[建立主体工程；maven，打包方式pom；配置pom.xml 2](#_Toc18251)

[建立子类工程；module，打包方式jar；导包。 2](#_Toc6412)

[1， 注册中心 2](#_Toc32736)

[2， 服务提供者 2](#_Toc15947)

[3， 服务消费者 3](#_Toc21700)

[Ribbon方式 3](#_Toc23775)

[Feign方式 3](#_Toc24554)

[4， 断路器 4](#_Toc19172)

[Ribbon项目使用断路器 4](#_Toc6930)

[Feign中使用断路器 4](#_Toc28393)

[Hystrix Dashboard (断路器：Hystrix 仪表盘) 4](#_Toc32561)

[5， 路由网关zuul 4](#_Toc24614)

[路由 4](#_Toc10318)

[服务过滤 5](#_Toc28015)

[6， 配置中心 5](#_Toc26931)

[构建Config Server 5](#_Toc26785)

[构建config client 6](#_Toc21911)

[7， 消息总线 6](#_Toc17057)

[改造config client 6](#_Toc16050)

[8， 服务链路追踪(Spring Cloud Sleuth) 7](#_Toc9250)

[构建server-zipkin 7](#_Toc15705)

[修改服务 7](#_Toc20941)

[9， 高可用的服务注册中心 7](#_Toc32277)

[10， docker部署spring cloud项目 8](#_Toc24196)

[11， 断路器监控(Hystrix Dashboard) 8](#_Toc26319)

[12， 断路器聚合监控(Hystrix Turbine) 8](#_Toc1235)

[13， 服务注册(consul) 9](#_Toc11020)

[Linux平台安装consul 9](#_Toc2249)

[Windows平台安装consul 9](#_Toc19143)

[构建工程 9](#_Toc19929)

# **建立主体工程；maven，打包方式pom；配置pom.xml**

1. 配置properties；java版本、spring-boot版本、字符集等。

<properties>

<java.version>1.8</java.version>

<maven.compiler.plugin.version>3.7.0</maven.compiler.plugin.version>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<maven.compiler.encoding>UTF-8</maven.compiler.encoding>

<spring-boot.version>1.5.7.RELEASE</spring-boot.version>

<spring-boot-maven-plugin.version>1.5.8.RELEASE</spring-boot-maven-plugin.version>

<spring-cloud.version>Dalston.SR4</spring-cloud.version>

<springfox-swagger.version>2.7.0</springfox-swagger.version>

<joda-time.version>2.9.9</joda-time.version>

<docker.plugin.version>0.4.13</docker.plugin.version>

<docker.image.prefix>bonc-ids</docker.image.prefix>

<docker.host>http://192.168.1.123:2375</docker.host>

</properties>

1. 配置dependencyManagement，通过${spring-boot.version}获取properties配置，统一管理依赖版本。spring-boot-dependencies，spring-cloud-dependencies，joda-time
2. 配置dependencies，导入依赖包：spring-boot，spring-cloud，swagger。版本通过${...}获取。spring-boot-starter，spring-boot-starter-web，spring-boot-starter-test，spring-boot-starter-thymeleaf，spring-cloud-starter-eureka，springfox-swagger2，springfox-swagger-ui

# 建立子类工程；module，打包方式jar；导包。

选择simple project，否则可能结构不全。

## 注册中心

导入spring-cloud-starter-eureka-server

启动类添加@EnableEurekaServer；

配置application.yml：

server:

port: 8761

eureka:

instance:

hostname: localhost

client:

registerWithEureka: **false**

fetchRegistry: **false**

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

## 服务提供者

导入spring-cloud-starter-eureka

启动类添加@EnableEurekaClient

配置application.yml：

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8762

spring:

application:

name: service-hi

## 服务消费者

### Ribbon方式

导包spring-cloud-starter-eureka、spring-cloud-starter-ribbon；

启动类添加@EnableDiscoveryClient

启动类注入bean: restTemplate(@Bean)；通过@LoadBalanced开启负载均衡的功能

配置application.yml：

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8764

spring:

application:

name: service-ribbon

### Feign方式

feign集成了ribbon。

导入spring-cloud-starter-eureka、spring-cloud-starter-feign；

启动类添加@EnableFeignClients、@EnableDiscoveryClient。

添加接口，接口添加@Feign(value=”服务名”)；方法添加@RequestMapping

配置application.yml：

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8765

spring:

application:

name: service-feign

feign:

hystrix:

enabled: **true**

## 断路器

服务故障时，打开断路器，调用fallback()。

断路器打开后，负载均衡不再访问故障服务？？？

服务恢复时，不自动断开断路器？？？

浏览器缓存原因；Hystric默认阀值5秒20次

### Ribbon项目使用断路器

导入spring-cloud-starter-hystrix；启动类添加@EnableHystrix

Service方法加@HystrixCommand(fallbackMethod=”a”)；实现方法a。

### Feign中使用断路器

修改配置文件，打开自带熔断器，

feign.hystrix.enabled=true

修改注解：@FeignClient(value = "service-hi",fallback = S.class)

创建S类实现接口，注入容器(@Component)；实现方法即为断路器调用方法。

### Hystrix Dashboard (断路器：Hystrix 仪表盘)

导入spring-cloud-starter-hystrix-dashboard；

启动类添加@EnableHystrixDashboard

## 路由网关zuul

Zuul的主要功能是路由转发和过滤器。zuul默认和Ribbon结合实现了负载均衡的功能

### 路由

导入spring-cloud-starter-eureka、spring-cloud-starter-feign；

启动类添加@EnableZuulProxy

配置application.yml：

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8769

spring:

application:

name: service-zuul

zuul:

routes:

api-a:

path: /api-a/\*\*

serviceId: service-ribbon

api-b:

path: /api-b/\*\*

serviceId: service-feign

### 服务过滤

继承抽象类ZuulFilter，重写方法。

**filterType()**：返回一个字符串代表过滤器的类型，在zuul中定义了四种不同生命周期的过滤器类型，具体如下：pre--路由之前，routing--路由之时，post--路由之后，error--发送错误调用

**filterOrder()**：过滤的顺序

**shouldFilter()**：这里可以写逻辑判断，是否要过滤，本文true,永远过滤。

**run()**：过滤器的具体逻辑。可用很复杂，包括查sql，nosql去判断该请求到底有没有权限访问

## 配置中心

在Spring Cloud中，有分布式配置中心组件spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。在spring cloud config 组件中，分两个角色，一是config server，二是config client。

### 构建Config Server

导入spring-cloud-starter-eureka，spring-cloud-config-server

启动类添加@EnableConfigServer

配置application.yml：

server:

port: 8888

spring:

application:

name: config-server

cloud:

config:

label: master

server:

git:

password: your password

searchPaths: respo

uri: https://github.com/forezp/SpringcloudConfig/

username: your username

http请求地址和资源文件映射如下:

/{application}/{profile}[/{label}]

/{application}-{profile}.yml

/{label}/{application}-{profile}.yml

/{application}-{profile}.properties

/{label}/{application}-{profile}.properties

### 构建config client

导入spring-cloud-starter-eureka，spring-cloud-starter-config

启动类添加@EnableConfigServer；@Value("${foo}")，获取foo配置值

配置bootstrap.properties：

spring.application.name=config-client

spring.cloud.config.label=master 远程仓库的分支

spring.cloud.config.profile=dev dev开发环境，test测试环境，pro正式环境

#spring.cloud.config.uri= http://localhost:8888/

eureka.client.serviceUrl.defaultZone=http://localhost:8889/eureka/

spring.cloud.config.discovery.enabled=true

spring.cloud.config.discovery.serviceId=config-server

server.port=8881

## 消息总线

Spring Cloud Bus 将分布式的节点用轻量的消息代理连接起来。它可以用于广播配置文件的更改或者服务之间的通讯，也可以用于监控。本文要讲述的是用Spring Cloud Bus实现通知微服务架构的配置文件的更改。

