# [7天学会spring cloud教程](https://www.cnblogs.com/skyblog/p/5127690.html)

　　按照官方的话说：Spring Cloud 为开发者提供了在分布式系统（如配置管理、服务发现、断路器、智能路由、微代理、控制总线、一次性 Token、全局锁、决策竞选、分布式会话和集群状态）操作的开发工具。最关键的是它足够简单，一般的开发人员只需要几天时间就可以学会它的基本用法。

　　本Spring Cloud 7天系列教程，包括7个例子和相关短文，都是最简单的用法，也是默认最基本的用法，在实际生产环境中也可以用上，当然是初步使用。

　　项目开源地址：<http://git.oschina.net/zhou666/spring-cloud-7simple>

     7个例子包括：

         1）一个基本的spring boot应用。

         2）分布式配置管理服务端

         3）分布式配置管理客户端（微服务应用）

         4）服务注册服务端

         5）服务注册发现客户端（微服务应用）

         6）spring boot风格的web前端应用

         7）使用docker发布应用

         7天学习周期如下：

         第1天：查看spring boot官方文档，实现及实验spring boot 应用。

         第2天：熟读spring cloud官方文档配置管理部分并熟悉配置管理相关概念。

         第3天：熟悉Git概念，并上传配置文件到Git服务器，最后实现分布式配置管理。

        第4天：熟读spring cloud官方文档服务注册部分，实现服务注册及发现。

        第5天：熟读spring cloud官方文档剩余部分，并实现断路器。

        第6天：深入spring boot相关概念，使用angularJS实现web前端应用。

        第7天：了解docker概念，并结合spring boot搭建一个docker应用。

**第一章《Spring Cloud之简介》**

微服务概念

        微服务是系统架构上的一种设计风格。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。被拆分成的每一个小型服务都围绕着系统中的某一项或一些耦合度较高的业务功能进行构建。并且每个服务都维护着自身的数据存储、业务开发、自动化测试案例以及独立部署机制。由于有了轻量级的通信协作基础，所以这些微服务可以使用不同的语言来编写。

特点

1. 是一种设计风格

2. 将一个独立的系统拆分成多个小型服务，各自独立运行。

3. 服务间基于HTTP的RESTful API进行通信。

4. 能够达到解耦的目的。

5. 各自维护自身的数据存储，能更加方便进行自动化测试。

与传统单体系统的比较

传统单体系统特点：通常将一个复杂庞大的系统构建一个单体项目。通常将需求分为三个主要部分：数据库、服务端处理、前端展现。

在业务发展初期，由于所有的业务逻辑在一个应用中，开发、测试、部署都还比较容易且方便。但是，随着企业的发展，系统为了应对不同的业务需求会不断为该单体项目增加不同的业务模块；同时随着移动端设备的进步，前端展现模块已经不仅仅局限于Web的形式，这对于系统后端向前端的支持需要更多的接口模块。单体应用由于面对的业。不断扩大的需求会使得单体应用变得越来越臃肿。单体应用的问题就逐渐凸显出来，由于单体系统部署在一个进程内，往往我们修改了一个很小的功能，为了部署上线会影响其他功能的运行。

并且，单体应用中的这些功能模块的使用场景、并发量、消耗的资源类型都各有不同，对于资源的利用又互相影响，这样使得我们对各个业务模块的系统容量很难给出较为准确的评估。但是随着系统的发展，维护成本会变得越来越大，且难以控制。

我们将系统中的不同功能模块拆分成多个不同的服务，这些服务都能够独立部署和扩展。由于每个服务都运行在自己的进程内，在部署上有稳固的边界，这样每个服务的更新都不会影响其他服务的运行。同时，由于是独立部署的，我们可以更准确地为每个服务评估性能容量 ，通过配合服务间的协作流程也可以更容易地发现系统的瓶颈位置，以及给出较为准确的系统级性能容量评估。

微服务框架的选择

服务治理： 阿里巴巴开源的\*\*Dubbo\*\*和当当网在其基础上扩展的\*\*DubboX\*\*、Netflix的Eureka、Apache的Consul等。

分布式配置管理： 百度的Disconf、Netflix的Archaius、360的QConf、Spring Cloud的Config、淘宝的Diamond等。

批量任务： 当当网的Elastic-Job、LinkedIn的Azkaban、Spring Cloud的Task等。

服务跟踪： 京东的Hydra、Spring Cloud的Sleuth、Twitter的Zipkin等。

spring Cloud的出现，可以说是对微服务架构的巨大支持和强有力的技术后盾。它不像我们之前所列举的框架那样，只是解决微服务中的某一个问题，而是一个解决微服务架构实施的\*\*综合性解决框架\*\*，它整合了诸多被广泛实践和证明过的框架作为实施的基础部件，又在该体系基础上创建了一些非常优秀的边缘组件。

