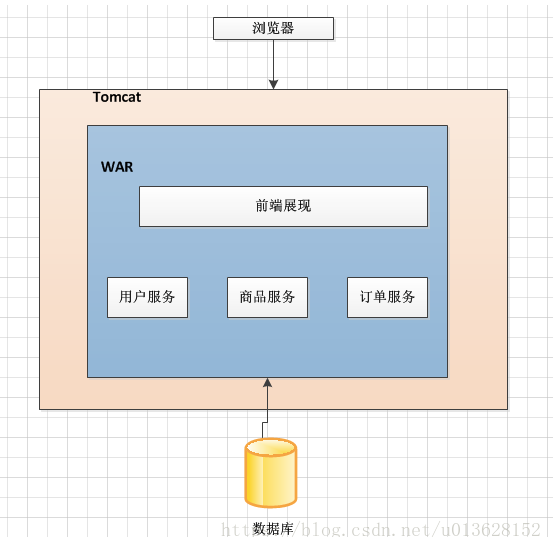
# 1微服务架构介绍

## 1.1项目单体架构

单体架构也称之为单体系统或者是单体应用。就是一种把系统中所有的功能、模块耦合在一个应用中的架构方式。单体服务架构，将所有的功能模块(service)打包到一起并放在一个web容器中运行。

### 1.1.1单体架构特征

打包成一个独立的单元(导成一个唯一的 jar 包或者是 war 包)，会一个进程的方式来运行



### 1.1.2单体架构优点

项目易于管理

部署简单

### 1.1.3单体架构缺点

编译难，部署难，测试难:

代码量变多，即使更改一行代码，也需花大量时间编译，部署前要编译打包，解压等所以部署难，部署完了还要测试所以测试难。

技术选择难:

在变得越来越大的同时，我们的应用所使用的技术也会变得越来越多。这些技术有些是不兼容的，就比如在一个项目中大范围地混合使用C++和Java几乎是不可能的事情。在这种情况下，我们就需要抛弃对某些不兼容技术的使用，而选择一种不是那么适合的技术来实现特定的功能。

扩展难:

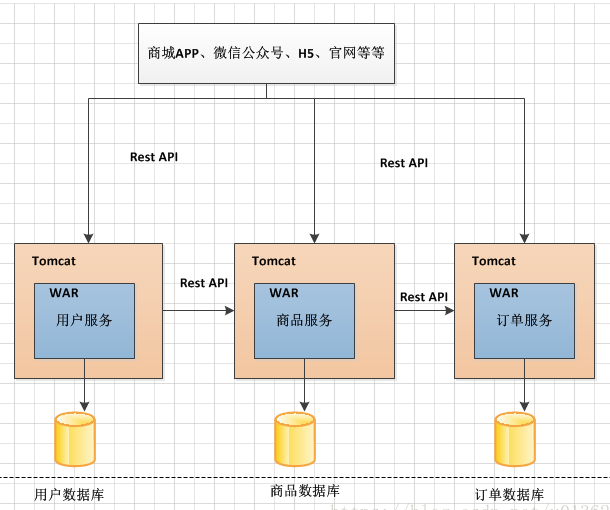
按照单体架构组织的代码将只产生一个包含了所有功能的WAR包，因此在对服务的容量进行扩展的时候，我们只能选择重复地部署这些WAR包来扩展服务能力，而不是仅仅扩展出现系统瓶颈的组成. 跨语言程度差.

## 1.2 MicroService(微服务)架构

微服务架构是一种架构模式，它提倡将单一应用程序划分成一组小的服务，就是将复杂臃肿的单体应用进行细粒度的服务拆分，每个微服务可以交给小的团队进行开发和维护，拆分出来的服务各自独立打包部署。

### 1.2.1微服务架构特征

微服务就是把一个单体项目,拆分为多个微服务,每个微服务可以独立技术选型,独立开发,独立部署,独立运维.并且多个服务相互协调,相互配合,最终完成用户的价值.



