**中软国际考题-笔试**

1. 单选题

1、Spring Cloud 的主要优点不包括（D）

A、约定优于配置 B、轻量级组件 C、开发灵活 D、项目结构简单

2、下列对微服务的优缺点描述错误的是（D）

A．每一个服务足够内聚,代码容易理解

B．开发效率提高,一个服务只做一件事

C．微服务能够被小团队单独开发

D．多服务运维复杂度低,随着服务的增加,运维的压力也不会增大

3、Spring Cloud Netflix项目不包括以下哪个组件（B）

A. Eureka B.Zookeeper C. Zuul D. Ribbon

4、微服务的优缺点分别是什么（B）

A. 不易于与第三方集成 B. 每一个服务足够内聚,代码容易理解

C. 不可以灵活搭配,连接公共库/连接独立库 D. 微服务不是松耦合的,没有功能意义的服务

5、下列关于SpringCloud描述错误的是？（D）

A．SpringCloud是关注全局的微服务协调整理治理框架以及一整套的落地解决方案

B. SpringCloud将SpringBoot开发的一个个单体微服务整合并管理起来，为各个微服务之间提供：配置管理，服务发现，断路器，路由，微代理，事件总线等的集成服务。

C. 微服务(Microservices Architecture)是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成。

D. 系统中的各个微服务可被独立部署，各个微服务之间是紧耦合的。

6、下列关于springClound不正确的是：（D）

A：服务发现——Netflix Eureka

B：客服端负载均衡——Netflix Ribbon

C：断路器——Netflix Hystrix

D：分布式配置——Netflix Zuul

7、有关微服务的说法错误的是：C

A. 易于和第三方集成，微服务允许容易且灵活的方式集成自动部署，通过持续集成工具，如Jenkins,Hudson,bamnoo

B. 微服务易于被一个开发人员理解，修改和维护，这样小团队能够更关注自己的工作成果。无需通过合作就能体现价值

C. 微服务不能使用不同的语言开发

D. 微服务允许你利用融合最新技术

8、SpringCloud功能以下说法正确的是（B）

A. 是一个Web容器 B. 构建微服务

C.持久化技术 D. 消息队列技术

1. 多选题
2. 以下哪些是SpringCloud提供的服务治理工具？（A、B、C、D）

A.配置管理 B.服务发现 C.断路器 D.智能路由

1. 以下哪些是Ribbon的负载均衡策略（A、B、C、D）

A.简单轮询负载均衡

B.加权响应时间负载均衡

C.区域感知轮询负载均衡

D.随机负载均衡

3、Spring-Cloud构建在Spring-Boot之上的优势是哪些？（A、B、C、D）

A．创建独立的spring引用程序main方法运行

B．嵌入的tomcat无需部署war文件

C．简化maven配置

D．自动配置spring添加对应功能starter自动化配置

4、如下哪些版本的SpringCloud支持SpringBoot2.X构建？（A、C）

A. Greenwich B. Dalston C. Finchley D.Camden

5、使用Spring Cloud有什么优势？（A、B）

A. 冗余-分布式系统中的冗余问题。

B. 性能-问题 由于各种运营开销导致的性能问题

C. 一个生命周期短暂的微服务框架

D. 提供与外部系统的集成

6、下列哪些是Zuul中定义的标准过滤器类型（ABCD）

A PRE。B ROUTING。C OST。D ROR。E AFTER

7、下列关于 SpringCloud特点正确的是：（A、B.、C）

A. 约定优于配置 B. 开箱即用、快速启动 C. 组件支持丰富，功能齐全

D. 重量级的组件;

8、以下微服务的优点是（ABCD）（多选）

A. 每个服务足够内聚，足够小，代码容易理解这样能聚焦一个指定的业务功能或业务需求

B. 开发简单、开发效率提高，一个服务可能就是专一的只干一件事

C. 微服务能够被小团队单独开发，这个小团队是2-5人的开发人员组成

D. 微服务是松耦合的，是有功能意义的服务，无论是在开发阶段或部署阶段都是独立的

9下列有关微服务的说法正确的是：（ABCD）（多选）

A. 开发人员要处理分布式系统的复杂性

B. 多服务运维难度，随着服务的增加，运维的压力也在增大

C. 系统部署依赖

D. 服务间通信成本

10、和SpringCloud技术类似的技术有哪些？（A、B）

A.dubbo B. dubboX C. .SpringMVC D.Hibernate

1. 填空题（每空1分，共10分）
2. \_\_\_\_\_\_\_\_\_是一个采用的是基于接口的注解的HTTP客户端，它整合了ribbon。 (Feign)
3. 通过\_\_\_\_\_\_\_\_\_注解开启FeignCleint（@EnableFeignCleints）
4. Netflix开源了\_\_\_\_\_\_\_\_\_组件，实现了断路器模式，SpringCloud对这一组件进行了整合。（Hystrix）
5. 微服务架构是一种\_\_\_\_\_\_\_\_，它提倡将单一应用程序划分成一组\_\_\_\_\_\_\_\_，服务之间互相协调，为用户提供最终价值。(架构模式、小的服务)
6. Eureka包含了两个组件：\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_。这两个组件用于实现服务发现、简化服务的交互，进行心跳检查、客户端缓存等机制。(Eureka Server、Eureka Client)
7. Hystrix是一个\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_，旨在隔离远程系统，服务和第三方库的访问点。

