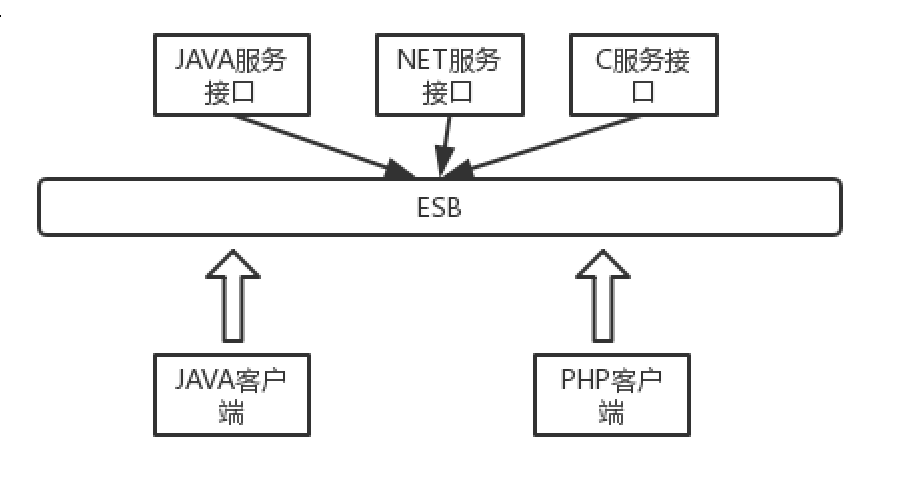
# RestFul简介

10几年前还是WebService的天下，但用过WebService的同学可能都知道，它使用的是WSDL文件描述的SOAP协议进行通讯了，这里重点不是讲WebService但在我们今天要讲到的RestFul却是基于WebService演进而来。

WebService的问题大概如下：

* 由于使用的SOAP协议，使用WSDL，这本质上是使用的XML进行内容通信，速度太慢，处理的效率太低
* 如果想使用本地接口的方式调用，要利用开发工具根据WSDL文件生成很多工具代码，接口任何变动都回导致工具代码重新生成，开发特别繁琐

在2005年的时候，又有一个新的概念产生，既SOA(面向服务架构)，他提出了一个服务总线(ESB)的概念。



这ESB类似于设计模式里面适配器模式，不管服务提供方是使用什么语音实现的，在他注册到消息总线里面，用过使用同样的一套规范，这样客户端调用就省事了很多。

所有的服务由服务总线通过管理，但这个消息总线其实只是一种思想，具体要要通过具体的RPC框架实现，说到RPC，其中最有代表意义的就是前面以及学习过的dubbo开发技术，他可以使用dubbo协议，或者RMI协议，在传输层是用netty。

除开耳熟能详的dubbo,成长比较快的是Rest协议，这其实就是因为JSON的广泛应用，如果说是XML技术成就了webservice，那么JSON就造就了Rest服务。

不过Rest还是一个未成型的标准，SpringCloud就真正讲Rest作为了RPC的实现技术，SpringCloud这技术以及成为了这行业的趋势，而且 SpringCloud 也依照于 SpringBoot

开发技术，可以实现项目的打包发布以及单独运行。

# Rest项目演练

## 项目搭建

既然是微服务，整个项目必然会参加成一个个微模块，

microcloud-api 模块，作为公共的信息导入配置模块；

microcloud-provider-product：作为服务提供者;

microcloud-consumer：作为微服务调用的客户端使用;

microcloud

新建立一个maven项目：microcloud

其中pom文件如下

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>enjoy</groupId>

<artifactId>springcloud</artifactId>

<packaging>pom</packaging>

<version>1.0-SNAPSHOT</version>

<modules>

<module>microcloudapi</module>

<module>microcloudproviderproduct</module>

<module>microcloudconsumer</module>

</modules>

<properties>

<jdk.version>1.8</jdk.version>

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

</properties>

<dependencyManagement>

<dependencies>

<dependency> <!-- 进行SpringCloud依赖包的导入处理 -->

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

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

<version>Finchley.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency> <!-- SpringCloud离不开SpringBoot，所以必须要配置此依赖包 -->

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

<artifactId>spring-boot-dependencies</artifactId>

<version>2.1.2.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.31</version>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>enjoy</groupId>

<artifactId>microcloud-api</artifactId>

<version>1.0.0</version>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<finalName>microcloud</finalName>

<plugins>

<plugin>

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

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

<configuration>

<source>${jdk.version}</source><!-- 源代码使用的开发版本 -->

<target>${jdk.version}</target><!-- 需要生成的目标class文件的编译版本 -->

</configuration>

</plugin>

</plugins>

</build>

</project>

注意：spingcloud中针对依赖包的版本并不像传统项目一样使用的是数字形式定义，反而是使用了一系列英国的地铁或者城市名字来定义，springcloud使用了springboot，其中对于的版本如下

Release Train Boot Version

Greenwich 2.1.x

Finchley 2.0.x

Edgware 1.5.x

Dalston 1.5.x

【microcloud-api】模块，建立一个公共模板，这模块的主要功能是提供公共处理的工具类，实体，接口等。

pom文件如下

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springcloud</artifactId>

<groupId>enjoy</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>microcloud-api</artifactId>

<version>1.0.0</version>

<properties>

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

</properties>

</project>

由于实体对象不管是服务提供放还是消费者都需要用到，实体对象先创建到api模块中，创建一个Product实体

package cn.enjoy.vo;

import java.io.Serializable;

public class Product implements Serializable {

private Long productId;

private String productName;

private String productDesc;

public String getProductDesc() {

return productDesc;

}

public void setProductDesc(String productDesc) {

this.productDesc = productDesc;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public Long getProductId() {

return productId;

}

public void setProductId(Long productId) {

this.productId = productId;

}

@Override

public String toString() {

return "Product{" +

"productId=" + productId +

", productName='" + productName + '\'' +

", productDesc='" + productDesc + '\'' +

'}';

}

}

## 服务提供方

【microcloud-provider-product】创建一个Product Rest提供者的项目模块,这个模块对应的数据库脚本如下

CREATE DATABASE springcloud CHARACTER SET UTF8 ;

USE springcloud ;

CREATE TABLE product (

prodcutId BIGINT AUTO\_INCREMENT ,

productName VARCHAR(50) ,

productDesc VARCHAR(50) ,

CONSTRAINT pk\_prodcut\_id PRIMARY KEY(prodcutId)

) ;

INSERT INTO product(productName,productDesc) VALUES ('java编程',database()) ;

INSERT INTO product(productName,productDesc) VALUES ('Springboot',database()) ;

INSERT INTO product(productName,productDesc) VALUES ('西游记',database()) ;

INSERT INTO product(productName,productDesc) VALUES ('水浒传',database()) ;

INSERT INTO product(productName,productDesc) VALUES ('西厢记',database()) ;

【microcloud-provider-product】模块继续使用mybaits对数据库进行操作，pom文件如下

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springcloud</artifactId>

<groupId>enjoy</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<version>1.0.0</version>

<artifactId>microcloud-provider-product</artifactId>

<dependencies>

<dependency>

<groupId>enjoy</groupId>

<artifactId>microcloud-api</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