### 1.2.2微服务架构优点

**复杂度可控：**

在将应用分解的同时，规避了原本复杂度无止境的积累。每一个微服务专注于单一功能，并通过定义良好的接口清晰表述服务边界。由于体积小、复杂 度低，每个微服务可由一个小规模开发团队完全掌控，易于保持高可维护性和开发效率。

**独立部署：**

由于微服务具备独立的运行进程，所以每个微服务也可以独立部署。当某个微服务发生变更时无需编译、部署整个应用。由微服务组成的应用相当于具备一系列可并行的发布流程，使得发布更加高效，同时降低对生产环境所造成的风险，最终缩短应用交付周期。

**技术选型灵活：**

微服务架构下，技术选型是去中心化的。每个团队可以根据自身服务的需求和行业发展的现状，自由选择最适合的技术栈。由于每个微服务相对简单，故需要对技术栈进行升级时所面临的风险就较低，甚至完全重构一个微服务也是可行的。

**容错：**

当某一组建发生故障时，在单一进程的传统架构下，故障很有可能在进程内扩散，形成应用全局性的不可用。在微服务架构下，故障会被隔离在单个服务中。若设计良好，其他服务可通过重试、平稳退化等机制实现应用层面的容错。

**扩展：**

单块架构应用也可以实现横向扩展，就是将整个应用完整的复制到不同的节点。当应用的不同组件在扩展需求上存在差异时，微服务架构便体现出其灵活性，因为每个服务可以根据实际需求独立进行扩展。

### 1.2.3微服务架构缺点

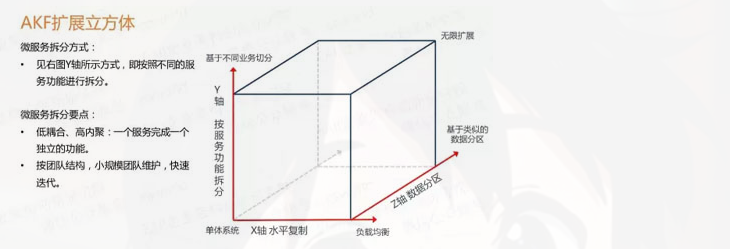
微服务过多，服务治理成本高，不利于系统维护。

分布式系统开发的技术成本高（容错、分布式事务等），对团队挑战大

# 2微服务应用4个设计原则



## 2.1AKF拆分原则



AKF扩展立方体(参考《The Art of Scalability》)，是一个叫AKF的公司的技术专家抽象总结的应用扩展的三个维度。理论上按照这三个扩展模式，可以将一个单体系统，进行无限扩展。

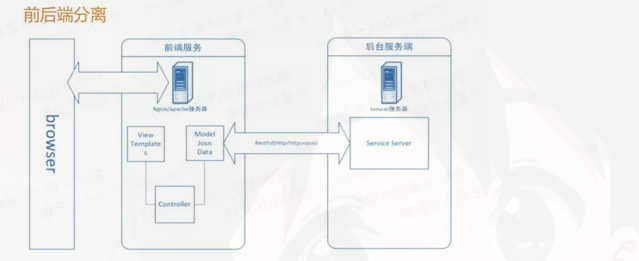
X 轴 ：指的是水平复制，很好理解，就是讲单体系统多运行几个实例，做个集群加负载均衡的模式。

Z 轴 ：是基于类似的数据分区，比如一个互联网打车应用突然用户量激增，集群模式撑不住了，那就按照用户请求的地区进行数据分区，北京、上海、四川等多建几个集群。

Y 轴 ：就是我们所说的微服务的拆分模式，就是基于不同的业务拆分。

场景说明：比如打车应用，一个集群撑不住时，分了多个集群，后来用户激增还是不够用，经过分析发现是乘客和车主访问量很大，就将打车应用拆成了三个乘客服务、车主服务、支付服务。三个服务的业务特点各不相同，独立维护，各自都可以再次按需扩展。