(延迟、容错库)

1. Ribbon的核心组件均为接口类型,有以下几个（ ）、（ ）和（ ）（ServerList、ServerListFilter、IRule）
2. Spring Cloud Bus支持：新建、\_\_\_\_\_\_\_\_\_\_ 、RabbitMQ。(Kafka配置)
3. 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，服务之间互相协调，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通讯机制互相协作（基于\_\_\_\_\_\_\_\_\_\_ 协议的\_\_\_\_\_\_\_\_\_\_ API）。（HTTP，RESTFul）
4. 微服务每个服务都围绕着具体业务进行构建，并且能够被独立的部署到\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_\_ 等。（生产环境，类生产环境）
5. 应当尽量避免\_\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_ 服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建。（统一的，集中式的）
6. SpringCloud可以实现服务的注册与\_\_\_\_\_\_\_\_\_\_\_\_。（发现）
7. Dubbo是基于RPC远程调用的，SpringCloud是基于\_\_\_\_\_\_\_\_。（REST）
8. ZooKeeper保证的是CP,Eureka保证的是\_\_\_\_\_\_\_\_\_。（AP）
9. 配置服务启动后会自动注册进eureka服务中的注解是\_\_\_\_\_\_\_\_\_。（@EnableEurekaClient）
10. 配置服务发现的注解是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（@EnableDiscoveryClient）
11. 判断题
12. SpringCloud和Dubbo的一个差异体现在SpringCloud的默认RPC基于二进制，而Dubbo的默认RPC基于文本。（ F ）
13. Eureka是Spring Cloud默认的服务注册和发现模块，Eureka是一个高可用的组件，它没有后端缓存（因此可以在内存中完成），每一个实例注册之后需要向注册中心发送心跳。（T）
14. Java流行的微服务的框架有Dubbo和Spring-Cloud，Dubbo提供了服务的发现和断路器等功能。（ F ）
15. Eureka Server提供了REST端点，非JVM的微服务可以使用这些REST端点操作Eureka，从而实现注册与发现。（ T ）
16. 在Spring Cloud中，Ribbon默认的配置类是RibbonConfiguration，也可以使用一个POJO自定义Ribbon的配置。（ F : RibbonConfiguration->RibbonClientConfiguration）
17. Spring Cloud有两种服务调用方式：Ribbon+restTemplate和Feign。（ T ）
18. Feign是一个声明式的伪Http客户端。采用的是基于接口的注解；整合了ribbon，并和Eureka结合，默认实现了负载均衡的效果。（ T ）
19. Feign是一个伪客户端，它不做任何的请求处理。它通过处理注解生成Request，从而实现简化HTTP API开发的目的。（ T ）
20. Feign断路器的使用包括打开断路器配置、在服务接口使用@FeignClient注解指定fallback属性值，该属性值指定熔断类。（ T ）
21. zuul只能实现路由，不能进行过滤从而实现安全验证。（ F ）
22. Spring Cloud中的分布式配置中心组件spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。（ T ）
23. spring cloud config 组件中，如果配置存放在Git仓库中，可以使用JCE实现敏感配置内容加密。（ T ）
24. Zipkin Server是将数据存储在内存中，通常使用MySQL、Elasticsearch等进行后端数据存储。（ T ）
25. Eureka不再从注册列表移除因长时间没收到心跳而应该过期的服务（ T ）
26. 当网络稳定时,当前实例新的注册信息会被同步到其他节点中(最终一致性) （ T ）
27. 微服务架构的核心思想是，一个应用是由多个小的、相互独立的、微服务组成，这些服务运行在自己的进程中，开发和发布都没有依赖。（ T ）
28. 微服务架构引入策略 – 对传统企业而言，开始时可以考虑引入部分合适的微服务架构原则对已有系统进行改造或新建微服务应用，逐步探索及积累微服务架构经验，而非全盘实施微服务架构。（ T）
29. 由于Spring Cloud为服务治理做了一层抽象接口，所以在Spring Cloud应用中可以支持多种不同的服务治理框架，比如：Netflix Eureka、Consul、Zookeeper。（T）
30. Spring Cloud Zuul 没有内置 zuul proxy 代理服务器。（ F ）
31. 每个微服务都有自己的存储能力，可能有自己的数据库，也可以有统一的数据库。（ T ）
32. 微服务只是业务逻辑代码，不会和HTML、CSS或其他界面组件混合。（T）
33. 微服务允许你利用融合最新技术。（T）
34. SpringCloud和SpringBoot没有关系。（ F ）
35. eureka服务器可以做多台集群。（T）
36. Spring Cloud Ribbon是基于Netflix Ribbon实现的一套服务器端负载均衡的工具。（F）
37. Feign是一个声明式WebService客户端。（T）

