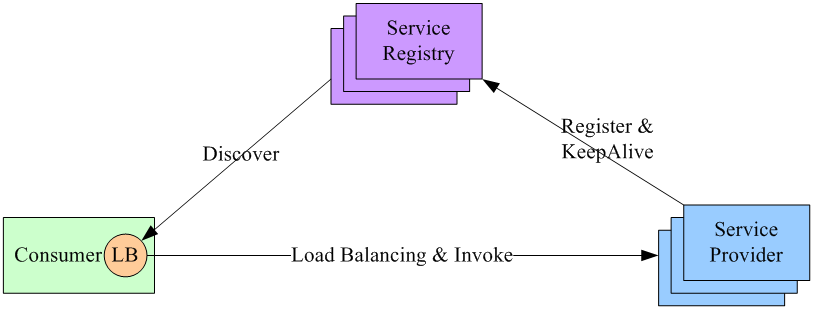
Springcloud

服务注册发现组件，这里介绍的是Netflix的技术——Eureka。Eureka分为服务端和客户端（如没有特别指明，下面所有的客户端指的是eureka客户端，而不是手机客户端等）。服务端即服务注册表，客户端即服务实例（即提供自身为服务，也可能消费其他服务）。首先存在一个服务端，客户端启动时，会根据配置信息，将自身服务信息（包括服务名、实例url地址等）注册到服务端，即声明了一个服务。当请求服务时，客户端首先向服务端查询指定服务的所有服务实例，然后根据一定的规则（轮询、随机等）获取到具体服务实例信息，并根据对应的url请求服务。如下图：



注意：在Eureka中消费者也是服务提供者，上图中Consumer和Service Provider都是客户端。

服务端和不同的客户端都是独立的项目进程。一个服务实例实质上就是一个运行的程序进程（ip+port定位）

# DEMO

## Parent:

在开始discovery程序前，先定义parent pom文件，引入一些公共的jar

<!-- spring cloud -->

<!-- 服务注册，也就是所有eureka客户端都需要该jar进行服务注册 -->

<dependency>

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

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

</dependency>

<!-- 配置文件/注解 注解配置、配置中心相关需要使用 -->

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

<!-- 服务选择负载均衡，两种不同的使用方式feign是在ribbon之上的封装 -->

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<!-- web/rest rest风格的controller层-->

<dependency>

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

<artifactId>spring-boot-starter-data-rest</artifactId>

</dependency>

<!-- spring boot的基础包，但是因为已经被包含在spring-boot-starter-data-rest中，这里就不进行重复引入

<dependency>

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

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

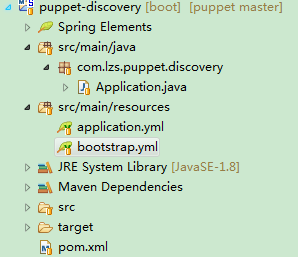
</dependency>

-->

上面包括了服务注册发现、配置相关、web层相关以及负载均衡相关的公共jar，当然也可以根据需要再添加或删除一些。

\*\*\*starter\*\*\* 是spring boot中的一些为了更好自动化配置定义的包，目的就是减少配置，只需要引入jar包并进行极少或者不需要任何配置，就可使用相应的组件。

## Discovery: 服务注册与提供



**pom.xml**

首先需要引入依赖：

<dependency>

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

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

</dependency>

**启动类**

编写启动类，非常简单，如下：

@SpringBootApplication

@EnableEurekaServer

**public** **class** Application {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Application.**class**, args);

}

}

因为spring boot内置了tomcat容器，因此不需要进行容器的配置并打成war放到容器中，直接启动主类即可。基本上所有的spring boot程序都是这样的启动方式：

SpringApplication.*run*(Application.**class**, args);

@SpringBootApplication表明这是一个spring boot应用，主要是为了让spring boot能够根据依赖引入的各种starter进行自动配置。@SpringBootApplication包含了

@Configuration

@EnableAutoConfiguration

@ComponentScan

这几个注解，因此能够自动将依赖中的相应组件加载进来。

@EnableEurekaServer表明这是discovery服务端应用，接受Eureka客户端的服务注册请求和服务获取请求。

**配置文件**

yml文件和properties类似，只是在呈现方式上更加直观结构化，并且yml文件配置加载是有上下先后顺序的。更多详情请google

Spring Cloud中，bootstrap优先application加载，并且这两个文件名都是特殊的文件名（感觉，没做过严谨的测试）。

**bootstrap.yml**

spring:

application:

name: discovery

profiles:

active: discovery1 #default active profile

这里定义了应用名和激活的配置profile标识（下面会讲到）

**application.yml**

spring:

profiles: discovery1

server:

port: ${PORT:8761}

eureka:

instance:

hostname: discovery1

preferIpAddress: **true** # default false, but strong recommend to set true

client:

registerWithEureka: **true**

fetchRegistry: **true**

serviceUrl:

defaultZone: http://discovery2:8762/eureka/,http://discovery3:8763/eureka/

---

spring:

profiles: discovery2

server:

port: ${PORT:8762}

eureka:

instance:

hostname: discovery2

preferIpAddress: **true**

client:

registerWithEureka: **true**

fetchRegistry: **true**

serviceUrl:

defaultZone: http://discovery1:8761/eureka/,http://discovery3:8763/eureka/

---

spring:

profiles: discovery3

server:

port: ${PORT:8763}

eureka:

instance:

hostname: discovery3

preferIpAddress: **true**

client:

registerWithEureka: **true**

fetchRegistry: **true**

serviceUrl:

defaultZone: http://discovery1:8761/eureka/,http://discovery2:8762/eureka/

---

spring:

profiles: discovery

server:

port: ${PORT:8761}

eureka:

instance:

hostname: discovery

client:

registerWithEureka: **false**

fetchRegistry: **false**

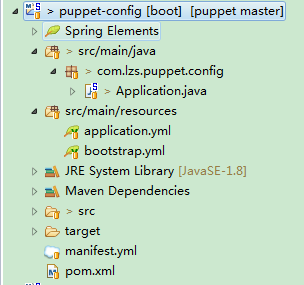
serviceUrl:

defaultZone: http://localhost:${server.port}/eureka/

上面有4种配置，通过---来分隔，相当于maven中的不同profile文件。我们之前在bootstrap.yml中激活的是profile=discovery的配置，这个配置中应用启动端口是server.port=8761，eureka实例hostname=discovery。${PORT:8761}表达式意思是：若存在参数PORT则使用PORT值，否则使用:号后面的值。hostname默认是主机名（计算机名），用于呈现，作用不大。eureka.instance还有许多的设置，都是一些服务实例信息，例如ip地址、实例id等。eureka.client 是服务注册和获取相关的配置，例如上面的registerWithEureka=false 表示自身不注册到eureka服务端（因为当前是discovery服务端，当然不注册自己到自己），默认true。fetchRegistry表示是否从eureka服务器获取注册信息，同理这里也是不需要的，默认true。defaultZone就比较重要了，是设置eureka服务器所在的地址，查询服务和注册服务都需要依赖这个地址。但是这个地址只对eureka客户端起作用，服务端不需要，所以这里设置defaultZone是无意义的。

另外的profile=discovery123 三种配置是集群的配置方式，例子中配置了3个discovery的服务端集群，服务端之间会定时进行信息同步。在集群模式下需要将自身注册到服务注册表registerWithEureka=true 以及能够感知获取服务fetchRegistry=true 并设置了不同的defaultZone，访问到另两个eureka服务器。

## Config: 配置中心



**pom.xml**

引入依赖：

<!-- 配置中心服务端 -->

<dependency>

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

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

</dependency>

**启动类**

@SpringBootApplication

@EnableDiscoveryClient

@EnableConfigServer

**public** **class** Application {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Application.**class**, args);

}

}

和discovery基本差不多，区别是@EnableConfigServer。表明当前应用是配置中心服务器。

**配置文件**

**bootstrap.yml**

spring:

application:

name: config

**application.yml**

server:

port: ${PORT:0}

eureka:

instance:

hostname: config

instanceId: ${spring.cloud.client.hostname}:${spring.application.name}:${server.port}-${random.value}

preferIpAddress: **true** # default false, but strong recommend to set true

client:

registerWithEureka: **true**

fetchRegistry: **true**

serviceUrl:

defaultZone: http://discovery:8761/eureka/ #,http://discovery1:8761/eureka/,http://discovery2:8762/eureka/,http://discovery3:8763/eureka/

spring:

cloud:

config:

server:

git:

uri: https://github.com/lizhaosheng/springcloud-demo.git

searchPaths: config-repo

大致和discovery差不多，注意到eureka.instance.instanceId 的值是一个很长的表达式。instanceId是服务实例注册id，要求不同的服务id必须不同，否则前面的相同id的服务将被后面的覆盖。instanceId值表达式中${spring.application.name}即bootstrap.yml文件中的对应值，${server.port}即application.yml的对应值，${random.value}是系统随机变量。可见，在yml文件中可以使用前面或者比当前文件先加载的文件中的值。

eureka.instance.preferIpAddress=true 表示在服务注册时优先使用ip地址（更多的是显示时以及获取服务信息时），这时即使设置了hostname，实际获取（代码或者eureka服务端界面看到的）到的hostname也是ip，默认preferIpAddress=false，hostname 不设置则是主机名。

defaultZone设置服务注册以及获取的服务端地址（也就是discovery），注解掉的红色部分是集群的配置方式。

spring.cloud.config.server是配置中心服务器的配置，这里配置git方式获取，uri是git仓库地址，searchPaths是搜索路径。

**扩展：**

所有设置了从配置中心服务器获取配置的应用，在启动后会首先连接到配置中心服务器下载配置文件，并将下载的配置文件和本地的配置文件进行合并，然后再根据合并的配置文件开始程序的初始化的。合并同名配置本地配置优先远程配置。

## Ribbon-server: 服务提供

## Ribbon-client: 服务消费

## Gateway: 网关

# 各种自动化配置组件

## Jdbc 数据源-mysql

## Mybatis 数据访问层

## Redis 缓存

## Rabbitmq 消息队列

## Freemark 视图模板

注意controller层不能使用@RestController 注解，因为该注解包含@ResponseBody 返回json格式的数据，并非字符串视图名。

## Spring session 分布式session方案

# 实践

## 传统Spring MVC项目迁移