## 2.2前后端分离



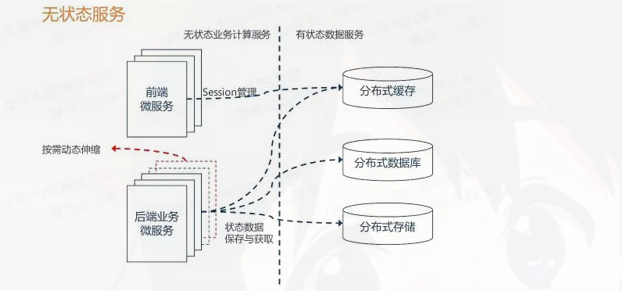
前后端分离原则，简单来讲就是前端和后端的代码分离也就是技术上做分离，我们推荐的模式是最好直接采用物理分离的方式部署，进一步促使进行更彻底的分离。不要继续以前的服务端模板技术，比如JSP ，把Java JS HTML CSS 都堆到一个页面里，稍复杂的页面就无法维护。这种分离模式的方式有几个好处：

前后端技术分离，可以由各自的专家来对各自的领域进行优化，这样前端的用户体验优化效果会更好。

分离模式下，前后端交互界面更加清晰，就剩下了接口和模型，后端的接口简洁明了，更容易维护。

前端多渠道集成场景更容易实现，后端服务无需变更，采用统一的数据和模型，可以支撑前端的web UI 移动App等访问

## 2.3无状态服务

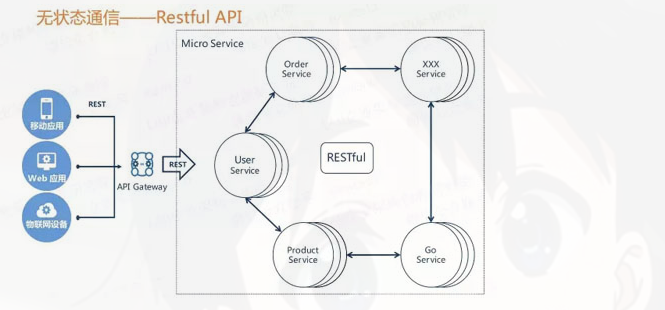


对于无状态服务，首先说一下什么是状态：如果一个数据需要被多个服务共享，才能完成一笔交易，那么这个数据被称为状态。进而依赖这个“状态”数据的服务被称为有状态服务，反之称为无状态服务。

那么这个无状态服务原则并不是说在微服务架构里就不允许存在状态，表达的真实意思是要把有状态的业务服务改变为无状态的计算类服务，那么状态数据也就相应的迁移到对应的“有状态数据服务”中。

场景说明：例如我们以前在本地内存中建立的数据缓存、Session缓存，到现在的微服务架构中就应该把这些数据迁移到分布式缓存中存储，让业务服务变成一个无状态的计算节点。迁移后，就可以做到按需动态伸缩，微服务应用在运行时动态增删节点，就不再需要考虑缓存数据如何同步的问题。