五、简答题

1、什么是微服务？

答案：以前的模式是所有的代码在同一个工程中部署在同一个服务器中，同一个项目的不同模块不同功能互相抢占资源，微服务是将工程根据不同的业务规则拆分成微服务，微服务部署在不同的机器上，服务之间进行相互调用， Java微服务的框架有：dubbo（只能用来做微服务），spring cloud（提供了服务的发现，断路器等）

2、Spring cloud如何实现服务的注册和发现？

答案：服务在发布时,指定对应的服务名（服务名包括了IP地址和端口）,将服务注册到注册中心（eureka或者zookeeper）， 这一过程是spring cloud自动实现 只需要在main方法添加@EnableDisscoveryClient ，同一个服务修改端口就可以启动多个实例， 调用方法：传递服务名称通过注册中心获取所有的可用实例，通过负载均衡策略调用（ribbon和feign）对应的服务

3、ribbon和feign区别是什么？

答案：Ribbon添加maven依赖，spring-starter-ribbon，使用@RibbonClient(value="服务名称")，使用RestTemplate调用远程服务对应的方法

feign添加maven依赖，spring-starter-feign，服务提供方提供对外接口，调用方使用，在接口上使用@FeignClient("指定服务名")

Ribbon和Feign的区别：

Ribbon和Feign都是用于调用其他服务的，不过方式不同。

1.启动类使用的注解不同，Ribbon用的是@RibbonClient，Feign用的是@EnableFeignClients。

2.服务的指定位置不同，Ribbon是在@RibbonClient注解上声明，Feign则是在定义抽象方法的接口中使用@FeignClient声明。

3.调用方式不同，Ribbon需要自己构建http请求，模拟http请求然后使用RestTemplate发送给其他服务，步骤相当繁琐。

Feign则是在Ribbon的基础上进行了一次改进，采用接口的方式，将需要调用的其他服务的方法定义成抽象方法即可，

不需要自己构建http请求。不过要注意的是抽象方法的注解、方法签名要和提供服务的方法完全一致。

4、spring cloud断路器的作用是什么？

答案：在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud可以用RestTemplate+Ribbon和Feign来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。为了解决这个问题，业界提出了断路器（Hystrix）模型。较底层的服务如果出现故障，会导致连锁故障。当对特定的服务的调用的不可用达到一个阀值（Hystric 是5秒20次） 断路器将会被打开。当一个服务提供者关掉之后，刷新的时候服务请求成功和请求失败是成对出现的：当服务消费者去请求那个被关掉的服务提供者的时候就会请求失败，当服务消费者去请求正常的服务提供者时就能获得期望的结果。请求失败时断路器即会生效。

