# spring cloud简介

spring cloud是基于springboot的分布式微服务框架，为开发人员提供了在分布式系统操作的开发工具，包括配置管理、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

# 服务注册中心Eureka

Eureka是一个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，每一个实例注册之后需要向注册中心发送心跳（因此可用在内存中完成），默认情况下eureka server也是一个eureka client，必须要指定一个server。

## 创建服务注册中心（eureka server）

步骤：

-1）在启动类上添加@EnableEurekaServer注解，表明自己是一个eurekaServer

-2）配置application.properties

-3）测试：http://localhost:8080，在DS Replicas中未发现有服务被发现

在application.properties中配置：

--表明自己是一个eureka server

eureka.client.registerWithEureka：false

fetchRegistry：false

//启动一个服务注册中心，只需要在main类上添加@EnableEurekaServer

@SpringBootApplication

@EnableEurekaClient

public class EurekaServer {

public static void main(String[] args) {

SpringApplication.run(EurekaServer.class, args);

}

}

## 创建服务提供者（eureka client）

当client向server注册时，它会提供一些元数据，例如主机和端口，URL，主页等。服务注册中心从每个client实例接收心跳消息。如果心跳超时，则通常将该实例从注册server中删除。

步骤：

-1）在启动类上添加@EnableEurekaClient注解，表明自己是一个eurekaClient

-2）在application.yml中配置服务中心server的地址

-3）测试：启动后刷新注册中心地址，在DS Replicas中能看到一个服务（服务名为EUREKA-CLIENT1，端口号为8081），打开http://localhost:8081/hello?name=AAA，能得到相应输出

附：启动server工程；再启动client工程，它的端口号为8081；将client工程的配置文件的端口号改成8082并启动；这时候会发现：服务名为EUREKA-CLIENT1在server中注册了2个实例，这相当于一个小的集群。

在application.properties中配置：

eureka.client.serviceUrl.defaultZone=http://localhost:8080/eureka

# 服务消费者（rest+ribbon）

在微服务架构中，业务都会拆分成一个独立的服务，服务与服务的通讯是基于http restful的。

Spring Cloud有两种服务调用方式：1）ribbon+restTemplate，2）feign。

ribbon是一个负载均衡客户端，可以很好的控制http和tcp的一些行为。Feign默认集成了ribbon。

# 分布式锁

分布式锁（全局锁）实现：基于redis实现、基于Zookeeper实现、基于Consul实现

## 基于Concul的分布式锁

基于Consul的分布式锁主要利用key/value存储API中的acquire和release操作来实现。

acquire操作：当锁不存在持有者时返回true，并且设值setValue，同时执行操作的session会持有对该key的锁，否则就返回false。

release操作：使用指定session来释放某个key的锁，如果指定的session无效，则会返回false，否则就会设值setValue，并返回true。

# SpringBoot:RestTemplate

用于进行http请求测试。

例如：

@RunWith(SpringRunner.class)

@SpringBootTest(classes = LockAndTransactionApplication.class, webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

public class ControllerTest {

@Autowired

private TestRestTemplate testRestTemplate;

@Test

public void concurrentComment() {

String url = "http://localhost:9090/ getComments";

for (int i = 0; i < 100; i++) {

new Thread(() -> {

MultiValueMap<String, String> params = new LinkedMultiValueMap<>();

params.add("username", "admin");

params.add("comments", "测试内容" + i);

String result = testRestTemplate.postForObject(url, params, String.class);

}).start();

}

}

}

# Zookeeper介绍

## 概述

Zookeeper是一个开源的分布式协调服务，由雅虎创建，是Google Chubby的开源实现。分布式应用程序可以基于Zookeeper实现数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master选举、配置维护、名字服务、分布式同步、分布式锁和分布式队列等功能。

## 核心概念

1）集群角色：

在Zookeeper中有三种角色：Leader、Follower、Observer

一个Zookeeper集群同一时刻只会有一个Leader，其它都是Follower或Observer。

Zookeeper默认只有Leader和Follower两种角色，没有Observer角色。

2）节点读写服务分工：

Zookeeper集群的所有机器通过一个Leader选举过程来选定一台被称为“Leader”的机器，Leader服务器为客户端提供读和写服务。

Follower和Observer都能提供读服务，不能提供写服务。两者唯一的却别在于，Observer机器不参与Leader选举过程，也不参与写操作的“过半写成功”策略，因此Observer可以在不影响写性能的情况下提升集群的读性能。

3）Session：

Session是指客户端会话。在Zookeeper中，一个客户端连接是指客户端和Zookeeper服务器之间的TCP长连接。

ZooKeeper对外的服务端口默认是2181，客户端启动时，首先会与服务器建立一个TCP连接，从第一次连接建立开始，客户端会话的生命周期也开始了，通过这个连接，客户端能够通过心跳检测和服务器保持有效的会话，也能够向ZooKeeper服务器发送请求并接收响应，同时还能通过该连接接收来自服务器的watch事件通知。