</dependencies>

</project>

【microcloud-provider-product】创建一个ProductMapper对数据库的操作接口，这个接口方法特别简单

package cn.enjoy.mapper;

import cn.enjoy.vo.Product;

import java.util.List;

public interface ProductMapper {

boolean create(Product product);

public Product findById(Long id);

public List<Product> findAll();

}

【microcloud-provider-product】新增修改application.yml文件，追加对mybatis以及数据库的支持

server:

port: 8080

mybatis:

mapper-locations: # 所有的mapper映射文件

- classpath:mapping/\*.xml

spring:

datasource:

type: com.alibaba.druid.pool.DruidDataSource # 配置当前要使用的数据源的操作类型

driver-class-name: com.mysql.cj.jdbc.Driver # 配置MySQL的驱动程序类

url: jdbc:mysql://localhost:3306/springcloud?serverTimezone=GMT%2B8 # 数据库连接地址

username: root # 数据库用户名

password: root1234% # 数据库连接密码

logging:

level:

cn.enjoy.mapper: debug

【microcloud-provider-product】创建修改 src/main/resources/mapping/ProductMapper.xml文件

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="cn.enjoy.mapper.ProductMapper">

<select id="findById" resultType="cn.enjoy.vo.Product" parameterType="long">

select productId,productName,productDesc from product WHERE productId=#{id} ;

</select>

<select id="findAll" resultType="cn.enjoy.vo.Product">

SELECT productId,productName,productDesc from product;

</select>

<insert id="create" parameterType="cn.enjoy.vo.Product">

INSERT INTO product(productName,productDesc) VALUES (#{productName},database()) ;

</insert>

</mapper>

【microcloud-provider-product】建立IProductService接口，并创建相关实现类

package cn.enjoy.service;

import cn.enjoy.vo.Product;

import java.util.List;

public interface IProductService {

Product get(long id);

boolean add(Product product);

List<Product> list();

}

package cn.enjoy.service.impl;

import cn.enjoy.mapper.ProductMapper;

import cn.enjoy.service.IProductService;

import cn.enjoy.vo.Product;

import org.springframework.stereotype.Service;

import javax.annotation.Resource;

import java.util.List;

@Service

public class ProductServiceImpl implements IProductService {

@Resource

private ProductMapper productMapper;

@Override

public Product get(long id) {

return productMapper.findById(id);

}

@Override

public boolean add(Product product) {

return productMapper.create(product);

}

@Override

public List<Product> list() {

return productMapper.findAll();

}

}

【microcloud-provider-product】 定义主程序类，并定义好mapper扫描包

package cn.enjoy;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

@MapperScan("cn.enjoy.mapper")

public class ProductApp{

public static void main(String[] args) {

SpringApplication.run(ProductApp.class,args);

}

}

【microcloud-provider-product】编写单元测试

package cn.enjoy;

import cn.enjoy.service.IProductService;

import cn.enjoy.vo.Product;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;

@SpringBootTest(classes = ProductApp.class)

@RunWith(SpringRunner.class)

public class ProductServiceTest {

@Resource

private IProductService iProductService;

@Test

public void testGet() {

System.out.println(iProductService.get(1));

}

@Test

public void testAdd() {

Product dept = new Product() ;

dept.setProductName("lison-" + System.currentTimeMillis());

System.out.println(iProductService.add(dept));

}

@Test

public void testList() {

System.out.println(iProductService.list());

}

}

测试完成，确认所有服务的方法都正确

【microcloud-provider-product】建立ProductController建立一个Rest服务类

package cn.enjoy.controller;

import cn.enjoy.service.IProductService;

import cn.enjoy.vo.Product;

import org.springframework.web.bind.annotation.\*;

import javax.annotation.Resource;

@RestController

@RequestMapping("/prodcut")

public class ProductController {

@Resource

private IProductService iProductService;

@RequestMapping(value="/get/{id}")

public Object get(@PathVariable("id") long id) {

return this.iProductService.get(id) ;

}

@RequestMapping(value="/add")

public Object add(@RequestBody Product product) {

return this.iProductService.add(product) ;

}

@RequestMapping(value="/list")

public Object list() {

return this.iProductService.list() ;

}

}

浏览器访问：

调用get请求：[localhost:8080/prodcut/get/1](http://localhost:8080/prodcut/get/1)

调用list请求：localhost:8080/prodcut/list

## 服务消费方

创建一个maven新模块：【microcloud-consumer】

这个模块作为服务的消费方，调用前面的product服务

【microcloud-consumer】修改pom文件，pom文件内容如下

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springcloud</artifactId>

<groupId>enjoy</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>microcloud-consumer</artifactId>

<dependencies>

<dependency>

<groupId>enjoy</groupId>

<artifactId>microcloud-api</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

</dependencies>

</project>

【microcloud-consumer】修改application.yml配置文件

server:

port: 80

【microcloud-consumer】创建Rest配置类，在这需要调用Rest服务，一般需要用到RestTemplate类对象

package cn.enjoy.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.client.RestTemplate;

@Configuration

public class RestConfig {

@Bean

public RestTemplate restTemplate() {

return new RestTemplate();

}

}

【microcloud-consumer】新建一个controller，负责使用RestTemplate调用远程的product服务

package cn.enjoy.controller;

import cn.enjoy.vo.Product;

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

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

import org.springframework.web.client.RestTemplate;

import javax.annotation.Resource;

import java.util.List;

@RestController

@RequestMapping("/consumer")

public class ConsumerProductController {

public static final String PRODUCT\_GET\_URL = "http://localhost:8080/prodcut/get/";

public static final String PRODUCT\_LIST\_URL="http://localhost:8080/prodcut/list/";

public static final String PRODUCT\_ADD\_URL = "http://localhost:8080/prodcut/add/";

@Resource

private RestTemplate restTemplate;

@RequestMapping("/product/get")

public Object getProduct(long id) {

Product product = restTemplate.getForObject(PRODUCT\_GET\_URL + id, Product.class);

return product;

}

@RequestMapping("/product/list")

public Object listProduct() {

List<Product> list = restTemplate.getForObject(PRODUCT\_LIST\_URL, List.class);

return list;

}

@RequestMapping("/product/add")

public Object addPorduct(Product product) {

Boolean result = restTemplate.postForObject(PRODUCT\_ADD\_URL, product, Boolean.class);

return result;

}

}

【microcloud-consumer】编写启动类

package cn.enjoy;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class ConsumerApp {

public static void main(String[] args) {

SpringApplication.run(ConsumerApp.class,args);

}

}

调用测试

新增：<http://localhost/consumer/product/add?productName=lison>

列表查询：<http://localhost/consumer/product/list>

获得单个数据：<http://localhost/consumer/product/get?id=1>

# SpringSecurity

## 服务提供方配置安全验证

前面使用了RestTemplate进行远程接口调用，但要注意，这些Rest服务最终都可能暴露在公网的，任何人都可能调用，如果你的Rest服务属于一些私密信息，这样会导致信息的泄露。

如果想进行安全方面的处理，首先要在服务的提供方上进行处理。

【microcloud-provider-product】修改pom文件，追加 SpringSecurity 相关依赖信息

<dependency>

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

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

</dependency>

【microcloud-provider-product】修改application.yml配置文件，进行安全的用户名配置

spring:

security:

user:

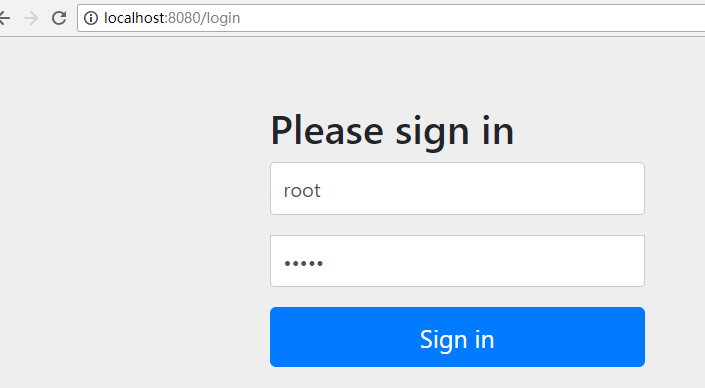
name: admin # 认证用户名

password: enjoy # 认证密码

roles:

- USER # 授权角色

在项目中访问rest接口，localhost:8080/prodcut/list，这个时候会要求先输入用户名以及密码才能允许访问



输入用户名密码就可以调用前面的接口了

## 服务消费方处理

服务提供方目前已经使用了密码验证，这个时候服务的消费方如果想直接访问就不可能了，这个时候一个以头的信息进行处理，然后使用Base64进行加密处理后才能得到正确的访问路径

【microcloud-consumer】 修改RestConfig配置类，在里面添加 HttpHeaders 的配置信息

package cn.enjoy.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpHeaders;

import org.springframework.web.client.RestTemplate;

import java.nio.charset.Charset;

import java.util.Base64;

@Configuration

public class RestConfig {

@Bean

public RestTemplate restTemplate() {

return new RestTemplate();

}

@Bean

public HttpHeaders getHeaders() { // 要进行一个Http头信息配置

HttpHeaders headers = new HttpHeaders(); // 定义一个HTTP的头信息

String auth = "admin:enjoy"; // 认证的原始信息

byte[] encodedAuth = Base64.getEncoder()

.encode(auth.getBytes(Charset.forName("US-ASCII"))); // 进行一个加密的处理

String authHeader = "Basic " + new String(encodedAuth);

headers.set("Authorization", authHeader);

return headers;

}

}

【microcloud-consumer】 修改ConsumerProductController，在进行服务端调用的时候加上这个头信息

package cn.enjoy.controller;

import cn.enjoy.vo.Product;

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

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

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

import org.springframework.web.client.RestTemplate;

import javax.annotation.Resource;

import java.util.List;

@RestController

@RequestMapping("/consumer")

public class ConsumerProductController {

public static final String PRODUCT\_GET\_URL = "http://localhost:8080/prodcut/get/";

public static final String PRODUCT\_LIST\_URL="http://localhost:8080/prodcut/list/";

public static final String PRODUCT\_ADD\_URL = "http://localhost:8080/prodcut/add/";

@Resource

private RestTemplate restTemplate;

@Resource

private HttpHeaders httpHeaders;

@RequestMapping("/product/get")

public Object getProduct(long id) {

Product product = restTemplate.exchange(PRODUCT\_GET\_URL + id,HttpMethod.GET,new HttpEntity<Object>(httpHeaders), Product.class).getBody();

return product;

}

@RequestMapping("/product/list")

public Object listProduct() {

List<Product> list = restTemplate.exchange(PRODUCT\_LIST\_URL,HttpMethod.GET,new HttpEntity<Object>(httpHeaders), List.class).getBody();

return list;

}

@RequestMapping("/product/add")

public Object addPorduct(Product product) {

Boolean result = restTemplate.exchange(PRODUCT\_ADD\_URL, HttpMethod.POST,new HttpEntity<Object>(product,httpHeaders), Boolean.class).getBody();

return result;

}

}

调用测试

新增：<http://localhost/consumer/product/add?productName=lison>

列表查询：<http://localhost/consumer/product/list>

获得单个数据：<http://localhost/consumer/product/get?id=1>

## 【microcloud-security】模块

现在服务提供方只有一个Product服务，但真实的项目开发中必然有多个服务提供方，绝大多数情况下，这些服务都会用到安全验证，而且密码也会一样，如果每个服务都单独维护，每次密码变动改动都会很大，所以应该单独建立一个安全验证的模块

创建一个microcloud-security模块，修改其pom文件如下

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springcloud</artifactId>

<groupId>enjoy</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<version>1.0.0</version>

<artifactId>microcloud-security</artifactId>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

</dependencies>

</project>

【springcloud】修改父工程pom文件，把相应的版本依赖加到里面

<dependency>

<groupId>enjoy</groupId>

<artifactId>microcloud-api</artifactId>

<version>1.0.0</version>

</dependency>

<dependency>

<groupId>enjoy</groupId>

<artifactId>microcloud-security</artifactId>

<version>1.0.0</version>

</dependency>

【microcloud-security】建立一个统一的安全配置类，这个类负责用户以及密码相关的配置

package cn.enjoy.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration

@EnableWebSecurity

public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

public void configure(AuthenticationManagerBuilder auth)

throws Exception {

auth.inMemoryAuthentication().passwordEncoder(new BCryptPasswordEncoder()).withUser("root").password(new BCryptPasswordEncoder().encode("enjoy")).roles("USER").

and().withUser("admin").password(new BCryptPasswordEncoder().encode("enjoy")).roles("USER", "ADMIN");

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.httpBasic().and().authorizeRequests().anyRequest()

.fullyAuthenticated();

http.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

}

}

【microcloud-provider-product】修改pom文件，删除spring-boot-starter-security的依赖信息，并加入自己定义的microcloud-security依赖

<!--<dependency>-->

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

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

<!--</dependency>-->

<dependency>

<groupId>enjoy</groupId>

<artifactId>microcloud-security</artifactId>

</dependency>

【microcloud-provider-product】修改application.yml，删除与安全相关的配置项。

~~# security:~~

~~# user:~~

~~# roles:~~

~~# - USER # 授权角色~~

~~# name: root~~

~~# password: enjoy~~

调用测试

新增：<http://localhost/consumer/product/add?productName=lison>

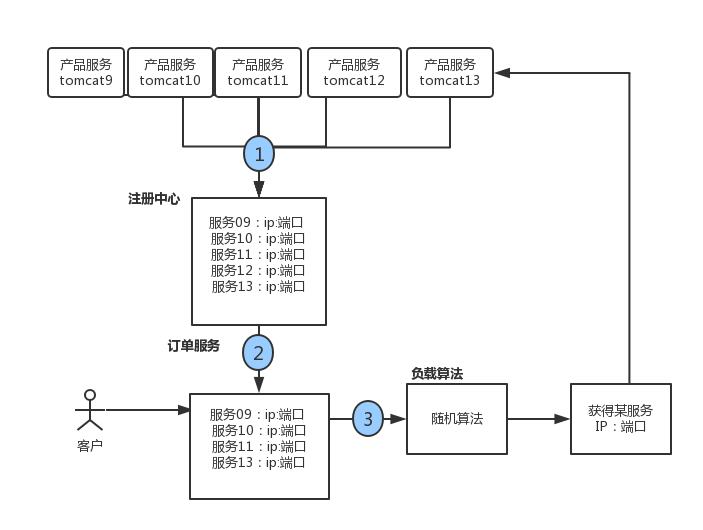
列表查询：<http://localhost/consumer/product/list>