5、什么是Spring Cloud？（5分）

答案：Spring Cloud为开发人员提供了快速构建分布式系统的一些通用模式（例如配置管理，服务发现，断路器，智能路由，微代理，控制总线，一次性令牌，全局锁，领导选举，分布式会话，群集状态）。分布式系统的协调导致了锅炉板模式，并且使用Spring Cloud开发人员可以快速地站起来来实现这些模式的服务和应用程序。它们可以在任何分布式环境中正常工作，包括开发人员自己的笔记本电脑，裸机数据中心和受管平台，Cloud Foundry。

6、负载平衡的意义什么？

答案：在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

7、**服务注册和发现是什么意思？Spring Cloud如何实现？**。(分别写出2条即可)（5分）

答案：当我们开始一个项目时，我们通常在属性文件中进行所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降，而某些位置可能会发生变化。手动更改属性可能会产生问题。 Eureka服务注册和发现可以在这种情况下提供帮助。由于所有服务都在Eureka服务器上注册并通过调用Eureka服务器完成查找，因此无需处理服务地点的任何更改和处理。

8、SpringCloud和dubbo的区别？（5分）

答案：

Dubbo Spring Cloud

服务注册中心 Zookeeper Spring Cloud Netflix Eureka

服务调用方式 RPC REST API

服务网关 无 Spring Cloud Netflix Zuul

断路器 不完善 Spring Cloud Netflix Hystrix

分布式配置 无 Spring Cloud Config

服务跟踪 无 Spring Cloud Sleuth

消息总线 无 Spring Cloud Bus

数据流 无 Spring Cloud Stream

批量任务 无 Spring Cloud Task

9、Eureka的作用是什么？（5分）

ureka是Netflix的一个子模块，也是核心模块之一。Eureka是一个基于REST的服务，用于定位服务，做服务注册与发现

10、Ribbon的作用是什么？（5分）

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡的工具

11、Feign的作用是什么？（5分）

Feign是一个声明式WebService客户端。使用Feign能让编写Web Service客户端更加简单,

12、服务熔断的作用是什么？（5分）

熔断机制是应对雪崩效应的一种微服务链路保护机制

六、程序实现题

1、编写一个基于SpringCloud的服务注册和发现程序。

|  |
| --- |
| public class Test {  public static void main(String[] args) {  String money = "￥1001";  int a = Integer.*parseInt*(money.split("￥")[1]);  String num[] = {"零","一","二","三","四","五","六","七","八","九","十"};  String wei[] = {"","十","百","千"};  int qian,bai,shi,ge;  StringBuffer sf = new StringBuffer();  qian = a/1000;  bai = (a%1000)/100;  shi = ((a%1000)%100)/10;  ge = ((a%1000)%100)%10;  int type=0;  if(qian!=0){  sf.append(num[qian]+""+wei[3]);  type =1;  }else{  type=0;  }  if(bai!=0){  sf.append(num[bai]+""+wei[2]);  type = 1;  }else{  if((!(shi==0&&ge==0)&&type==1)){  sf.append(num[0]);  type=2;  }  }  if(shi!=0){  sf.append(num[shi]+""+wei[1]);  type = 1;  }else{  if(type==1&&ge!=0){  sf.append(num[0]);  type=2;  }  }  if(ge!=0){  sf.append(num[ge]+""+wei[0]);  }  sf.append("元整");  System.*out*.println(sf);  }  } |