Session的SessionTimeout值用来设值一个客户端会话的超时时间。当由于服务器压力太大、网络故障或客户端主动断开连接等各种原因导致客户端连接断开时，只要在SessionTimeout规定的事件内能重新连接上集群中任意一台服务器，那么之前创建的会话仍然有效。

4）数据节点：

ZooKeeper结构其实就是一个树形结构，Leader是根节点，其它节点相当于Follower节点。

ZooKeeper节点分两类：

-1）持久节点：

-2）临时节点：其生命周期跟客户端会话绑定，一旦客户端会话失效，那么该客户端所创建的所有临时节点都会被移除。

5）状态信息：

每个节点除了存储数据内容之外，还存储了节点本身的一些状态信息。用get命令可以同时获得某个节点的内容和状态信息。

在ZooKeeper中，version属性是用来实现乐观锁机制中的“写入校验”的。

6）事务操作：

在ZooKeeper中，能改变ZooKeeper服务器状态的操作称为事务操作。一般包括数据节点创建和删除、数据内容更新和客户端会话创建与失效等操作。对应每一个事务请求，ZooKeeper都会为其分配一个全局唯一的事务ID，用ZXID表示，通常是一个64位的数字。每一个ZXID对应一次更新操作，从这些ZXID中可以间接地识别出ZooKeeper处理这些事务操作请求的全局顺序。

7）Watcher（事件监听器）：

ZooKeeper允许用户在指定节点上注册一些 Watcher，并且在一些特定事件触发的时候，ZooKeeper服务端会将事件通知到感兴趣的客户端上去。该机制是 ZooKeeper 实现分布式协调服务的重要特性。

## 应用场景

ZooKeepe是一个高可用的分布式数据管理与协调框架。基于对ZAB算法的实现，该框架能够很好地保证分布式环境中数据的一致性。也基于这样的特性，使得ZooKeeper成为了解决分布式一致性问题的利器。

### 数据发布与订阅（配置中心）

数据发布与订阅，即所谓的配置中心，即发布者将数据发布到ZooKeeper节点上，供订阅者进行数据订阅，进而达到动态获取数据的目的，实现配置信息的集中式管理和动态更新。

发布/订阅系统一般有两种设计模式：

-1）推模式（Push）：服务端主动将数据更新发送给所有订阅的客户端。

-2）拉模式（Pull）：客户端主动发起请求来获取最新数据，通常客户端都采用定时轮询拉取的方式。

ZooKeeper采用的是推拉相结合的方式：

客户端想服务端注册自己需要关注的节点，一旦该节点的数据发生变更，那么服务端就会向相应的客户端发送Watcher事件通知，客户端接收到这个消息通知后，需要主动到服务端获取最新的数据。

### 命名服务

在分布式系统中，通过使用命名服务，客户端应用能够根据指定名字来获取资源或服务的地址，提供者等信息。被命名的实体通常可以是集群中的机器，提供的服务，远程对象等等。--这些我们都可以统称他们的名字。

ZooKeeper的命名服务即生成全局唯一的ID。

### 分布式协调服务/通知

ZooKeeper中特有的Watcher注册与异步通知机制，能够很好的实现分布式环境下不同机器，甚至不同系统之间的通知与协调，从而实现对数据变更的实时处理。使用方法通常是不同的客户端。

如果及其节点发生了变化，那么所有订阅的客户端都能够受到相应的Watcher通知，并做出相应的处理。

ZooKeeper的分布式协调/通知，是一种通用的分布式系统机器间的通信方式。

### Master选举

针对Master选举的需求，通常情况下，我们可以选择常见的关系型数据库中的主键特性来实现：希望成为Master的机器都向数据库中插入一条相同主键Id的记录，数据库回帮我们进行主键冲突检查，即只有一台机器能插入成功，该机器就是Master。

利用 ZooKeepr 的强一致性，能够很好地保证在分布式高并发情况下节点的创建一定能够保证全局唯一性，即 ZooKeeper 将会保证客户端无法创建一个已经存在的 数据单元节点。

也就是说，如果同时有多个客户端请求创建同一个临时节点，那么最终一定只有一个客户端请求能够创建成功。利用这个特性，就能很容易地在分布式环境中进行 Master 选举了。

### 分布式锁

分布式锁是控制分布式系统之间同步访问共享资源的一种方式。

分布式锁分类：

-1）排它锁

定义锁：ZooKeepr上的一个机器节点可以表示一个锁

获得锁：把ZooKeepr上的一个节点看作是一个锁，获得锁就通过创建心事节点的方式来实现。ZooKeeper 会保证在所有客户端中，最终只有一个客户端能够创建成功，那么就可以认为该客户端获得了锁。同时，所有没有获取到锁的客户端就需要到/exclusive\_lock节点上注册一个子节点变更的Watcher监听，以便实时监听到lock节点的变更情况。

释放锁：因为锁是一个临时节点，释放锁有两种方式。1.当前获得锁的客户端机器发生宕机或重启，那么该临时节点就会被删除，释放锁；2. 正常执行完业务逻辑后，客户端就会主动将自己创建的临时节点删除，释放锁。

-2）共享锁：