**第二章《Spring Cloud 之服务治理Spring Cloud Eureka 》**

什么是服务治理？要用来干什么

作用

它主要用来实现各个微服务实例的自动化注册与发现。在最初开始构建微服务系统的时候可能服务并不多，我们可以通过做一些静态配置来完成服务的调用。比如，有两个服务A和B，其中服务A需要调用服务B来完成一个业务操作时，为了实现服务B的高可用，不论采用服务端负载均衡还是客户端负载均衡，都需要手工维护服务B的具体实例清单。但是随着业务的发展，系统功能越来越复杂，相应的微服务应用也不断增加，我们的静态配置就会变得越来越难以维护。并且面对不断发展的业务，我们的集群规模、服务的位置、服务的命名等都有可能发生变化\*\*，如果还是通过手工维护的方式，那么极易发生错误或是命名冲突等问题。同时，对于这类静态内容的维护也必将\*\*在最初开始构建微服务系统的时候可能服务并不多，我们可以通过做一些静态配置来完成服务的调用。比如，有两个服务A和B，其中服务A需要调用服务B来完成一个业务操作时，为了实现服务B的高可用，不论采用服务端负载均衡还是客户端负载均衡，\*\*都需要手工维护服务B的具体实例清单\*\*。但是随着业务的发展，系统功能越来越复杂，相应的微服务应用也不断增加，\*\*我们的静态配置就会变得越来越难以维护\*\*。并且面对不断发展的业务，\*\*我们的集群规模、服务的位置、服务的命名等都有可能发生变化\*\*，如果还是通过手工维护的方式，那么极易发生错误或是命名冲突等问题。同时，对于这类静态内容的维护也必将消耗大量的人力。

为了解决微服务架构中的服务实例维护问题，产生了大量的服务治理框架和产品。这些框架和产品的实现都围绕着服务注册与服务发现机制来完成对微服务应用实例的自动化管理。

服务注册

在服务治理框架中，通常都会构建一个注册中心，每个服务单元向注册中心登记自己提供的服务，将主机与端口号、版本号、通信协议等一些附加信息告知注册中心，注册中心按服务名分类组织服务清单。

服务发现

由于在服务治理框架下运作，服务间的调用不再通过指定具体的实例地址来实现，而是通过向服务名发起请求调用实现。所以，服务调用方在调用服务提供方接口的时候，并不知道具体的服务实例位置。因此，调用方需要向服务注册中心咨询服务，并获取所有服务的实例清单，以实现对具体服务实例的访问。实际的框架为了性能等因素，不会采用每次都向服务注册中心获取服务的方式，并且不同的应用场景在缓存和服务剔除等机制上也会有一些不同的实现策略。

开发步骤

由于是分布式开发，我们可以通常创建一个父工程，子工程为各个微服务。

父工程的pom.xml文件

部分文件省略。需要源码可以上我的github上下载。

<dependencyManagement>

<dependencies>

<!--统一版本，不会出现版本兼容错误-->

<dependency>

<groupId>io.spring.platform</groupId>

<artifactId>platform-bom</artifactId>

<version>Brussels-SR10</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<!--引入spring cloud-->

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Edgware.SR3</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.3</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

</plugins>

</build>

<modules>

<module>demo-eureka</module>

<module>demo-user</module>

</modules>

创建注册中心服务eureka

pom.xml文件

<dependencies>

<!--eureka server -->

<dependency>

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

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

<!-- spring boot test-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

application.yml

#server config

server:

port: 8761

session:

timeout: 1800

# eureka config

eureka:

instance:

hostname: localhost

client:

# 由于该应用为注册中心，所以设置为false，代表不向注册中心注册自己

registerWithEureka: false

# 由于注册中心的职责就是维护服务实例，它并不需要去检索服务，所以也设置为false

fetchRegistry: false

serviceUrl:

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

启动类

@SpringBootApplication

@EnableEurekaServer

public class EurekaApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaApplication.class, args);