2、手写实现简单的RPC服务。（10分）

|  |
| --- |
| 1. 用户对象-User  package com.nezha.learn.demo;  import java.io.Serializable;  public class User implements Serializable {  private static final long serialVersionUID = 1L;  private String name;  private Integer age;  public User(){  }  public User(String username, Integer age) {  this.name = username;  this.age = age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }  @Override  public String toString() {  return "User String is:"+name+",age:"+age;  }  }  2. 暴露的接口-IUserService  package com.nezha.learn.demo;  public interface IUserService {  User findById(Long id);  }  3. 接口的实现（服务的提供者）  接口的实现  package com.nezha.learn.demo;  public class UserServiceImpl implements IUserService {  @Override  public User findById(Long id) {  User user = new User();  user.setName("nezha");  user.setAge(123);  return user;  }  }  服务提供者  package com.nezha.learn.demo;  import java.io.ObjectInputStream;  import java.io.ObjectOutputStream;  import java.lang.reflect.Method;  import java.net.ServerSocket;  import java.net.Socket;  public class Provider {  public static void main(String[] args) {  try {  //1.Socket绑定本地端口  ServerSocket serverSocket = new ServerSocket(8888);  //2.监听端口  while (true){  Socket socket = serverSocket.accept();  //1.接收所有的参数  ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());  String apiClassName = inputStream.readUTF();  String methodName = inputStream.readUTF();  Class[] paramTypes = (Class[]) inputStream.readObject();  Object[] args4Method = (Object[]) inputStream.readObject();  Class clazz = null;  //2.服务注册，找到具体的实现类  if (apiClassName.equals(IUserService.class.getName())){  clazz = UserServiceImpl.class;  }  //3.执行UserServiceImpl的方法  Method method = clazz.getMethod(methodName,paramTypes);  Object invoke = method.invoke(clazz.newInstance(),args4Method);  //4.返回结果给客户端  ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());  outputStream.writeObject(invoke);  outputStream.flush();  //5.关闭连接  outputStream.close();  inputStream.close();  socket.close();  }  }catch (Exception e){  e.printStackTrace();  }  }  }  4. 服务的调用者  package com.nezha.learn.demo;  import java.io.ObjectInputStream;  import java.io.ObjectOutputStream;  import java.lang.reflect.InvocationHandler;  import java.lang.reflect.Method;  import java.lang.reflect.Proxy;  import java.net.Socket;  public class Consumer {  public static void main(String[] args) {  //1.获取代理类  IUserService userService = (IUserService) rpc(IUserService.class);  //2.触发InvocationHandler,进行远程代理  User user = userService.findById(123L);  System.out.println(user);  }  public static Object rpc(Class clazz){  return Proxy.newProxyInstance(  clazz.getClassLoader(),  new Class[]{clazz},  (proxy, method, args) ->  new InvocationHandler() {  //代理执行方法，上面设置了代理的类  @Override  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  //1.建立远程连接  Socket socket = new Socket("127.0.0.1",8888);  //2.要调用的类、方法、参数  String apiName = clazz.getName();  String methodName = method.getName();  //为了鉴别方法的重载，这里需要传入参数类型  Class[] paramTypes = method.getParameterTypes();  //3.传输类信息，请求远程执行结果  ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());  outputStream.writeUTF(apiName);  outputStream.writeUTF(methodName);  outputStream.writeObject(paramTypes);  outputStream.writeObject(args);  outputStream.flush();  //4.接收返回的结果  ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());  Object object = inputStream.readObject();  inputStream.close();  outputStream.close();  socket.close();  return object;  }  });  }  } |

3、编写Feign实现断路器的配置文件和关键代码。（10分）

|  |
| --- |
| 答案：   1. 配置文件打开断路器（2分）   feign.hystrix.enabled=true   1. 服务接口指定的定义（5分）   @FengnClient（vale=”myservice”,fallback=MyServiceHystric.class）  public interface MyService{  @RequestMapping(value=”/test”,method=RequestMethod.GET)  String getResponseFromClient(@RequestParam(value=”name”) String name);  }   1. 定义熔断器类（3分）   @Component  public class MyServiceHystric implement MyService{  @Override  public String getResponseFromClient(String name){  return “mess”+name;  }  } |

