微服务之服务发现开发规范

# 前言

本文主要介绍服务发现开发规范文档，属于开发中必须要配置的最为基础的部分。看本文之前要先了解微服务和Spring Cloud，了解Spring Cloud应该具备哪些要点，有个系统的了解之后再看本文，本文档会持续根据项目需求更新。

如果想了解全部的Spring Cloud架构，需按照演示PPT中介绍的逐步深入了解。本文档中介绍的内容是按照真实可运行的Demo Service Discovery工程编写，会持续更新本文文档。

# 版本更新日志

| 作者 | 版本 | 时间 | 备注 |
| --- | --- | --- | --- |
| 朱军军 | 1.0 | 20150828 | 初次编写 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 目录

微服务之服务发现开发规范 1

前言 2

版本更新日志 3

目录 4

Spring Boot 2.0 5

Spring Cloud Finchley 5

Cloud Plugins 5

Configuration Center 5

Registration Center 5

Service Discovery 5

API Gateway 5

Start Spring IO 6

Develop Consul Service Discovery 6

Maven构建工程 6

Spring Cloud插件 6

Docker插件 6

Spring Cloud配置 7

微服务之间调用 8

外部调用微服务 8

# Spring Boot 2.0

开发Spring微服务入门必学框架，Spring提供Spring Cloud实现微服务，而Spring Cloud是基于Spring Boot实现的。本文档不包括如何使用Spring Boot，可访问Spring官网（https://projects.spring.io/spring-boot/）学习如何使用Spring Boot，如果Spring Boot自学都学不会，那么Spring Cloud就更别想会了，Spring Cloud组件是开箱即用，简化了微服务开发，但别误以为就很简单，门槛很高的，你跨过去了门槛才是第一步。

Spring Boot 2.0与Spring Boot 1.0版本有很大区别，当然核心思想是一致的。说的区别在于很多底层实现代码有很大改动，新增了很多功能，也删除了很多功能，当然也优化了很多功能。由于Spring的核心编程思想就是整合不重复造轮子，所以把1.0版本中某些特殊配置和功能也做了整合，比如Actuator。

相对的Spring Cloud也跟以前的配置和开发也是有所改变的。比如说配置选项，Spring Boot 2.0中的Cloud配置选项重新做了归纳，注意一点Spring Cloud版本不同选择的Spring Boot版本也是不同的。官方提供的[https://start.spring.io](http://start.spring.io)会帮助你选择对应的版本。

# Spring Cloud Finchley

这个版本基于Spring Boot 2.0开发的Spring Cloud框架，也是我们现在环境用的版本，完成版本号为: Spring Cloud Finchley.M9. 官方文档：<https://projects.spring.io/spring-cloud/>

## Cloud Plugins

之前有提过，Spring Cloud是开箱即用的插件开发，也就是说它集成了很多插件，比如：注册中心，服务发现，配置中心，企业总线等等。在我们的开发文档中，我们目前阶段用的插件主要有consul，ribbon, feign, turbine, hystrix, zipkin, bus, data jpa.随着版本的迭代会加入更多的插件.

## Configuration Center

配置中心已经开发好了，提供微服务分布式配置管理功能，所有服务发现开发中必须要配置配置中心地址，去获取配置文件，这里不做过多介绍。

## Registration Center

注册中心使用了Consul，直接安装Consul就可以，服务发现开发中需要配置注册中心地址，以实现注册服务功能。

## Service Discovery

服务发现开发才是本文需要介绍的，二次开发中服务发现才是实现业务的模块，是需要迭代开发的，企业级开发中，服务发现的开发量是巨大的，通常情况下，企业开发也只有这块是open出来供供应商注册开发，配置中心、注册中心是不会open出来随意更改的。本文制定了服务发现的开发规范和标准。

## API Gateway

Spring Cloud提供了两种Gateway：zuul和Cloud Gateway。我们开发环境使用的是zuul实现的API Gateway。通常情况下，企业内部会有多个API Gateway。会根据接口性质，业务性质，网络性质不同，会创建多个API Gateway工程。

# Start Spring IO

[https://start.spring.io](http://start.spring.io)提供了在线创建Spring Boot工程，可以快速构建Spring Boot工程，避免版本兼容问题，但是在实际项目中不能依赖该在线平台。因为实际应用中很多插件和jar是这个平台提供不了的。

# Develop Consul Service Discovery

## Maven构建工程

### Spring Cloud插件

添加如下插件，这些都是最基础的配置插件，针对不同的业务比如访问数据库操作，需要额外加载Data JPA插件。

|  |  |
| --- | --- |
| spring-boot-starter-actuator | 系统的自检，监控，统计，查看配置功能 |
| Spring-boot-starter-web | 提供web功能，比如restful |
| Spring-boot-starter-config | 客户端配置功能，实现读取配置中心配置 |
| spring-cloud-starter-consul-discovery | Consul服务发现 |
| spring-cloud-starter-netflix-ribbon | Ribbon负载均衡 |
| spring-cloud-starter-openfeign | Feign |
| spring-cloud-starter-zipkin | Zipkin客户端 |
| spring-cloud-starter-bus-amqp | RabbitMQ，实现MQ队列这里实现了动态加载配置 |
| spring-cloud-starter-netflix-hystrix | 实现断融 |
| spring-boot-starter-test | 实现测试 |

### Docker插件

|  |
| --- |
| <plugin>  <groupId>com.spotify</groupId>  <artifactId>dockerfile-maven-plugin</artifactId>  <version>1.3.6</version>  <configuration>  <repository>${docker.image.prefix}/${project.artifactId}</repository>  <buildArgs>  <JAR\_FILE>target/${project.build.finalName}.jar</JAR\_FILE>  </buildArgs>  </configuration>  </plugin> |