}

}

运行界面

打开浏览器，输入http://localhost:8761。如果出现以下界面表示配置成功。

# 第三章《Spring Cloud 之配置中心Spring Cloud Config》

spring config

概念

图解

快速开始

config server

config client

spring config

概念

Spring Cloud Config为分布式系统中的外部化配置提供服务器和客户端支持。通过Config Server，您可以在所有环境中管理应用程序的外部属性。客户端和服务器上的概念与Spring Environment和PropertySource抽象，因此它们非常适合Spring应用程序，但可以与任何运行在任何语言的应用程序一起使用。随着应用程序从开发到测试转移到部署管道中，您可以管理这些环境之间的配置，并确保应用程序具有迁移时所需的所有内容。服务器存储后端的默认实现使用git，因此它可以轻松支持配置环境的标签版本，并且可以用于管理内容的各种工具。使用Spring配置很容易添加替代实现并将其插入。在微服务架构中，需要用到两个角色，一个是config server,一个是config client。而config client 就是如订单服务，用户服务等。

图解

快速开始

config server

创建一个spring boot项目。该项目的pom.xml文件所依赖的jar包如下。

<dependencies>

<dependency>

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

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<!--config服务需要用这个包-->

<dependency>

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

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

application.yml文件如下

#server config

server:

port: 7600

session:

timeout: 1800

#

spring:

application:

name: demo-config

cloud:

config:

server:

git:

#配置git仓库地址

uri: https://gitee.com/lvhaibao/springcloud-demo-config.git

#配置仓库路径

search-paths: respo

#如果Git仓库为公开仓库，可以不填写用户名和密码，如果是私有仓库需要填写

username:

password:

#配置仓库的分支

label: master

# eureka 注册的配置，把该服务注册到服务中心

eureka:

client:

serviceUrl:

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

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

创建spring boot启动类。只需要加上@EnableConfigServer注解即可

package org.lhb.demo.config;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.cloud.config.server.EnableConfigServer;

/\*\*

\* @Author lvhaibao

\* @Date 2018/6/13 22:26

\*/

@EnableConfigServer

@SpringBootApplication

public class ConfigController {

public static void main(String[] args){

new SpringApplicationBuilder(ConfigController.class).web(true).run(args);

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

启动应用程序，在浏览器打开http://localhost:7600/foo/dev

如出现以下

{"name":"foo","profiles":["dev"],"label":"master",

"version":"792ffc77c03f4b138d28e89b576900ac5e01a44b","state":null,"propertySources":[]}

1

2

则表示配置成功。

config client

在创建config client的时候，笔者就具体到用户服务中。服务名为demo-config。

1.创建spring boot项目。所需要依赖的包如下：

<dependencies>

<dependency>

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

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- 配置config client必须需要这个包 -->

<dependency>

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

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

bootstrap.xml文件如下：

由于用到统一配置服务。所以application.xml文件可以省略。采用bootstrap.xml文件

#server config

server:

port: 7300

session:

timeout: 1800

#

spring:

application:

name: demo-user

cloud:

# config 配置

config:

# 选择的分支

label: master

# 选择的开发版本

profile: dev

# config服务的地址

uri: http://localhost:7600/

# eureka config

eureka:

client:

serviceUrl:

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

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

启动类配置

package org.lhb.demo.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

/\*\*

\* @Author lvhaibao

\* @Date 2018/6/6 22:43

\*/

@SpringBootApplication

@EnableEurekaClient

public class UserApplication {

public static void main(String[] args){

SpringApplication.run(UserApplication.class,args);

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

在https://gitee.com/lvhaibao/springcloud-demo-config.git 创建一个名为demo-user-dev.yml文件。里面的配置如下：

aa: ddd

1

5.编写一个controller类。

package org.lhb.demo.user.controller;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* @Author lvhaibao

\* @Date 2018/6/6 22:45

\*/

@RestController

@RequestMapping("user")

public class UserController {

@Value("${server.port}")

private String serverPort;

@Value("${aa}")

private String aa;

@RequestMapping("/getAll")

public List<String> getAll(){

List<String> list = new ArrayList<>();

list.add("zhangsan");

list.add("lisi");

list.add("wangwu");

list.add("serverPort"+serverPort);

return list;

}

@RequestMapping("/getAa")

public Object getAa(){

return aa;

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

先启动config servr。然后启动demo-user服务。

在浏览器访问http://localhost:7300/user/getAa。

如果页面显示ddd.则表示配置成功。

参考资料：http://projects.spring.io/spring-cloud/spring-cloud.html#\_spring\_cloud\_config

---------------------

作者：海宝大大

来源：CSDN

原文：https://blog.csdn.net/baidu\_34389984/article/details/80863460

版权声明：本文为博主原创文章，转载请附上博文链接！