4、基于springcloud的单点登录服务及基于zuul的网关服务

（10分）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ① service-registry-server服务注册配置信息不再过多描述，标准用法      ② api-gateway-server网关服务配置信息   |  |  | | --- | --- | | 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  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | spring:    aop: #aop代理      proxyTargetClass: true    application:      name: api-gateway-server    server:    port: 8080    tomcat:      uri-encoding: UTF-8    #服务注册  eureka:    client:      serviceUrl:        defaultZone: http://127.0.0.1:8761/eureka/  #  server:  #    enable-self-preservation: false  #关闭eureka自我保护,生产环境不建议关闭自我保护    #认证中心index页面地址,如果直接登录认证中心则会跳转到该地址  uaa.server.index-path: /uaa/index  #认证中心跳转路径前缀  uaa.server.service.path: /uaa/\*\*  #不走认证的url集合  http.authorize.matchers: /\*\*/css/\*\*,/\*\*/styles/\*\*,/\*\*/js/\*\*,/\*\*/plugin/\*\*,/\*\*/plugins/\*\*,/\*\*/template/\*\*,/\*\*/img/\*\*,/\*\*/fonts/\*\*,/\*\*/cvr100u/\*\*,/css/\*\*,/js/\*\*,/plugin/\*\*,/template/\*\*,/img/\*\*,/fonts/\*\*,/cvr100u/\*\*    #网关信息  zuul:    routes:      uaa-server:        sensitiveHeaders: "\*"  #敏感headers也支持全局设置(必须这样设置)        path: ${uaa.server.service.path}        stripPrefix: false    add-proxy-headers: true  #X-Forwarder-Host请求头默认添加到转发请求中    #安全认证信息  security:    basic:      enabled: false    oauth2:      sso:        loginPath: /login      client:        accessTokenUri: http://127.0.0.1:7769/uaa/oauth/token        userAuthorizationUri: /uaa/oauth/authorize        clientId: acme        clientSecret: acmesecret      resource:        jwt:          keyValue: |            -----BEGIN PUBLIC KEY-----            MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnGp/Q5lh0P8nPL21oMMrt2RrkT9AW5jgYwLfSUnJVc9G6uR3cXRRDCjHqWU5WYwivcF180A6CWp/ireQFFBNowgc5XaA0kPpzEtgsA5YsNX7iSnUibB004iBTfU9hZ2Rbsc8cWqynT0RyN4TP1RYVSeVKvMQk4GT1r7JCEC+TNu1ELmbNwMQyzKjsfBXyIOCFU/E94ktvsTZUHF4Oq44DBylCDsS1k7/sfZC2G5EU7Oz0mhG8+Uz6MSEQHtoIi6mc8u64Rwi3Z3tscuWG2ShtsUFuNSAFNkY7LkLn+/hxLCu2bNISMaESa8dG22CIMuIeRLVcAmEWEWH5EEforTg+QIDAQAB            -----END PUBLIC KEY-----        id: openid        serviceId: ${PREFIX:}resource |       ③ uaa-server配置信息   |  |  | | --- | --- | | 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  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | spring:    application:      name: uaa-server    #数据库连接信息    datasource:      url: jdbc:mysql://localhost:3306/uaa?characterEncoding=UTF-8      username: root      password: root      driver-class-name: com.mysql.jdbc.Driver      max-idle: 5      max-wait: 10000      min-idle: 2      initial-size: 3      validation-query: SELECT 1      time-between-eviction-runs-millis: 18800      jdbc-interceptors: ConnectionState;SlowQueryReport(threshold=50)    jpa:      database: MYSQL      show-sql: true    #使用redis存储session,redis服务地址    redis:      host: 127.0.0.1      port: 6379  #不缓存thymeleaf模板，开发环境下配置该属性，生产环境下请勿配置  thymeleaf:      cache: false      cache-period: 0  template:      cache: false    server:    port: 7769    context-path: /uaa   #认证服务上下文地址(必须配置)    use-forward-headers: false    tomcat:      uri-encoding: UTF-8    #服务注册  eureka:    instance:      preferIpAddress: true    client:      serviceUrl:        defaultZone: http://127.0.0.1:8761/eureka/    security:    basic:      enabled: false    user:      password: password    ignored: /css/\*\*,/js/\*\*,/favicon.ico,/webjars/\*\*    sessions: NEVER #永远不自己创建session    #jwt信息(自定义的属性,AuthorizationServerConfigurer配置类中用到)  jwt:    access:      token:        converter:          resource:            location: classpath:keystore.jks            password: foobar            key-pair-alias: test    #自定义的属性,WebSecurityConfigurer配置类中用到  http:    authorize:      #不走认证的url集合      matchers: /\*\*/css/\*\*,/\*\*/js/\*\*,/\*\*/plugin/\*\*,/\*\*/template/\*\*,/\*\*/img/\*\*,/\*\*/fonts/\*\*,/\*\*/cvr100u/\*\*,/css/\*\*,/js/\*\*,/plugin/\*\*,/template/\*\*,/img/\*\*,/fonts/\*\*,/cvr100u/\*\*    login: | |