工程支持Docker发布需引用Docker插件，还要配置编写Dockerfile文件（如下），发布远程Docker需要设置环境变量。

|  |
| --- |
| FROM openjdk:8-jdk-alpine  VOLUME /tmp  ARG JAR\_FILE  ADD ${JAR\_FILE} app.jar  EXPOSE 32001  ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"] |

设置环境变量（或JVM变异参数），2375是Docker默认端口。

DOCKER\_HOST=<tcp://192.168.1.101:2375>

编译命令：

mvn clean install -DskipTests dockerfile:build

docker run -d -p 35001:35001 -t springcloud/configuration-server

springcloud/Configuration为镜像名称，springcloud为Maven docker插件中${docker.image.prefix}中设置的前缀。

相关docker命令需要了解。

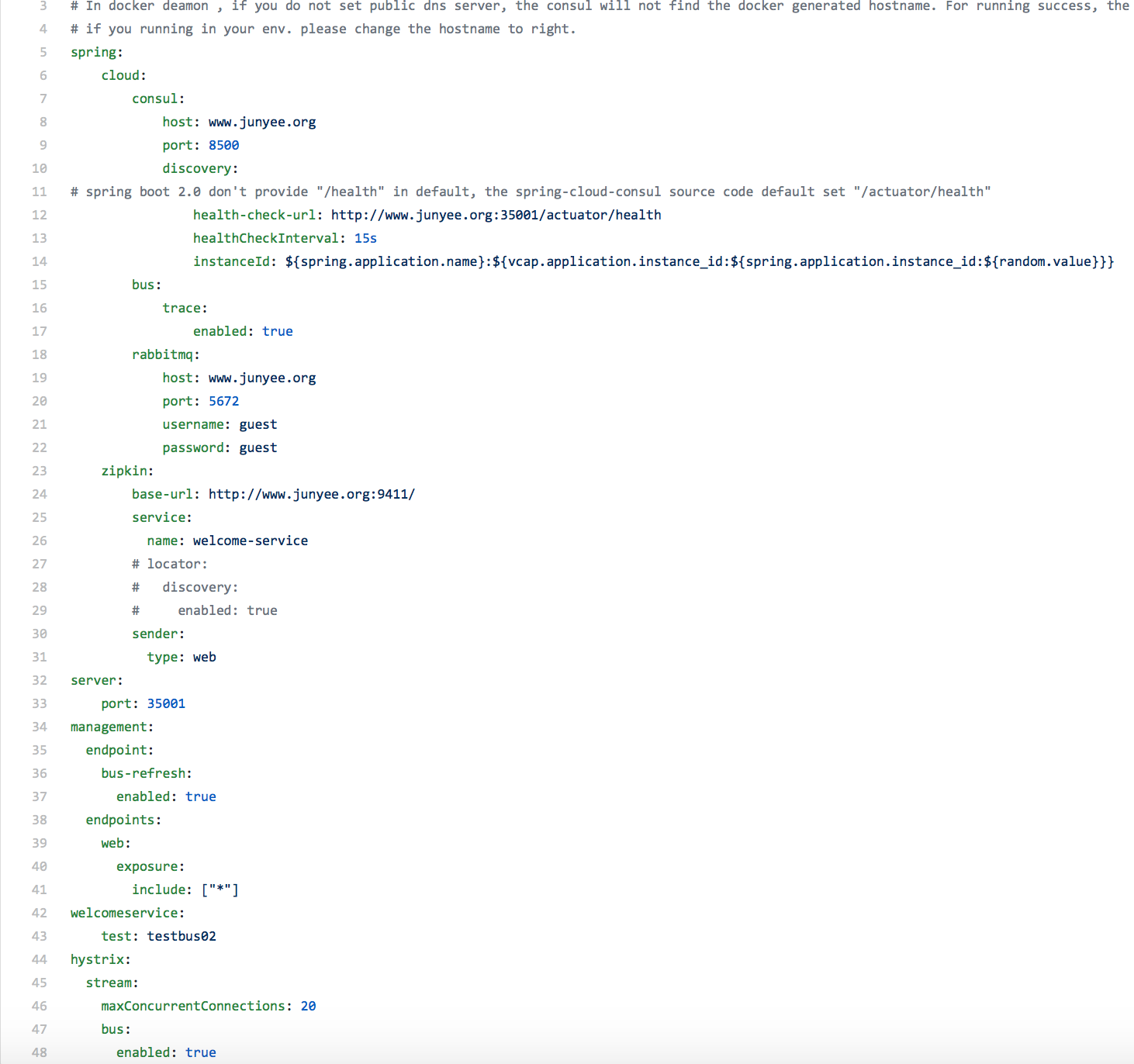
## Spring Cloud配置

Spring Cloud的相关配置文件都是配置在配置中心Git内，所以我们工程内只需要配置配置中心的地址，业务方面配置无需在工程内配置。如下为：bootstrap.yml配置，所有application.yml配置都必须配置在Configuration Center内。

|  |
| --- |
| **spring:**  **application:**  **name:** api-gateway  **cloud:**  **config:**  **profile:** dev  **uri:** http://www.junyee.org:8888/ |

关于多环境配置是通过application-{profilename}.yml区分的，这是已经很成熟的多环境配置模式，不做深入说明。

Git内的配置按照项目实际需求配置，通常这部分工作在项目工程创建时，最好由核心开发团队配置基础配置选项，剩余部分按照实际业务自己配置。



## 微服务之间调用

微服务之间调用采用Feign调用。

## 外部调用微服务

外部系统分为内网和外网，无论哪种都要通过API-Gateway访问微服务。