### 改造config client

导包spring-retry，spring-boot-starter-aop,spring-cloud-starter-bus-amqp，spring-boot-starter-actuator。

修改配置文件

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

# spring.rabbitmq.username=

# spring.rabbitmq.password=

发送post请求：[http://localhost:8881/bus/refresh](http://localhost:8881/bus/refresh" \t "http://blog.csdn.net/forezp/article/details/_blank)，重新读取配置文件

/bus/refresh接口可以指定服务，即使用”destination”参数，比如 “/bus/refresh?destination=customers:\*\*” 即刷新服务名为customers的所有服务，不管ip。

## [服务链路追踪(Spring Cloud Sleuth)](http://blog.csdn.net/forezp/article/details/70162074)

Spring Cloud Sleuth集成了服务追踪组件zipkin。主要功能就是在分布式系统中提供追踪解决方案，并且兼容支持了 zipkin，你只需要在pom文件中引入相应的依赖即可。

### 构建server-zipkin

导包zipkin-server，zipkin-autoconfigure-ui

启动类添加@EnableZipkinServer，开启ZipkinServer的功能

配置文件

server.port=9411

### 修改服务

导包spring-cloud-starter-zipkin

启动类添加@EnableZipkinServer，开启ZipkinServer的功能

配置文件

server.port=8988

spring.zipkin.base-url=http://localhost:9411 spring.application.name=service-hi

服务中调用其他服务；zipkin会自动记录。

## [高可用的服务注册中心](http://blog.csdn.net/forezp/article/details/70183572)

修改服务注册中心，添加多个服务注册中心，实现Eureka Server集群化。

添加两个配置文件，application-peer1.yml，application-peer2.yml

server:

port: 8761

spring:

profiles: peer1

eureka:

instance:

hostname: peer1

client:

serviceUrl:

defaultZone: http://peer1:8769/eureka/

修改hosts文件

127.0.0.1 peer1

127.0.0.1 peer2

更改服务注册url

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8762

spring:

application:

name: service-hi

eureka.instance.preferIpAddress=true是通过设置ip让eureka让其他服务注册它。

Eureka-eserver peer1 8761,Eureka-eserver peer2 8769相互感应，当有服务注册时，两个Eureka-eserver是对等的，它们都存有相同的信息。

## [docker部署spring cloud项目](http://blog.csdn.net/forezp/article/details/70198649)

### 改造eureka-server工程

在pom文件加上插件，docker-maven-plugin

<plugin>

<groupId>com.spotify</groupId>

<artifactId>docker-maven-plugin</artifactId>

<version>0.4.3</version>

<configuration>

<imageName>${docker.image.prefix}/${project.artifactId}</imageName>

<dockerDirectory>src/main/docker</dockerDirectory>

<resources>

<resource>

<targetPath>/</targetPath>

<directory>${project.build.directory}</directory>

<include>${project.build.finalName}.jar</include>

</resource>

</resources>

</configuration>

</plugin>

Spotify 的 docker-maven-plugin 插件是用maven插件方式构建docker镜像的。

imageName指定了镜像的名字，本例为 forep/eureka-server

dockerDirectory指定 Dockerfile 的位置

resources是指那些需要和 Dockerfile 放在一起，在构建镜像时使用的文件，一般应用 jar 包需要纳入。

修改配置文件：

server:

port: 8761

eureka:

instance:

prefer-ip-address: **true**

client:

registerWithEureka: **false**

fetchRegistry: **false**

**编写dockerfile文件**

FROM frolvlad/alpine-oraclejdk8:slim

VOLUME /tmp

ADD eureka-server-0.0.1-SNAPSHOT.jar app.jar

#RUN bash -c 'touch /app.jar'

