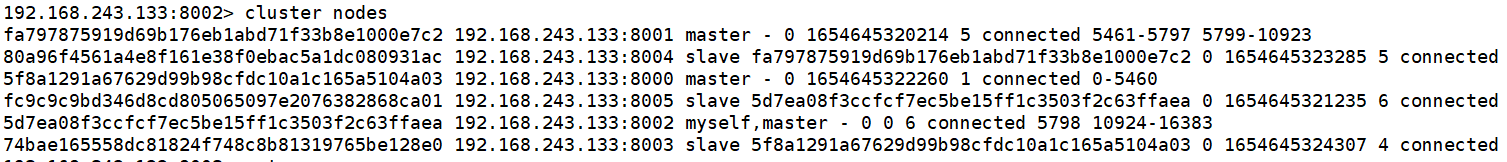
**Springboot整合JedisCluster操作集群**

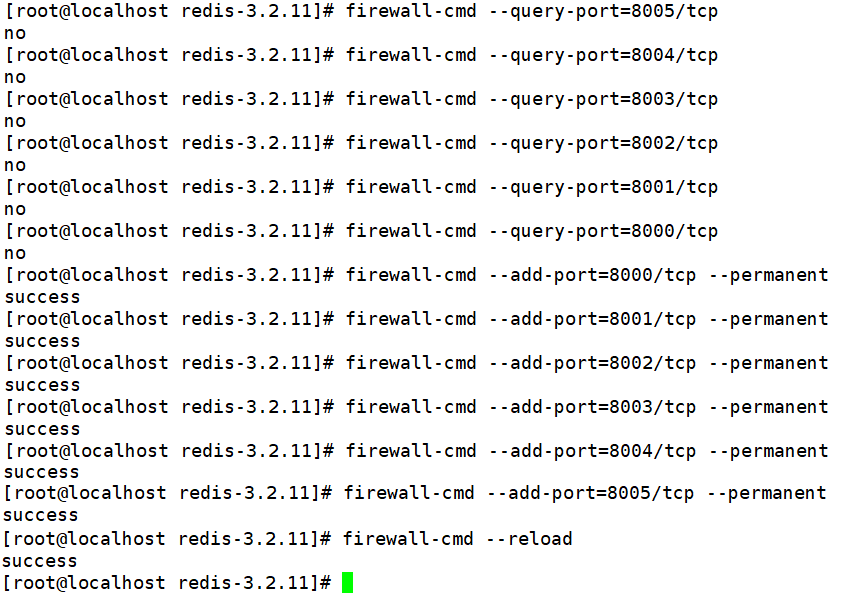
**1.环境**

## 1.1启动redis集群

启动redis集群，查看nodes,三主三从6个节点正常。



## 1.2 开启8000-8005端口号

****

**2.JedisCluster**

jedis将操作redis-cluster集群的工作封装成了一个对象JedisCluster,这个对象自带连接池.

在springboot-redis-test01项目中的cn.edu.scnu.jedis.test包中创建：JedistClusterTest类

**package** cn.edu.scnu.jedis.test;

**import** java.util.HashSet;

**import** java.util.Set;

**import** org.apache.commons.pool2.impl.GenericObjectPoolConfig;

**import** org.junit.Test;

**import** redis.clients.jedis.HostAndPort;

**import** redis.clients.jedis.JedisCluster;

**public** **class** JedistClusterTest {

@Test

**public** **void** test(){

//收集节点信息,至少提供一个节点的连接信息

Set<HostAndPort> clusterSet=**new** HashSet<HostAndPort>();

//将至少一个节点传递给set对象

clusterSet.add(**new** HostAndPort("192.168.243.133", 8005));

//底层通过连接的节点获取集群全部节点信息

//配置连接池的配置对象

GenericObjectPoolConfig config=

**new** GenericObjectPoolConfig();

config.setMaxTotal(200);

//构造一个JedisCluster

JedisCluster cluster=**new** JedisCluster(clusterSet, config);

cluster.set("gender", "male");

System.***out***.println(cluster.get("gender"));

}

}

运行后，在集群中查看：

图片包含 文本

描述已自动生成

图形用户界面, 文本

描述已自动生成

**2.整合到springboot**

## 2.1把JedisCluster交给交给spring框架维护

### 2.1.1 application.properties准备配置数据

**节点连接信息nodes**

**连接池配置信息 total idle wait等等**

右键点击springboot-redis-test01项目，点击“Build Path”—>“New Source Folder”创建“src/main/resources”文件夹，在该文件夹中创建“application.properties”文件

server.port=8095

#redis config

spring.redis.cluster.nodes=192.168.243.133:8000,192.168.243.133:8001,192.168.243.133:8002

spring.redis.cluster.maxTotal=200

spring.redis.cluster.maxIdle=8

spring.redis.cluster.minIdle=2

spring.application.name=redisservice

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

### 2.1.2创建配置类

创建配置类

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

ConfigurationProperties读取前缀的各种属性,Configuration标识配置类

**package** cn.edu.scnu.config;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**import** org.apache.commons.pool2.impl.GenericObjectPoolConfig;