获得单个数据：<http://localhost/consumer/product/get?id=1>

# Eureka服务注册与发现

在学习zookeeper的时候，重点就讲了服务注册与发现的流程，dubbo就是基于zookeeper来实现服务注册与发现的。

基于zookeeper的服务注册与发现大致流程如下



而在SpringCloud中，大量使用了Netflix的开源项目，其中Eureka就属于Netflix 提供的发现服务组件，所有的微服务都注册到Eureka中，它在其中扮演的就是注册中心的角色，后面所有的客户端直接从注册中心获取所需要的服务

## Eureka 服务端

新建一个microcloud-eureka模块，这模块做的事情非常简单，既启动Eureka的服务端，pom文件如下

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springcloud</artifactId>

<groupId>enjoy</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>microcloud-eureka</artifactId>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

</project>

注意：

如果是Edgware或之前的版本，用的是springboot 1.5.或者更低的版本，

<dependency>

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

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

</dependency>

【microcloud-eureka】修改application.yml文件，在里面配置eureka相关信息

server:

port: 7001

eureka:

instance: # eureak实例定义

hostname: localhost # 定义 Eureka 实例所在的主机名称

【microcloud-eureka】新增Eureka启动类，增加Eureka服务端注解

package cn.enjoy;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class EurekaApp {

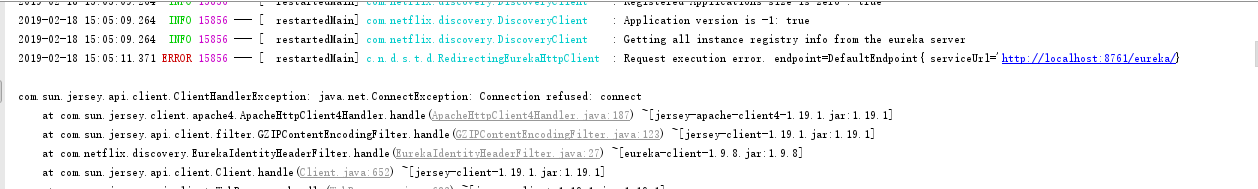
public static void main(String[] args) {

SpringApplication.run(EurekaApp.class,args);

}

}

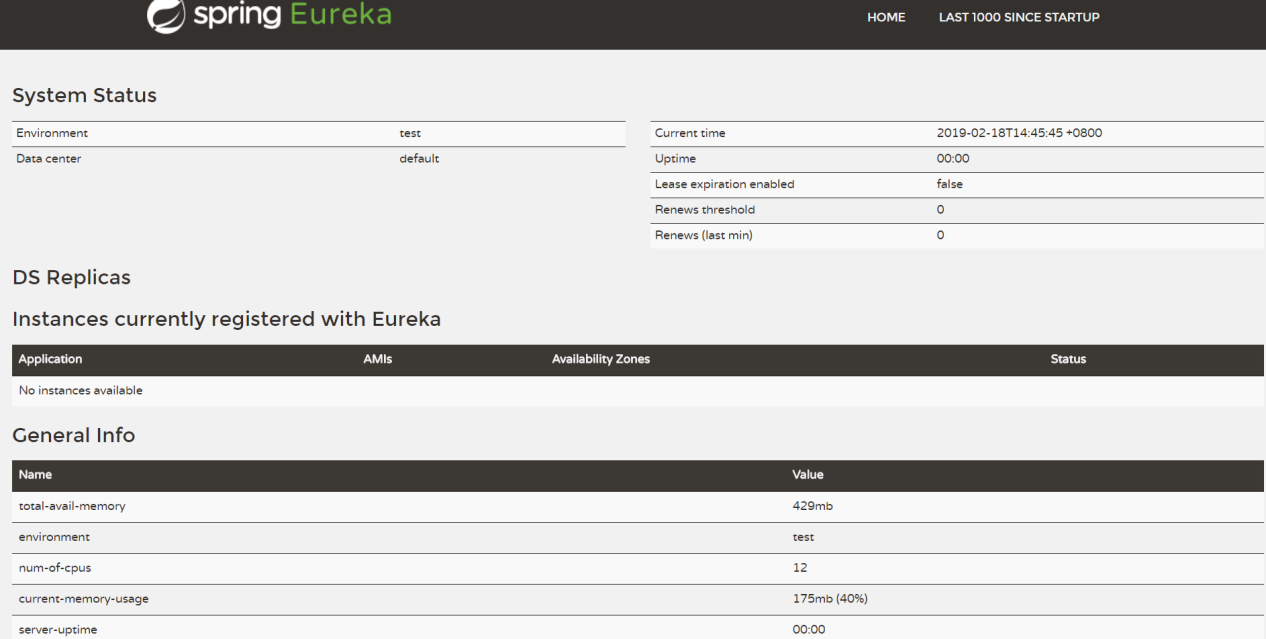
运行main方法



上面会有一些出错信息，先不用管，后面再来处理！

在浏览器上执行

<http://localhost:7001/>



## 服务提供方注册到Eureka

现在Eureka虽然有点小瑕疵，但现在已经能正常访问了，那么接下来就需要将用的微服务注册到Eureka服务当中，为后面客户端的使用做铺垫。

【microcloud-provider-product】修改pom文件，增加eureka客户端相关信息。

<dependency>

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

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

</dependency>

注意：如果是Edgware或之前的版本，用的是springboot 1.5.或者更低的版本

<dependency>

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

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

</dependency>

【microcloud-provider-product】修改application.yml配置文件，在者个文件中定义要注册的eureka服务的地址

eureka:

client: # 客户端进行Eureka注册的配置

service-url:

defaultZone: http://localhost:7001/eureka

【microcloud-provider-product】修改启动类，在这个类上增加eureka客户端的注解信息

package cn.enjoy;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@MapperScan("cn.enjoy.mapper")

@EnableEurekaClient

public class ProductApp{

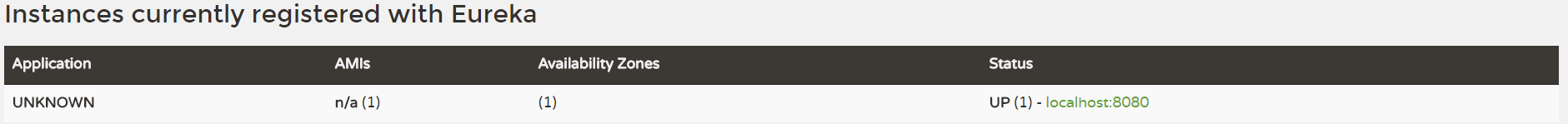
public static void main(String[] args) {

SpringApplication.run(ProductApp.class,args);