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

EXPOSE 8761

**docker file编写指令**

FROM

FROM <image>

FROM <image>:<tag>

FROM <image> <digest>

FROM指令必须指定且需要在Dockerfile其他指令的前面，指定的基础image可以是官方远程仓库中的，也可以位于本地仓库。后续的指令都依赖于该指令指定的image。当在同一个Dockerfile中建立多个镜像时，可以使用多个FROM指令。

VOLUME

格式为：VOLUME ["/data"]

使容器中的一个目录具有持久化存储数据的功能，该目录可以被容器本身使用，也可以共享给其他容器。当容器中的应用有持久化数据的需求时可以在Dockerfile中使用该指令。

ADD

从src目录复制文件到容器的dest。其中src可以是Dockerfile所在目录的相对路径，也可以是一个URL，还可以是一个压缩包

ENTRYPOINT

指定Docker容器启动时执行的命令，可以多次设置，但是只有最后一个有效。

EXPOSE

为Docker容器设置对外的端口号。在启动时，可以使用-p选项或者-P选项。

构建镜像

执行构建docker镜像maven命令：

mvn clean

mvn package docker:build

运行docker

docker run -p 8761: 8761 -t forezp/eureka-server

## [断路器监控(Hystrix Dashboard)](http://blog.csdn.net/forezp/article/details/70217283)

改造服务，添加Hystrix Dashboard

导包spring-boot-starter-actuator，spring-cloud-starter-hystrix，

spring-cloud-starter-hystrix-dashboard。

启动类添加@EnableHystrix、@EnableHystrixDashboard；

程序中声明断路点；ribbon--@HystrixCommand，feign--@FeignClient。

## [断路器聚合监控(Hystrix Turbine)](http://blog.csdn.net/forezp/article/details/70233227)

导包spring-cloud-starter-turbine，spring-cloud-netflix-turbine，

spring-boot-starter-actuator

启动类添加@EnableTurbine，开启turbine，@EnableTurbine注解包含了@EnableDiscoveryClient注解，即开启了注册服务。

配置文件

spring:

application.name: service-turbine

server:

port: 8769

security.basic.enabled: **false**

turbine:

aggregator:

clusterConfig: default # 指定聚合哪些集群，多个使用","分割，默认为default。可使用http://.../turbine.stream?cluster={clusterConfig之一}访问

appConfig: service-hi,service-lucy ### 配置Eureka中的serviceId列表，表明监控哪些服务

clusterNameExpression: new String("default")

# 1. clusterNameExpression指定集群名称，默认表达式appName；此时：turbine.aggregator.clusterConfig需要配置想要监控的应用名称

# 2. 当clusterNameExpression: default时，turbine.aggregator.clusterConfig可以不写，因为默认就是default

# 3. 当clusterNameExpression: metadata['cluster']时，假设想要监控的应用配置了eureka.instance.metadata-map.cluster: ABC，则需要配置，同时turbine.aggregator.clusterConfig: ABC

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

## [服务注册(consul)](http://blog.csdn.net/forezp/article/details/70245644)

spring cloud consul，提供服务发现和配置的工具。具有分布式、高可用、高扩展性。

### Linux平台安装consul

mkdir -p $GOPATH/src/github.com/hashicorp && cd $!

git clone https://github.com/hashicorp/consul.git

cd consul

make bootstrap

make bootstrap

### Windows平台安装consul

官网下载：[https://www.consul.io/downloads.html](https://www.consul.io/downloads.html" \t "http://blog.csdn.net/forezp/article/details/_blank)

安装后，在环境变量path下加上：E:\programfiles\consul；

cmd启动：consul agent -dev

### 构建工程

导包spring-cloud-starter-consul-discovery

启动类添加@EnableDiscoveryClient，开启服务发现

配置文件

spring:

cloud:

consul:

host: localhost

port: 8500

discovery:

healthCheckPath: ${management.contextPath}/health

healthCheckInterval: 15s

instance-id: consul-miya

application:

name: consul-miya

server:

port: 8502