**import** org.springframework.boot.context.properties.ConfigurationProperties;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** redis.clients.jedis.HostAndPort;

**import** redis.clients.jedis.JedisCluster;

@Configuration

@ConfigurationProperties(prefix="spring.redis.cluster")

**public** **class** RedisClusterConfig {

**private** List<String> nodes;

**private** Integer maxTotal;

**private** Integer maxIdle;

**private** Integer minIdle;

//初始化JedisCluster的方法

@Bean

**public** JedisCluster initJedisCluster(){

//收集节点信息

Set<HostAndPort> set=**new** HashSet<HostAndPort>();

**for** (String node : nodes) {

//node="192.168.243.133:8000"

String host=node.split(":")[0];

Integer port=Integer.

*parseInt*(node.split(":")[1]);

set.add(**new** HostAndPort(host,port));

}

GenericObjectPoolConfig config=**new** GenericObjectPoolConfig();

config.setMaxTotal(maxTotal);

config.setMaxIdle(maxIdle);

config.setMinIdle(minIdle);

**return** **new** JedisCluster(set,config);

}

getter&&setter

}

**2.2测试集群高可用**

访问一个系统(web应用),将数据存储并且从cluster获取回来(set,get)

### 2.2.1 在测试工程中创建启动类

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**package** cn.edu.scnu;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

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

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

**import** redis.clients.jedis.JedisCluster;

@SpringBootApplication

@RestController //为了方便，不单独写控制器类，直接在启动类写。

**public** **class** StarterRedis {

@Autowired

**private** JedisCluster cluster;

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

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

}

// 请求测试redis的读写功能,并且高可用代码逻辑

@RequestMapping("cluster")

**public** String setAndGet(String key, String value) {

cluster.set(key, value);