}

}

加上这注解后，启动



发现Application的名字是UNKNOWN，为此应该为这单独取一个名字

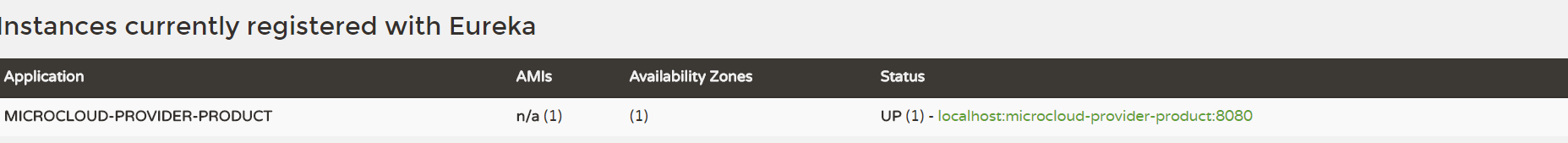
【microcloud-provider-product】修改application.yml配置文件，为这个微服务起一个名字

spring:

application:

name: microcloud-provider-product

重新启动后



现在虽然成功的实现了微服务注册，但是现在看下STATUS，这个时候名称还有点乱，我们现在开源自定义一个路径名称

【microcloud-provider-product】修改application.yml配置文件，追加主机名称的显示：

eureka:

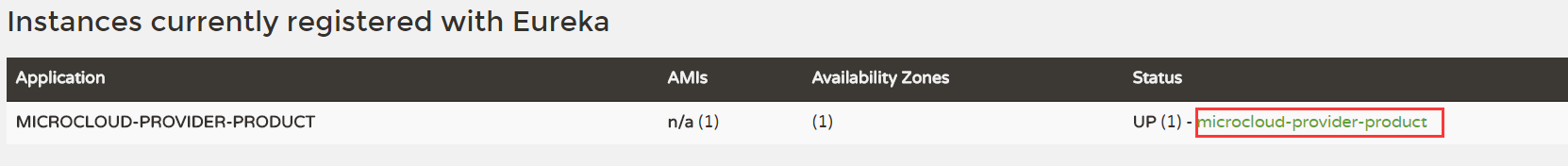
client: # 客户端进行Eureka注册的配置

service-url:

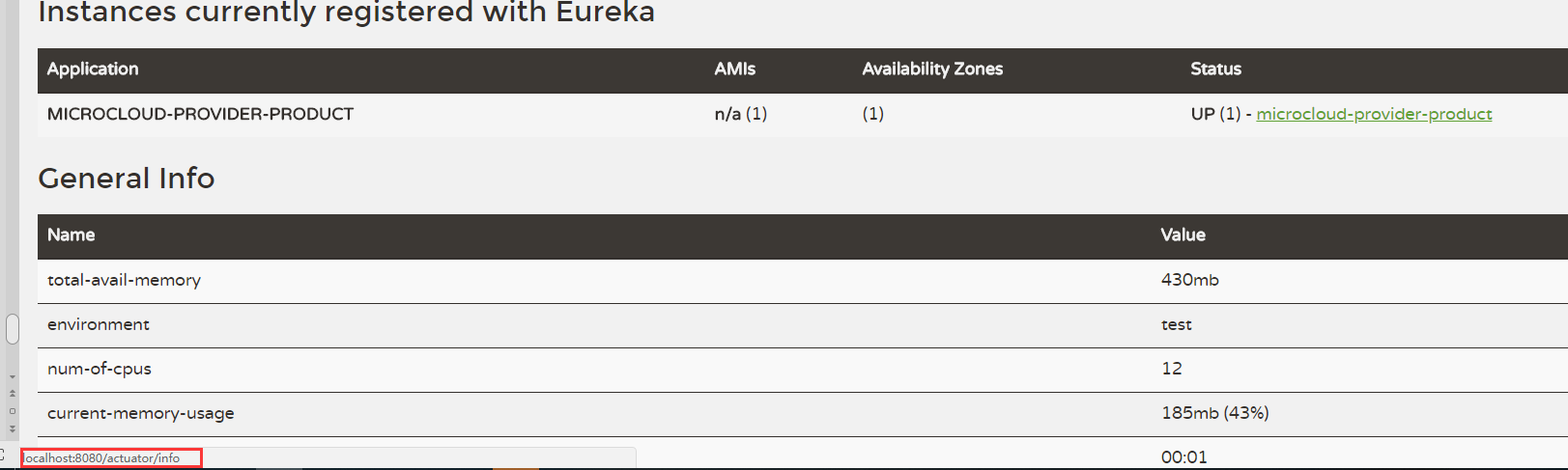
defaultZone: http://localhost:7001/eureka

instance:

instance-id: microcloud-provider-product



另外一般情况下，当鼠标点击查看的时候应该以IP作为链接项



【microcloud-provider-product】修改application.yml配置文件

eureka:

client: # 客户端进行Eureka注册的配置

service-url:

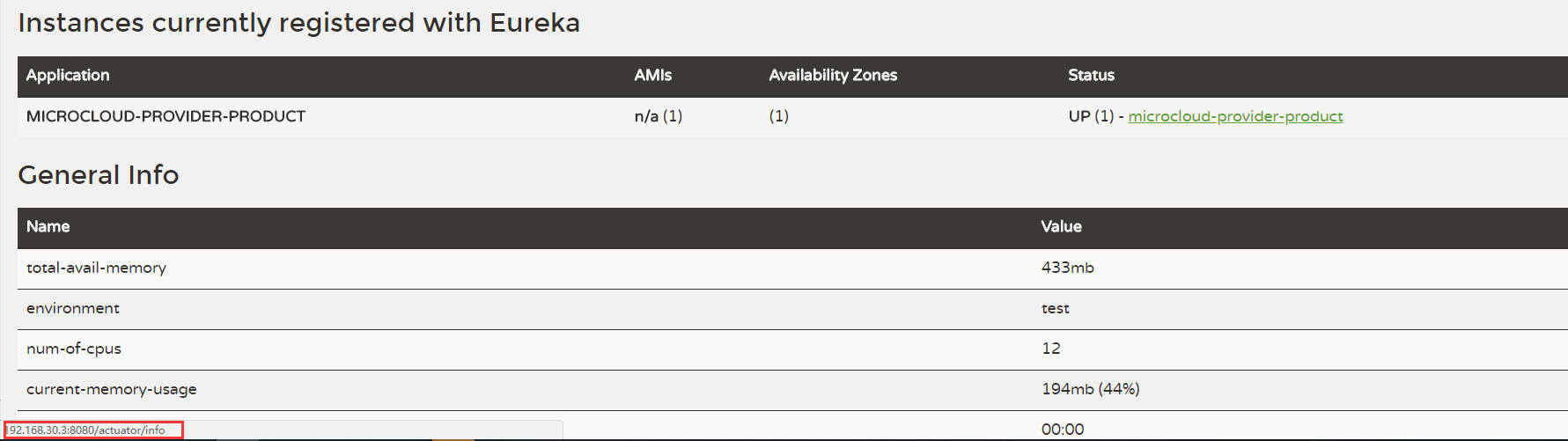
defaultZone: http://localhost:7001/eureka

instance:

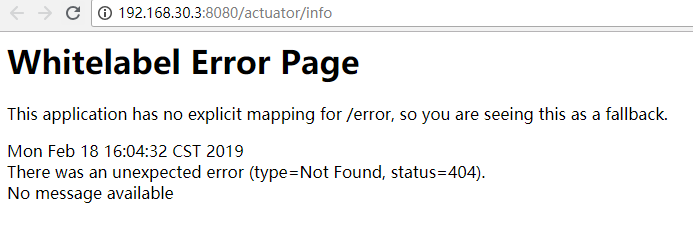
instance-id: microcloud-provider-product

prefer-ip-address: true

修改后地址栏变为



点击状态栏



【microcloud-provider-product】如果想看状态信息需要增加actuator模块，这一块的内容已经在讲springboot的时候讲过，修改pom文件，增加

<dependency>

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

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

</dependency>

【microcloud-provider-product】修改application.yml文件，追加info相关配置

info:

app.name: microcloud-provider-product

company.name: enjoy

build.artifactId: $project.artifactId$

build.modelVersion: $project.modelVersion$

注意：由于在yml文件中使用了$，这个时候启动是会报错的，因此还需要一个maven-resources-plugin插件的支持

【microcloud】在父工程增加插件，修改pom文件

<build>

<finalName>microcloud</finalName>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

<plugins>

<plugin>

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

<artifactId>maven-resources-plugin</artifactId>

<configuration>

<delimiters>

<delimiter>$</delimiter>

</delimiters>

</configuration>

</plugin>

<plugin>

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

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

<configuration>

<source>${jdk.version}</source><!-- 源代码使用的开发版本 -->

<target>${jdk.version}</target><!-- 需要生成的目标class文件的编译版本 -->

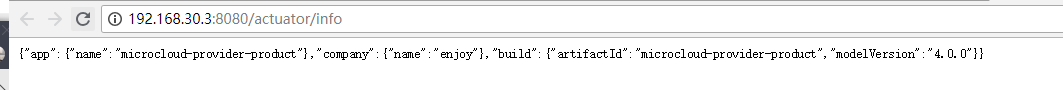
</configuration>

</plugin>

</plugins>

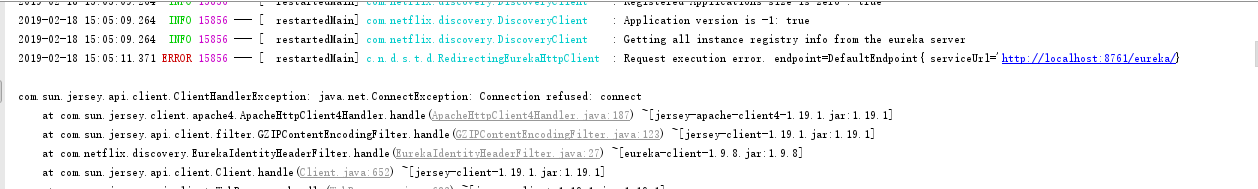
</build>

启动后：

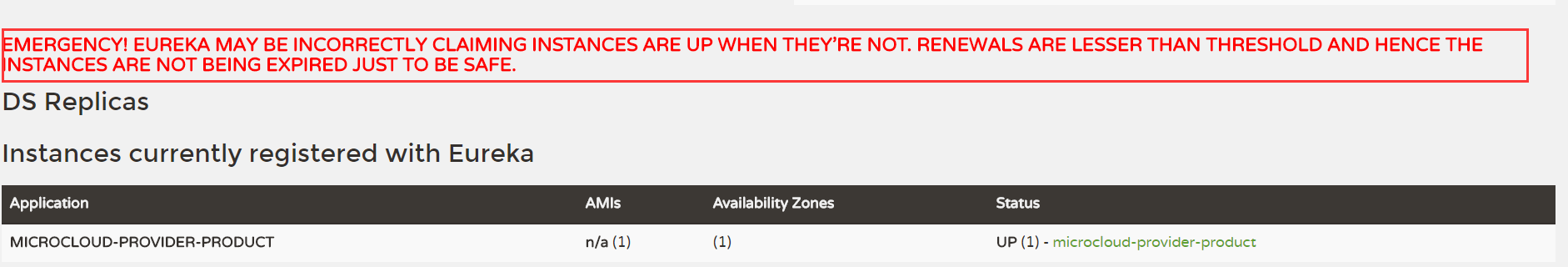


## 其他配置

在前面启动eureka中，会发现启动会报错，虽然这些错误不影响使用



另外在关闭【microcloud-provider-product】项目后，刷新eureka发现项目还在，隔一段时间后会发现



这其实就是触发了安全模式

【microcloud-eureka】设置服务的清理间隔时间，修改application.yml文件

server:

port: 7001

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

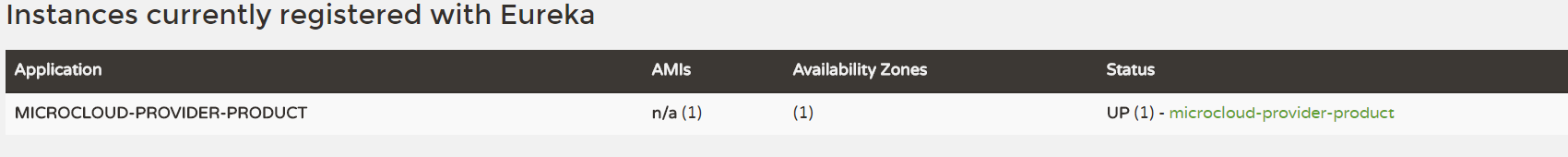
fetch-registry: false

register-with-eureka: false

instance: # eureak实例定义

hostname: localhost # 定义 Eureka 实例所在的主机名称

这个重新测试，服务提供方注册后，关闭服务发现服务实例依然还在。



【microcloud-provider-product】修改application.yml配置

eureka:

client: # 客户端进行Eureka注册的配置

service-url:

defaultZone: http://localhost:7001/eureka

instance:

instance-id: microcloud-provider-product

prefer-ip-address: true

lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔（默认是30秒）

lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔（默认是90秒）

由于所有的服务都注册到了 Eureka 之中

这样如果配置了“lease-expiration-duration-in-seconds”此选项，

表示距离上一次发送心跳之后等待下一次发送心跳的间隔时间，如果超过了此间隔时间，则认为该微服务已经宕机了。

【microcloud-provider-product】对于注册到 Eureka 上的服务，可以通过发现服务来获取一些服务信息，修改ProductController，增加一个方法

package cn.enjoy.controller;

import cn.enjoy.service.IProductService;

import cn.enjoy.vo.Product;

import org.springframework.cloud.client.discovery.DiscoveryClient;

import org.springframework.web.bind.annotation.\*;

import javax.annotation.Resource;

@RestController

@RequestMapping("/prodcut")

public class ProductController {

@Resource

private IProductService iProductService;

@Resource

private DiscoveryClient client ; // 进行Eureka的发现服务

@RequestMapping(value="/get/{id}")

public Object get(@PathVariable("id") long id) {

return this.iProductService.get(id) ;

}

@RequestMapping(value="/add")

public Object add(@RequestBody Product product) {

return this.iProductService.add(product) ;

