOCKER_HOST:

```
DOCKER_HOST
tcp://192.168.1.116:2375
```

在控制台下进入项目根目录,执行下面语句:

```
mvn clean package docker:build -DskipTests
```

3.2 Docker Compose

Docker Compose是用来定义和运行多容器应用的工具,

Docker Compose使用一个docker-compose.yml来描述多容器的定义,使用下面命令运行整个应用:

```
docker-compose up
```

3.3 Docker-compose.yml编写

```
postgresdb:
  image: busybox
  volumes:
    - /var/lib/postgresql/data

postgres:
  name: postgres
  image: postgres
  hostname: postgres
  volumes_from:
    - postgresdb
#  ports:
#   - "5432:5432"
  environment:
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=postgres

discovery:
```

```
  image: "discovery:1.0.0-SNAPSHOT"
  hostname: discovery
  name: discovery
  ports:
   - "8761:8761"

config:
  image: "config:1.0.0-SNAPSHOT"
  hostname: config
  name: config
  links:
    - discovery
  environment:
     EUREKA_HOST: discovery
     EUREKA_PORT: 8761
#  ports:
#    - "8888:8888"

person:
  image: person:1.0.0-SNAPSHOT
  hostname: person
  links:
    - discovery
    - config
    - postgres
  environment:
     EUREKA_HOST: discovery
     EUREKA_PORT: 8761
     SPRING_PROFILES_ACTIVE: docker
#  ports:
#    - "8082:8082"

some:
  image: some:1.0.0-SNAPSHOT
  hostname: some
  links:
    - discovery
    - config
  environment:
     EUREKA_HOST: discovery
     EUREKA_PORT: 8761
     SPRING_PROFILES_ACTIVE: docker
#  ports:
#    - "8083:8083"

ui:
  image: ui:1.0.0-SNAPSHOT
```

```
  hostname: ui
  links:
    - discovery
    - config
    - person
    - some
  environment:
      EUREKA_HOST: discovery
      EUREKA_PORT: 8761
      SPRING_PROFILES_ACTIVE: docker
  ports:
    - "80:80"

monitor:
  image: monitor:1.0.0-SNAPSHOT
  hostname: monitor
  links:
    - discovery
    - config
    - person
    - some
    - ui
  environment:
      EUREKA_HOST: discovery
      EUREKA_PORT: 8761
      SPRING_PROFILES_ACTIVE: docker
#  ports:
#    - "8989:8989"
```

解释:

!.enviroment:给容器使用的变量,在容器中使用${}来调用

2.links:当前容器依赖的容器,可直接使用依赖容器的已有端口

3.ports:将我们需要的暴露的端口映射出来,不需要暴露的端口则不做映射

3.4 运行

将docker-compose.yml上传至Linux服务器,在文件当前目录执行:

```
docker-compose up -d
```