## 2.4 Restful通信风格



作为一个原则来讲本来应该是个“无状态通信原则”，在这里我们直接推荐一个实践优选的Restful 通信风格 ，因为他有很多好处：

无状态协议HTTP，具备先天优势，扩展能力很强。例如需要安全加密是，有现成的成熟方案HTTPS可用。

JSON 报文序列化，轻量简单，人与机器均可读，学习成本低，搜索引擎友好。

语言无关，各大热门语言都提供成熟的Restful API框架，相对其他的一些RPC框架生态更完善。

当然在有些特殊业务场景下，也需要采用其他的RPC框架，如thrift、grpc。但绝大多数情况下Restful就足够用了。

# 3SpringCloud介绍

什么是 SpringCloud：是一个服务治理平台，提供了一些服务框架。包含了：服务注册

与发现、配置中心、消息中心 、负载均衡、数据监控等等。

## 3.1概念定义

Spring Cloud 是一个微服务框架，相比 Dubbo 等 RPC 框架, Spring Cloud 提供的全套的分布式系统解决方案。

Spring Cloud 对微服务基础框架 Netflix 的多个开源组件进行了封装，同时又实现

了和云端平台以及和 Spring Boot 开发框架的集成。

Spring Cloud 为微服务架构开发涉及的配置管理，服务治理，熔断机制，智能路由，

微代理，控制总线，一次性 token，全局一致性锁，leader 选举，分布式 session，集

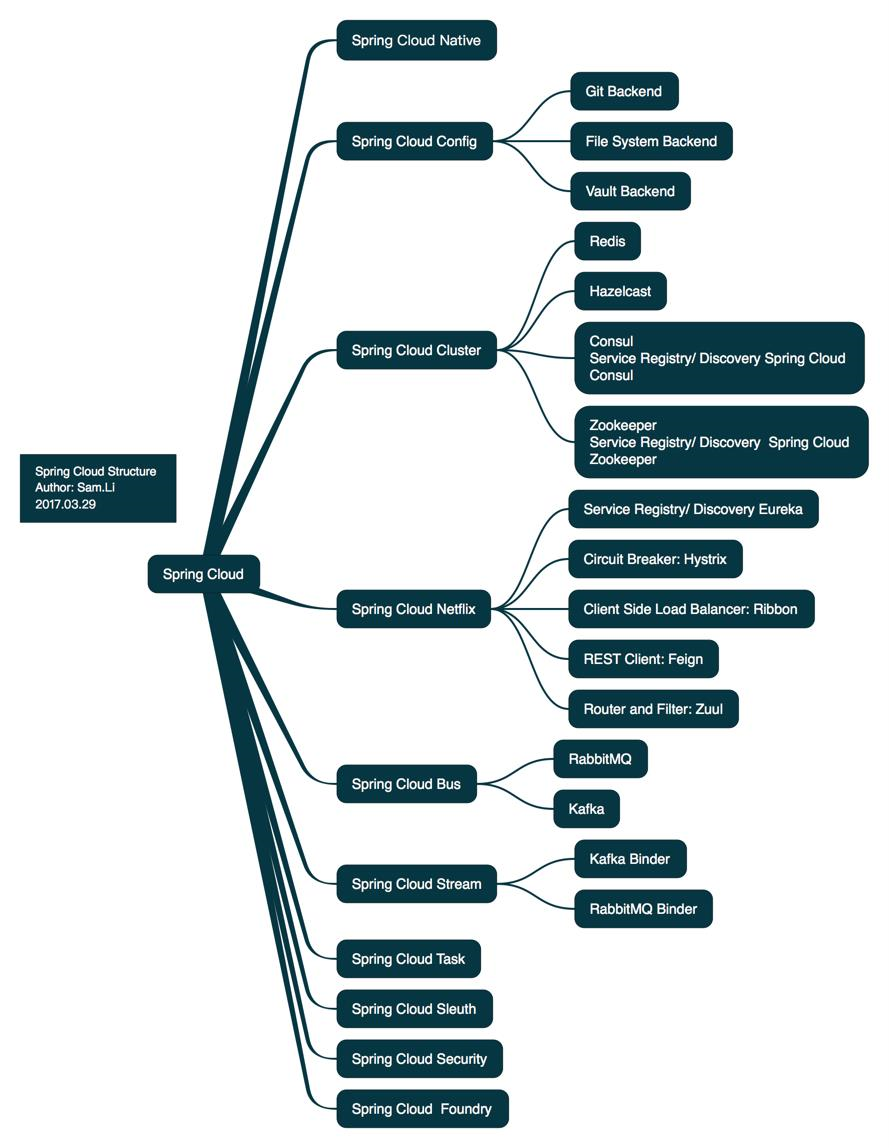
群状态管理等操作提供了一种简单的开发方式。

Spring Cloud 为开发者提供了快速构建分布式系统的工具，开发者可以快速的启动

服务或构建应用、同时能够快速和云平台资源进行对接。

Sping Cloud 是 Spring的一个顶级项目与 Spring Boot、Spring Data 位于同一位置。

## 3.2Spring Cloud 的子项目



### 3.2.1Spring Cloud Config

配置管理工具，支持使用 Git 存储配置内容，支持应用配置的外部化存储，支持客户端配置信息刷新、加解密配置内容等

### 3.2.2 Spring Cloud Bus

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。

### 3.2.3Spring Cloud Netflix

针对多种 Netflix 组件提供的开发工具包，其中包括

Eureka、Hystrix、Zuul、Archaius 等。

### 3.2.4 Netflix Eureka

一个基于 rest 服务的服务治理组件，包括服务注册中心、服务注册与服务发现机制的实现，实现了云端负载均衡和中间层服务器的故障转移。

### 3.2.5 Netflix Hystrix

容错管理工具，实现断路器模式，通过控制服务的节点, 从而对延迟和故障提供更强大的容错能力。

### 3.2.6 Netflix Ribbon

客户端负载均衡的服务调用组件。

### 3.2.7 Netflix Feign

基于 Ribbon 和 Hystrix 的声明式服务调用组件.