}

@RequestMapping(value="/list")

public Object list() {

return this.iProductService.list() ;

}

@RequestMapping("/discover")

public Object discover() { // 直接返回发现服务信息

return this.client ;

}

}

【microcloud-provider-product】修改ProductApp， 在主程序中启用发现服务项

package cn.enjoy;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

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

@SpringBootApplication

@MapperScan("cn.enjoy.mapper")

@EnableEurekaClient

@EnableDiscoveryClient

public class ProductApp{

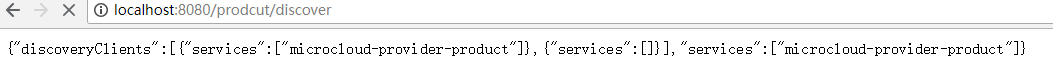
public static void main(String[] args) {

SpringApplication.run(ProductApp.class,args);

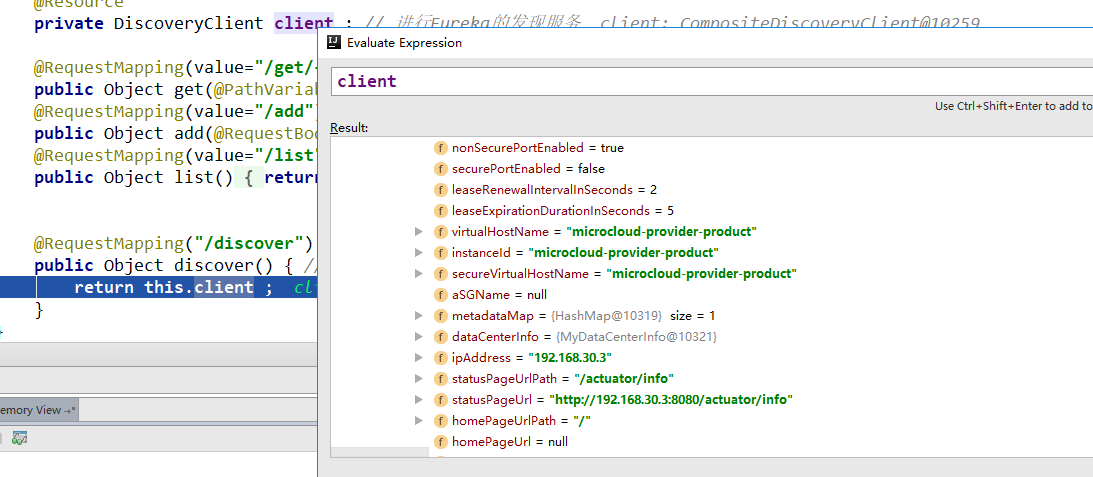
}

}

访问：localhost:8080/prodcut/discover



可以发现DiscoveryClient包含的很多信息



可以看到在eureka里面显示的信息都可以在这里获取得到。

## Eureka 安全机制

一般情况下Eureka 和服务的提供注册者都会在一个内网环境中，但免不了在某些项目中需要让其他外网的服务注册到Eureka，这个时候就有必要让Eureka增加一套安全认证机制了，让所有服务提供者通过安全认证后才能注册进来

【microcloud-eureka】修改pom文件，引入SpringSecurity的依赖包

<dependency>

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

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

</dependency>

【microcloud-eureka】 修改application.yml文件，增加用户、密码验证

server:

port: 7001

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

fetch-registry: false

register-with-eureka: false

service-url:

defaultZone: http://admin:enjoy@localhost:7001/eureka

instance: # eureak实例定义

hostname: localhost # 定义 Eureka 实例所在的主机名称

spring:

security:

user:

name: admin

password: enjoy

【microcloud-provider-product】修改application.yml文件，增加验证信息

eureka:

client: # 客户端进行Eureka注册的配置

service-url:

defaultZone: http://admin:enjoy@localhost:7001/eureka

instance:

instance-id: microcloud-provider-product

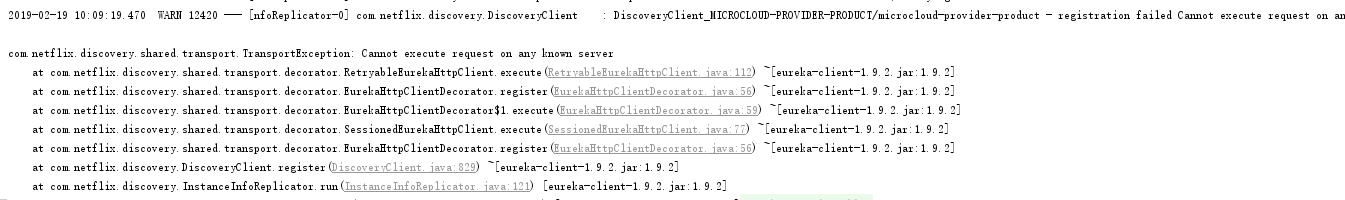
prefer-ip-address: true

lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔（默认是30秒）

lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔（默认是90秒）

重新启动

注意：如果是Edgware或之前的版本，做到这一步就行了，但使用现在版本，你会发现启动【microcloud-provider-product】后服务注册不上去



【microcloud-eureka】新增配置类EurekaSecurityConfig，重写configure方法，把csrf劫持关闭

package cn.enjoy;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableWebSecurity

public class EurekaSecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable();

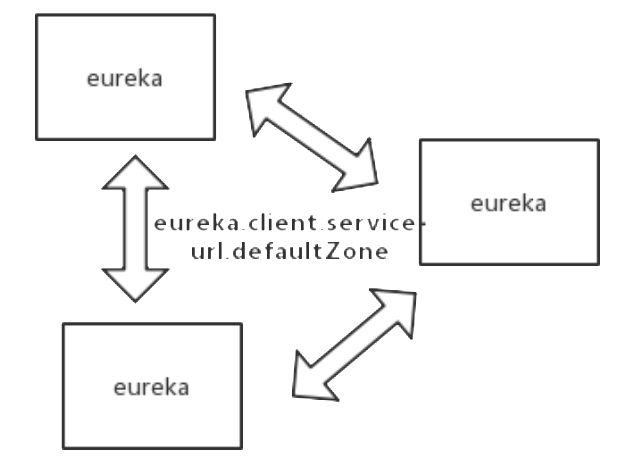
super.configure(http);

}

}

## HA 高可用

学习Zookepper服务注册的时候，注册中心是能实现高可用的，但现在的Eureka还是单节点的情况，如果Eureka出现了错误，将会导致整个集群无法继续使用，这个时候就需要考虑Eureka的高可用了。



现在需要3个eureka ，每个eureka都需要配置hostname,所有先修改hosts文件内容如下

127.0.0.1 eureka1

127.0.0.1 eureka2

127.0.0.1 eureka3

【microcloud-eureka】为了方便操作，讲microcloud-eureka项目复制两份，分别复制为【microcloud-eureka2】、 【microcloud-eureka2】

【microcloud-eureka】修改application.yml配置文件，修改端口以及注册位置

server:

port: 7001

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

fetch-registry: false

register-with-eureka: false

service-url:

#defaultZone: http://admin:enjoy@localhost:7001/eureka

defaultZone: http://admin:enjoy@eureka1:7001/eureka,http://admin:enjoy@eureka2:7002/eureka,http://admin:enjoy@eureka3:7003/eureka

instance: # eureak实例定义

hostname: eureka1 # 定义 Eureka 实例所在的主机名称

spring:

security:

user:

name: admin

password: enjoy

【microcloud-eureka2】修改application.yml配置文件

server:

port: 7002

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

fetch-registry: false

register-with-eureka: false

service-url:

#defaultZone: http://admin:enjoy@localhost:7001/eureka

defaultZone: http://admin:enjoy@eureka1:7001/eureka,http://admin:enjoy@eureka2:7002/eureka,http://admin:enjoy@eureka3:7003/eureka

instance: # eureak实例定义

hostname: eureka2 # 定义 Eureka 实例所在的主机名称

spring:

security:

user:

name: admin

password: enjoy

【microcloud-eureka3】修改application.yml配置文件

server:

port: 7003

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

fetch-registry: false

register-with-eureka: false

service-url:

#defaultZone: http://admin:enjoy@localhost:7001/eureka

defaultZone: http://admin:enjoy@eureka1:7001/eureka,http://admin:enjoy@eureka2:7002/eureka,http://admin:enjoy@eureka3:7003/eureka

instance: # eureak实例定义

hostname: eureka3 # 定义 Eureka 实例所在的主机名称

spring:

security:

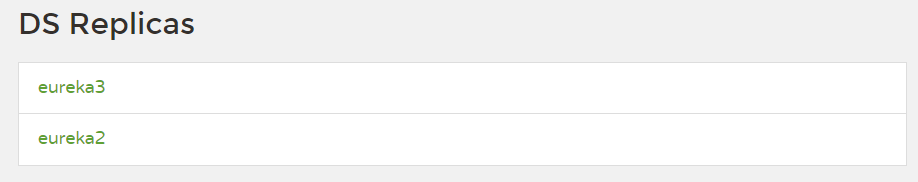
user:

name: admin

password: enjoy

启动eureka,eureka2,eureka3，进入服务的后台查看副本

登陆http://localhost:7001/



【microcloud-provider-product】修改application.yml配置文件，配置多台enreka的注册

server:

port: 8080

mybatis:

mapper-locations: # 所有的mapper映射文件

- classpath:mapping/\*.xml

spring:

datasource:

type: com.alibaba.druid.pool.DruidDataSource # 配置当前要使用的数据源的操作类型

driver-class-name: com.mysql.cj.jdbc.Driver # 配置MySQL的驱动程序类

url: jdbc:mysql://localhost:3306/springcloud?serverTimezone=GMT%2B8 # 数据库连接地址

username: root # 数据库用户名

password: root1234% # 数据库连接密码

application:

name: microcloud-provider-product

# security:

# user:

# roles:

# - USER # 授权角色

# name: root

# password: enjoy

logging:

level:

cn.enjoy.mapper: debug

eureka:

client: # 客户端进行Eureka注册的配置

service-url:

#defaultZone: http://admin:enjoy@localhost:7001/eureka

defaultZone: http://admin:enjoy@eureka1:7001/eureka,http://admin:enjoy@eureka2:7002/eureka,http://admin:enjoy@eureka3:7003/eureka

instance:

instance-id: microcloud-provider-product

prefer-ip-address: true

lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔（默认是30秒）

lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔（默认是90秒）

info:

app.name: microcloud-provider-product

company.name: enjoy

build.artifactId: $project.artifactId$

build.modelVersion: $project.modelVersion$

## 打包发布

在真实项目中，需要讲Eureka发布到具体服务器上进行执行，打包部署其实和springboot里面讲的大同小异

和properties文件稍微有点不同，对于properties文件，不同的环境会有不同的配置文件比如application-dev.properties，application-test.properties,application-pro.properties等

但如果是yml文件，所有的的配置都再同一个yml文件中

【microcloud-eureka】修改application.yml文件

spring:

profiles:

active:

- dev-7001

---

server:

port: 7001

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

fetch-registry: false

register-with-eureka: false

service-url:

defaultZone: http://admin:enjoy@eureka1:7001/eureka,http://admin:enjoy@eureka2:7002/eureka,http://admin:enjoy@eureka3:7003/eureka

instance: # eureak实例定义

hostname: eureka1 # 定义 Eureka 实例所在的主机名称

spring:

profiles: dev-7001

security:

user:

name: admin

password: enjoy

application:

name: microcloud-eureka

---

server:

port: 7002

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

fetch-registry: false

register-with-eureka: false

service-url:

defaultZone: http://admin:enjoy@eureka1:7001/eureka,http://admin:enjoy@eureka2:7002/eureka,http://admin:enjoy@eureka3:7003/eureka

instance: # eureak实例定义

hostname: eureka2 # 定义 Eureka 实例所在的主机名称

spring:

profiles: dev-7002

security:

user:

name: admin

password: enjoy

application:

name: microcloud-eureka2

---

server:

port: 7003

eureka:

server:

eviction-interval-timer-in-ms: 1000 #设置清理的间隔时间，而后这个时间使用的是毫秒单位（默认是60秒）

enable-self-preservation: false #设置为false表示关闭保护模式

client:

fetch-registry: false

register-with-eureka: false

service-url:

defaultZone: http://admin:enjoy@eureka1:7001/eureka,http://admin:enjoy@eureka2:7002/eureka,http://admin:enjoy@eureka3:7003/eureka

instance: # eureak实例定义

hostname: eureka3 # 定义 Eureka 实例所在的主机名称

spring:

profiles: dev-7003

security:

user:

name: admin

password: enjoy

application:

name: microcloud-eureka3

【microcloud-eureka】添加一个打包插件，修改pom文件

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springcloud</artifactId>

<groupId>enjoy</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>microcloud-eureka</artifactId>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<finalName>eureka-server</finalName>

<plugins>

<plugin> <!-- 该插件的主要功能是进行项目的打包发布处理 -->

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

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

<configuration> <!-- 设置程序执行的主类 -->

<mainClass>cn.enjoy.EurekaApp</mainClass>

</configuration>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

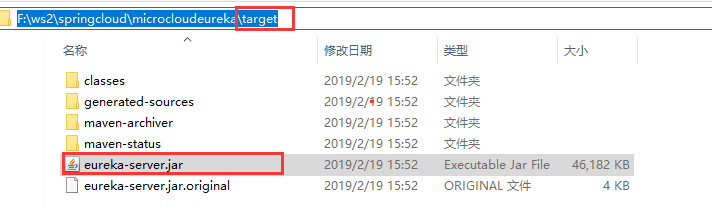
</project>

【microcloud-eureka】 在pom文件所在目录

mvn clean install package

接下来就可以在项目的编译目录发现

eureka-server.jar 文件



采用默认的方式执行 eureka-server.jar

那么此时将运行在 7001 端口上：java -jar eureka-server.jar

运行其它的两个 profile 配置：

· 运行“dev-7002”profile：java -jar eureka-server.jar --spring.profiles.active=dev-7002；

· 运行“dev-7003”profile：java -jar eureka-server.jar --spring.profiles.active=dev-7003