# 5、用代码、命令演示SpringCloud与Consul集成实现负载均衡功能?

|  |
| --- |
| **Consul集群环境搭建**  本次搭建采用了三台虚拟机，Linux系统，从Consul官网下载安装包，解压即可使用。在每台机器下建一个目录/data/consul/data，这个目录主要存放器群启动后生成的一些数据。需要开启的端口，8300， 8301， 8500， 8600。  分别在以下三个机器上执行一下启动命令。  机器1：10.0.0.45  ./consul agent -server -bootstrap-expect 2 -data-dir=/data/consul/data -node=consul-server-1 -bind=10.0.0.45 -client=0.0.0.0 &  机器2：10.0.0.100  ./consul agent -server -bootstrap-expect 2 -data-dir=/data/consul/data -node=consul-server-2 -bind=10.0.0.100 -client=0.0.0.0 &  机器3：10.0.0.191  ./consul agent -server -bootstrap-expect 2 -data-dir=/data/consul/data -node=consul-server-3 -bind=10.0.0.191 -client=0.0.0.0 -ui &  **参数说明：**  • server： 以server身份启动。默认是client • bootstrap-expect：集群要求的最少server数量，当低于这个数量，集群即失效。 • data-dir：data存放的目录，更多信息请参阅consul数据同步机制 • node：节点id，在同一集群不能重复。 • bind：监听的ip地址。默认绑定0.0.0.0，可以不指定 • client: 客户端的ip地址，0.0.0.0是指谁都可以访问 • ui: 可以访问UI界面  三台机器上的服务启动完成后，将两台机器添加到其中一台机器上，组建成集群。  分别在机器2和机器3上执行：./consul join 10.0.0.45，构成集群，同样也可以再增加别的节点。  启动成功后访问任意一台机器  [http://10.0.0.45:8500](http://10.0.0.45:8500/)，会看到如下界面，有3个健康的节点：  SpringCloud,Consul,集成,负载均衡  **三、SpringCloud服务注册**  注册中心已经搭建完毕，接下来就是准备两个服务，一个是生产者consul-producer，一个是消费者consul-consumer。  3.1 生产者代码，代码很简单，就是RestAPI请求并返回一个字符串  @RestController  public class ProducerController {  @GetMapping("/producer")  public String producer(){  System.out.println("I'm producer");  return "Hello, I'm producer";  }  }  以下是生产者consul-producer的启动配置文件application.yml：  spring:  application:  name: consul-producer #程序名称  cloud:  consul:  host: 10.0.0.100 #consul节点IP  port: 8500 #consul节点端口  discovery:  instance-id: ${spring.application.name}:${spring.cloud.client.ip-address}:${server.port} #实例ID  health-check-path: /actuator/health #健康检查  health-check-interval: 10s  server:  port: 9091 #消费者服务端口  3.2 消费者代码，代码也很简单，使用大家熟悉分Feign，Controller接受请求，用过service调用消费者提供的服务  /\*\*  \* 消费者  \*/  @RestController  public class ConsumerController {  @Autowired  private ConsumerService consumerService;  @GetMapping("/consumer")  public String getProducer(){  return consumerService.consumer();  }  }  以下是feign调用的接口声明：  /\*\*  \* 调用生产者服务  \*/  @FeignClient("consul-producer")  public interface ProducerRemote {  @GetMapping("/producer")  String producer();  }  以下是生产者consul-consumer的启动配置文件application.yml：  spring:  application:  name: consul-consumer #消费者名称  cloud:  consul:  host: 10.0.0.45 #consul注册节点IP  port: 8500 #consul注册节点端口  discovery:  instance-id: ${spring.application.name}:${spring.cloud.client.ip-address}:${server.port} #实例ID  health-check-path: /actuator/health #健康检查  health-check-interval: 10s  server:  port: 9092 #消费者服务端口  两部分的源码可在Github上查看<https://github.com/liuzwei/consul-demo.git>  生产者和消费者存在了，就启动可以了，接下来是启动两个生产者，一个消费者，为了直观的区分两个生产者，在调用生产者返回的string稍微做了区分，一个是返回“Hello, I'm producer”，而另一个则返回“Hello, I'm producer2”，为了实现负载均衡便于测试。  其中在本机上10.0.0.93分别启动了程序consul-consumer和consul-producer，在另一台机器10.0.0.191上启动另一个生产者实例consul-producer  启动之后在consul的界面上看到如下样子：  SpringCloud,Consul,集成,负载均衡  SpringCloud,Consul,集成,负载均衡  **四、测试负载功能**  通过浏览器，Get请求消费者的controller，会看到“Hello, I'm producer”和“Hello, I'm producer2”在交替显示，这也是fegin调用的特性，轮询机制实现负载均衡。  SpringCloud,Consul,集成,负载均衡  SpringCloud,Consul,集成,负载均衡 |