**return** cluster.get(key);

}

}

### 2.2.2 测试高可用

1）运行启动类，启动测试工程。

在浏览器中输入：

<http://localhost:8095/cluster?key=test&value=abc>

图形用户界面, 文本, 应用程序

描述已自动生成

到集群中查看结果

图片包含 文本

描述已自动生成

2）测试高可用

将8001节点kill掉

文本, 表格

中度可信度描述已自动生成

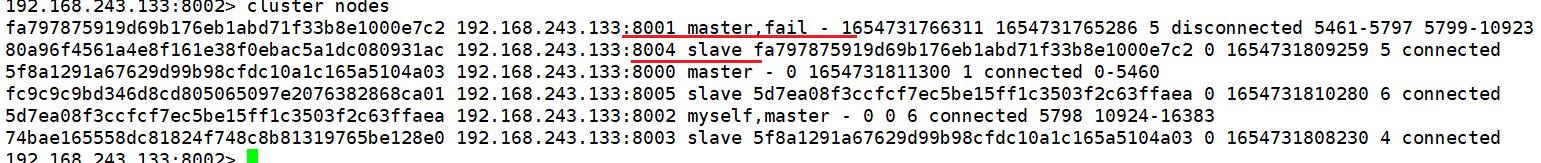
刷新页面出现报错信息

文本

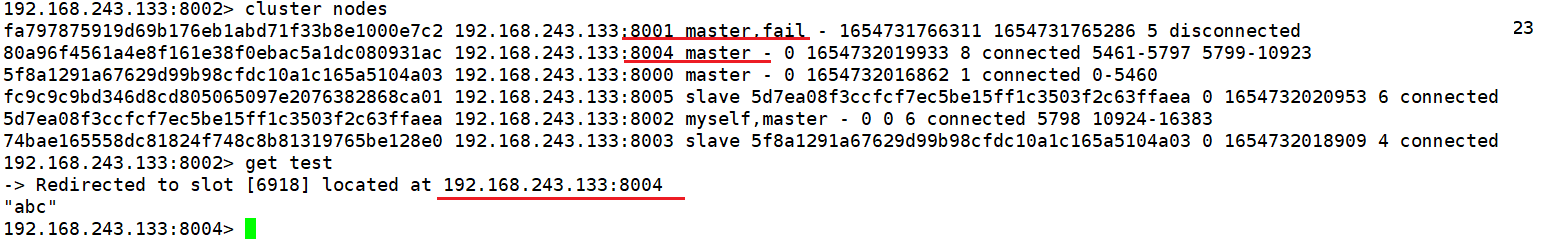
描述已自动生成

查看控制台也出现报错信息

到集群中查看，正在选举。



再次查看，选举完毕，运行get test后发现从8004获取数据：



刷新页面，依然报错。继续刷新，就出来了

图形用户界面, 应用程序, Word

描述已自动生成

测试key=name&value=xulaoshi

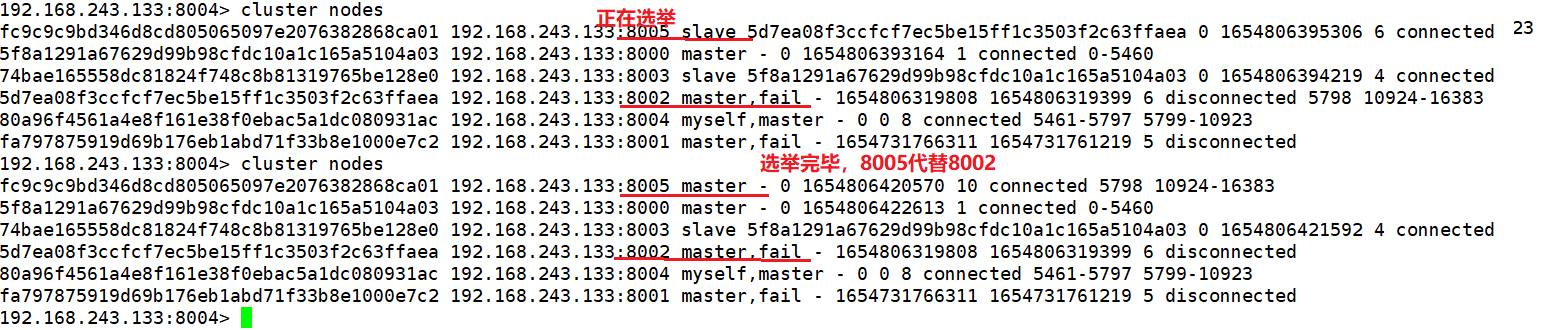
图形用户界面, 文本, 应用程序, Word

描述已自动生成

文本

描述已自动生成

将8002节点kill掉。



刷新页面得到结果：

图形用户界面, 应用程序, 网站

描述已自动生成

8005顶替了8002节点：

图片包含 徽标

描述已自动生成

测试key=haha&value=123

图形用户界面, 文本, 应用程序, Word

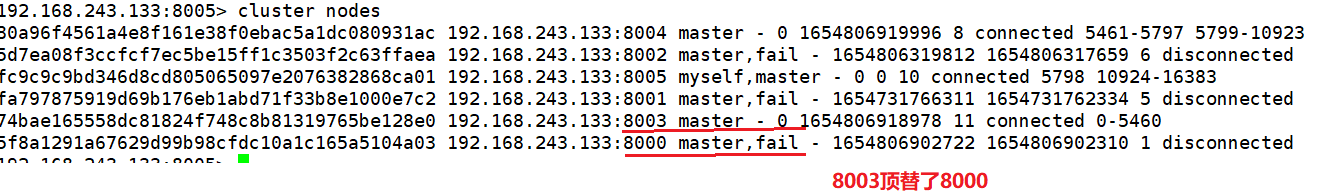
描述已自动生成

haha存在8000节点

文本

中度可信度描述已自动生成

Kill掉8000节点



刷新页面(刷两次，必须失败一次)

图形用户界面, 文本, 应用程序

描述已自动生成

发现haha存在8003节点

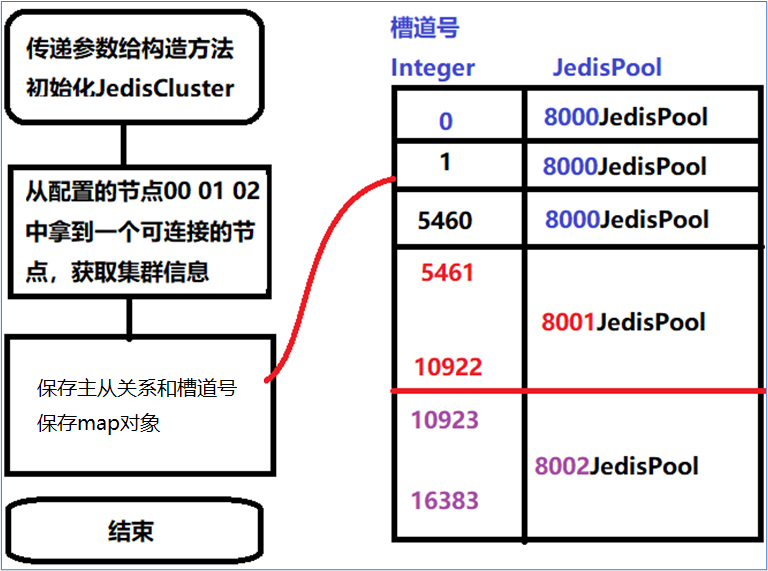
图片包含 图形用户界面

描述已自动生成

此时集群剩下8003-8005节点，依然能正常工作。尽管我们配置文件配置的8000-8002节点都已经down掉，整个集群都依然能正常工作。

**3.JedisCluster原理**

**3.1对象的创建过程**



**3.2执行命令**

set location gz