### 3.2.8 Netflix Zuul

微服务网关，提供动态路由，访问过滤等服务。

### 3.2.9Netflix Archaius

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。

台。

### 3.2.10Spring Cloud Sleuth

日志收集工具包，封装了 Dapper,Zipkin 和 HTrace 操作。

### 3.2.11Spring Cloud Consul

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。

### 3.2.12Spring Cloud Zookeeper

操作 Zookeeper 的工具包，用于使用 zookeeper 方式的服务注册和发现。

### 3.2.13Spring Cloud Stream

数据流操作开发包，封装了与 Redis,Rabbit、Kafka 等发送接收消息。

## 3.3Spring Cloud 版本说明

软件版本号：2.1.7.RELEASE

2：主版本号。当功能模块有较大更新或者整体架构发生变化时，主版本号会更新

0：次版本号。次版本表示只是局部的一些变动。

2：修改版本号。一般是 bug 的修复或者是小的变动

RELEASE:希腊字母版本号。次版本号用户标注当前版本的软件处于哪个开发阶段

### 3.3.1希腊字母版本号

Base：设计阶段。只有相应的设计没有具体的功能实现。

Alpha：软件的初级版本。存在较多的 bug

Bate：表示相对 alpha 有了很大的进步，消除了严重的 bug，还存在一些潜在的 bug。

Release：该版本表示最终版。

### 3.3.2Spring Cloud 版本号说明

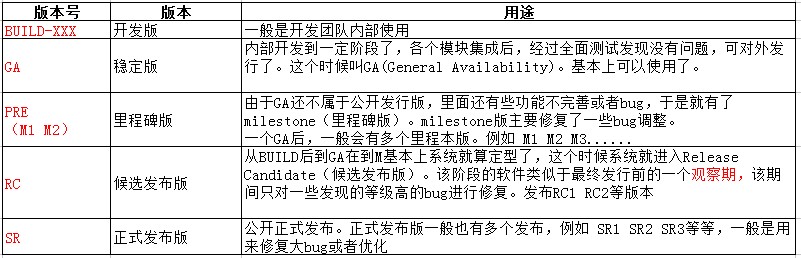
Spring Cloud 版本用的是单词:

设计的目的是为了更好的管理每个 Spring Cloud 的子项目的清单。避免子的版本号与子项目的版本号混淆。

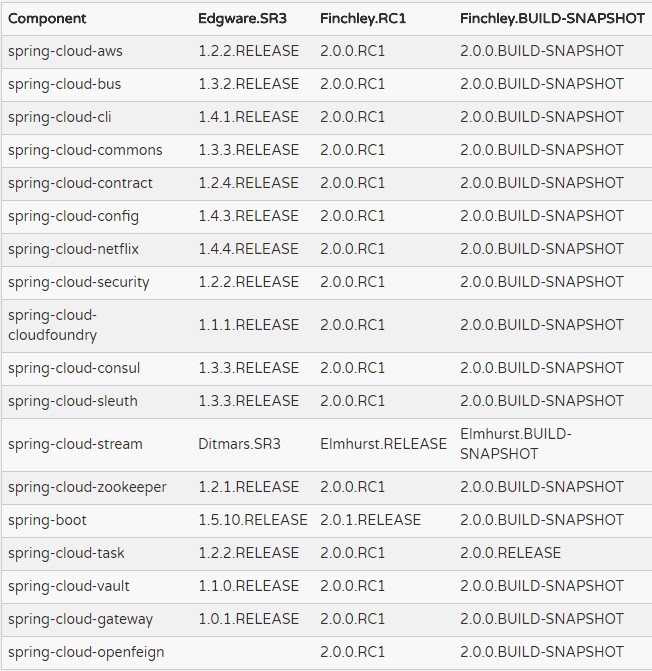
版本号单词的定义规则:

采用伦敦的地铁站名称来作为版本号的命名，根据首字母排序，字母顺序靠后的版本号越大。

版本发布计划说明:



### 3.3.3 Spring Cloud 与子项目版本兼容说明



# 4Springboot补充

## 4.1 logback 日志记录说明

将logback.xml放入到src/main/resources

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <configuration>  <!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径-->  <property name=*"LOG\_HOME"* value=*"${catalina.base}/logs/"* />  <!-- 控制台输出 -->  <appender name=*"Stdout"* class=*"ch.qos.logback.core.ConsoleAppender"*>  <!-- 日志输出编码 -->  <layout class=*"ch.qos.logback.classic.PatternLayout"*>  <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->  <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n  </pattern>  </layout>  </appender>  <!-- 按照每天生成日志文件 -->  <appender name=*"RollingFile"* class=*"ch.qos.logback.core.rolling.RollingFileAppender"*>  <rollingPolicy class=*"ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>  <!--日志文件输出的文件名-->  <FileNamePattern>${LOG\_HOME}/server.%d{yyyy-MM-dd}.log</FileNamePattern>  <MaxHistory>30</MaxHistory>  </rollingPolicy>  <layout class=*"ch.qos.logback.classic.PatternLayout"*>  <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->  <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n  </pattern>  </layout>  <!--日志文件最大的大小-->  <triggeringPolicy class=*"ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy"*>  <MaxFileSize>10MB</MaxFileSize>  </triggeringPolicy>  </appender>  <!-- 日志输出级别 -->  <root level=*"info"*>  <appender-ref ref=*"Stdout"* />  <appender-ref ref=*"RollingFile"* />  </root>  <!--日志异步到数据库 -->  <!-- <appender name="DB" class="ch.qos.logback.classic.db.DBAppender">  日志异步到数据库  <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">  连接池  <dataSource class="com.mchange.v2.c3p0.ComboPooledDataSource">  <driverClass>com.mysql.jdbc.Driver</driverClass>  <url>jdbc:mysql://127.0.0.1:3306/databaseName</url>  <user>root</user>  <password>root</password>  </dataSource>  </connectionSource>  </appender> -->  </configuration> |

## 4.2SpringBoot多文件配置

profile:代表的就是一个环境变量

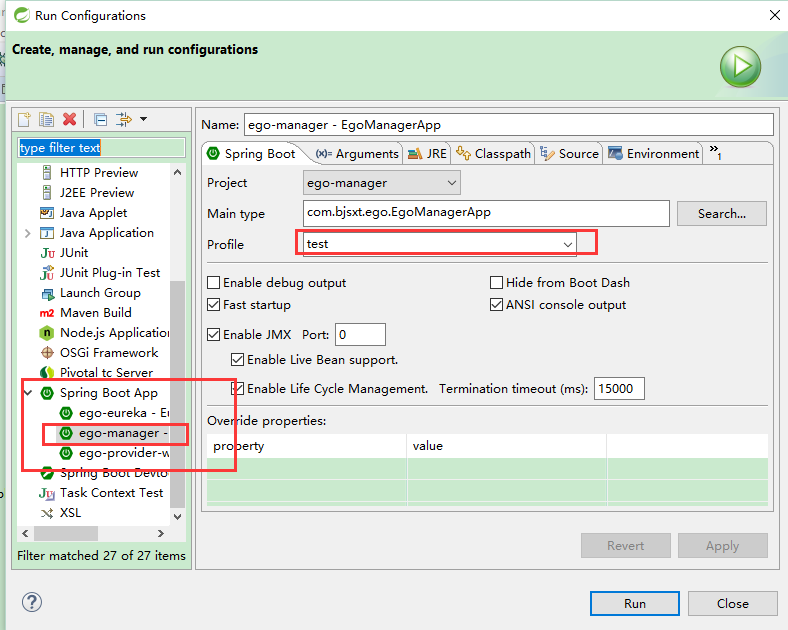
语法结构：application-{profile}.properties

可以为不同的环境定义不同的配置文件比如：

application-dev.properties 开发环境

application-test.properteis 测试环境

application-prod.properteis 生产环境



# 5 Eureka 服务注册中心

## 5.1什么是服务注册中心

服务注册中心是服务实现服务化管理的核心组件,类似于目录服务的作用,主要用来存储服务信息，譬如提供者 url 串、路由信息等。服务注册中心是 SOA 架构中最基础的设施之一。