6、金额转换：阿拉伯数字的金额转换成传统数字（如：￥1001-🡪一千零一十一元整）

（10分）

|  |
| --- |
| public class Test {  public static void main(String[] args) {  String money = "￥1001";  int a = Integer.*parseInt*(money.split("￥")[1]);  String num[] = {"零","一","二","三","四","五","六","七","八","九","十"};  String wei[] = {"","十","百","千"};  int qian,bai,shi,ge;  StringBuffer sf = new StringBuffer();  qian = a/1000;  bai = (a%1000)/100;  shi = ((a%1000)%100)/10;  ge = ((a%1000)%100)%10;  int type=0;  if(qian!=0){  sf.append(num[qian]+""+wei[3]);  type =1;  }else{  type=0;  }  if(bai!=0){  sf.append(num[bai]+""+wei[2]);  type = 1;  }else{  if((!(shi==0&&ge==0)&&type==1)){  sf.append(num[0]);  type=2;  }  }  if(shi!=0){  sf.append(num[shi]+""+wei[1]);  type = 1;  }else{  if(type==1&&ge!=0){  sf.append(num[0]);  type=2;  }  }  if(ge!=0){  sf.append(num[ge]+""+wei[0]);  }  sf.append("元整");  System.*out*.println(sf);  }  } |

7、代码实现：springboot集成mybatis的过程。（10分）

|  |
| --- |
| 添加mybatis的starter maven依赖                  <dependency>                          <groupId>org.mybatis.spring.boot</groupId>                          <artifactId>mybatis-spring-boot-starter</artifactId>                          <version>1.2.0</version>                  </dependency>          在mybatis的接口中 添加@Mapper注解          在application.yml配置数据源信息 |

8、简单编写：springboot如何添加【修改代码】自动重启功能（10分）

添加开发者工具集=====spring-boot-devtools

<dependencies>

<dependency>

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

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

<optional>true</optional>

</dependency>

</dependencies>

9、请编写一个服务器提供者的配置，微服务名称为microservicecloud-dept，注册中心服务器是http://localhost:7001/eureka

（10分）

|  |
| --- |
| eureka:  client: #客户端注册进eureka服务列表内  service-url:  #defaultZone: http://localhost:7001/eureka  instance:  instance-id: microservicecloud-dept8001 #不使用默认localhost:microservicecloud-dept:8001 自定义微服务名称  prefer-ip-address: true #访问路径可以显示IP地址  spring:  application:  name: microservicecloud-dept |