## 5.2服务注册中心的作用

实现服务注册和服务发现。

服务的依赖关系管理

## 5.3常见服务注册中心

Dubbo 的注册中心 Zookeeper

Springcloud 的注册中心 EUREKA

## 5.4Eureka注册中心

Eureka 是 Netflix 开发的服务发现组件，本身是一个基于 REST 的服务。Spring Cloud 将它集成在其子项目 spring-cloud-netflix 中，以实现 Spring Cloud 的服务注册于发现，同时还提供了负载均衡、故障转移等能力。

## 5.5 Eureka 注册中心三种角色

Eureka Server 通过 Register、Get、Renew 等接，口提供服务的注册和发现。

Service Provider服务提供方把自身的服务实例注册到 Eureka Server 中

Service Consumer服务调用方通过 Eureka Server 获取服务列表，消费服务。

## 5.6Eureka简单入门

## 5.7Eureka高可用集群

## 5.8Eureka集群部署到Linux

部署环境：需要安装 jdk1.8，正确配置环境变量。注意：需要关闭 linux 的防火墙，或者是开放 8761 端口。

### 5.8.1Eureka注册中心打包

Mvn install

### 5.8.2上传Eureka.jar

在/usr/local/创建一个 eureka 的目录

将项目的 jar 包拷贝到/usr/local/eureka

### 5.8.3上传启动脚本

server.sh

|  |
| --- |
| #!/bin/bash    cd `dirname $0`    CUR\_SHELL\_DIR=`pwd`  CUR\_SHELL\_NAME=`basename ${BASH\_SOURCE}`    JAR\_NAME="项目名称"  JAR\_PATH=$CUR\_SHELL\_DIR/$JAR\_NAME    #JAVA\_MEM\_OPTS=" -server -Xms1024m -Xmx1024m -XX:PermSize=128m"  JAVA\_MEM\_OPTS=""    SPRING\_PROFILES\_ACTIV="-Dspring.profiles.active=配置文件变量名称"  #SPRING\_PROFILES\_ACTIV=""  LOG\_DIR=$CUR\_SHELL\_DIR/logs  LOG\_PATH=$LOG\_DIR/${JAR\_NAME%..log    echo\_help()  {  echo -e "syntax: sh $CUR\_SHELL\_NAME start|stop"  }    if [ -z $1 ];then  echo\_help  exit 1  fi    if [ ! -d "$LOG\_DIR" ];then  mkdir "$LOG\_DIR"  fi    if [ ! -f "$LOG\_PATH" ];then  touch "$LOG\_DIR"  fi    if [ "$1" == "start" ];then    # check server  PIDS=`ps --no-heading -C java -f --width 1000 | grep $JAR\_NAME | awk '{print $2}'`  if [ -n "$PIDS" ]; then  echo -e "ERROR: The $JAR\_NAME already started and the PID is ${PIDS}."  exit 1  fi    echo "Starting the $JAR\_NAME..."    # start  nohup java $JAVA\_MEM\_OPTS -jar $SPRING\_PROFILES\_ACTIV $JAR\_PATH >> $LOG\_PATH 2>&1 &    COUNT=0  while [ $COUNT -lt 1 ]; do  sleep 1  COUNT=`ps --no-heading -C java -f --width 1000 | grep "$JAR\_NAME" | awk '{print $2}' | wc -l`  if [ $COUNT -gt 0 ]; then  break  fi  done  PIDS=`ps --no-heading -C java -f --width 1000 | grep "$JAR\_NAME" | awk '{print $2}'`  echo "${JAR\_NAME} Started and the PID is ${PIDS}."  echo "You can check the log file in ${LOG\_PATH} for details."    elif [ "$1" == "stop" ];then    PIDS=`ps --no-heading -C java -f --width 1000 | grep $JAR\_NAME | awk '{print $2}'`  if [ -z "$PIDS" ]; then  echo "ERROR:The $JAR\_NAME does not started!"  exit 1  fi    echo -e "Stopping the $JAR\_NAME..."    for PID in $PIDS; do  kill $PID > /dev/null 2>&1  done    COUNT=0  while [ $COUNT -lt 1 ]; do  sleep 1  COUNT=1  for PID in $PIDS ; do  PID\_EXIST=`ps --no-heading -p $PID`  if [ -n "$PID\_EXIST" ]; then  COUNT=0  break  fi  done  done    echo -e "${JAR\_NAME} Stopped and the PID is ${PIDS}."  else  echo\_help  exit 1  fi |

### 5.8.4修改server.sh权限

chmod -R 755 server.sh

### 5.8.5修改Linux的hosts文件

vim /etc/hosts

192.168.70.134 eureka1

192.168.70.135 eureka2

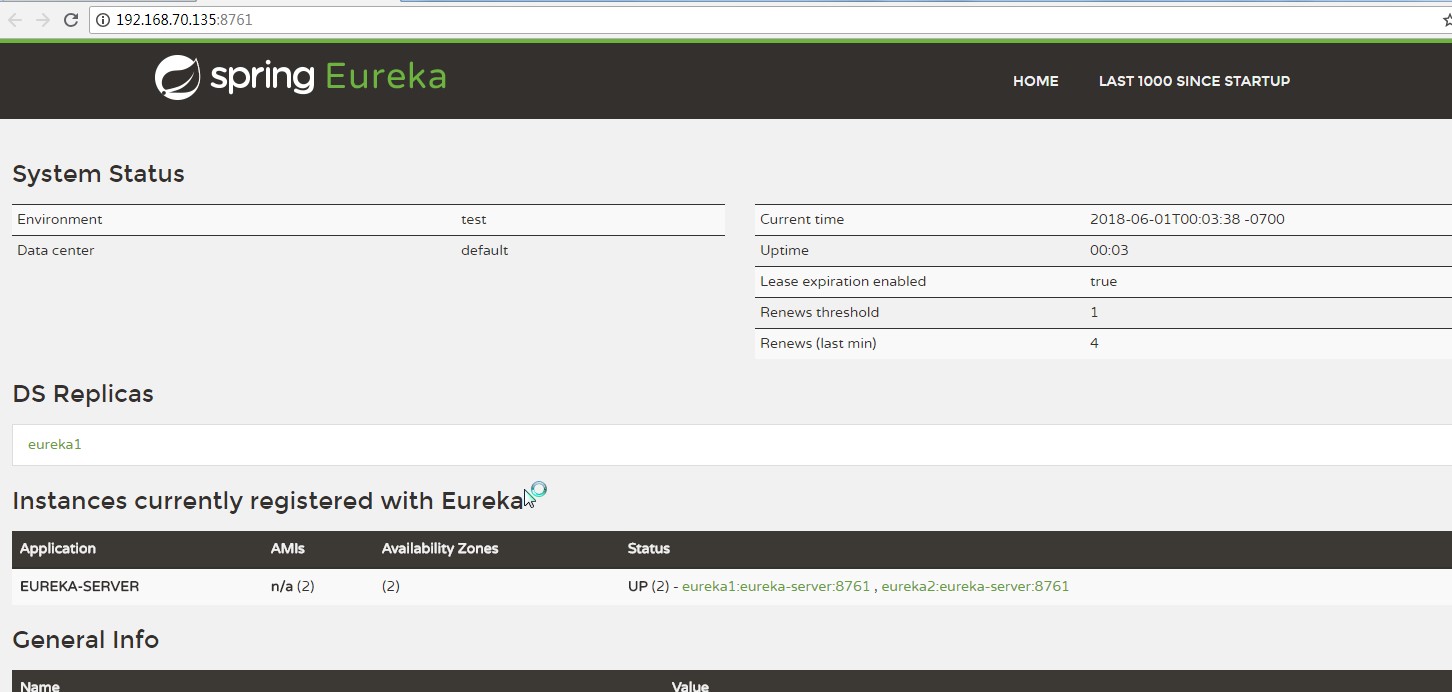
### 5.8.6启动 eureka 注册中心

./server.sh start 启动

./server.sh stop 停止

### 5.8.7关闭防火墙

### 5.8.8访问注册中心



### 5.8.9window中访问注册中心

eureka1:1111

eureka2:2222

注意：window的hosts文件中需要配置

192.168.70.134 eureka1

192.168.70.135 eureka2