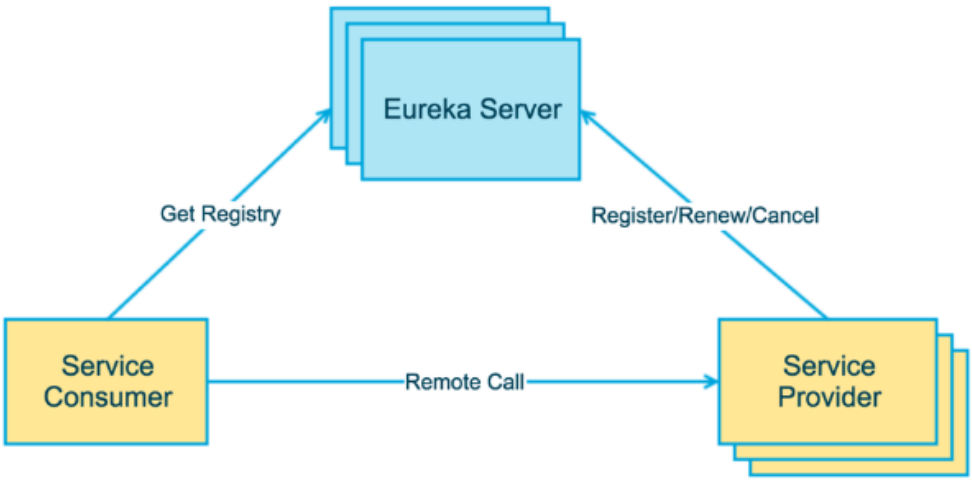
集群,主备,高可用汇总

一,eureka集群

1.eureka介绍

Eureka是Netflix开发的服务发现框架，SpringCloud将它集成在自己的子项目spring-cloud-netflix中，实现SpringCloud的服务发现功能。Eureka包含两个组件：Eureka Server和Eureka Client。



- Eureka Server提供服务注册服务，各个节点启动后，会在Eureka Server中进行注册，这样EurekaServer中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到。
- Eureka Client是一个java客户端，用于简化与Eureka Server的交互，客户端同时也就别一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳,默认周期为30秒，如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，Eureka Server将会从服务注册表中把这个服务节点移除(默认90秒)

2.eureka集群的搭建

2.1说明

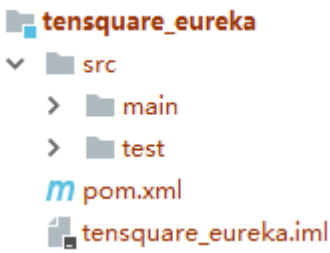
说明: 真实的集群是需要部署在不同的服务器上的，但是在我们测试时同时启动多个个虚拟机内存会吃不消，所以我们通常会搭建伪集群，也就是把所有的服务都搭建在一台虚拟机上，用端口进行区分。

三个Eureka服务端的端口为 6868,6869,6810, 为了效果比较明显, 进行域名映射配置,找到C:\Windows\System32\drivers\etc路径下的hosts文件配置如下

```
127.0.0.1  eureka6868.com
127.0.0.1  eureka6869.com
127.0.0.1  eureka6870.com
```

2.2创建Eureka服务端

- 创建tensquare_eureka模块



- 父工程tensquare_parent的pom.xml定义SpringCloud版本

```
  <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Finchley.M9</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

- 子工程tensquare_eureka添加依赖

```
  <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
 </dependencies>
```

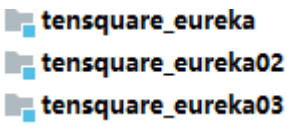
- 子工程tensquare_eureka添加application.yml

```
server:
  port: 6868
eureka:
  instance:
    hostname: eureka6868.com #eureka服务端的实例名称
  client:
    register-with-eureka: false #是否将自己注册到Eureka服务中，本身就是所有无需注册
    fetch-registry: false #是否从Eureka中获取注册信息
    serviceUrl: #Eureka客户端与Eureka服务端进行交互的地址(单机版本配置为: defaultZone:
http://${eureka.instance.hostname}:${server.port}/eureka/ )
      defaultZone: http://eureka6869.com:6869/eureka/,http://eureka6870.com:6870/eureka/
```

- 编写启动类,创建包com.tensquare.eureka， 包下建立类，添加@EnableEurekaServer 开启eureka服务

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

2.3复制二个Eureka服务端,修改配置文件



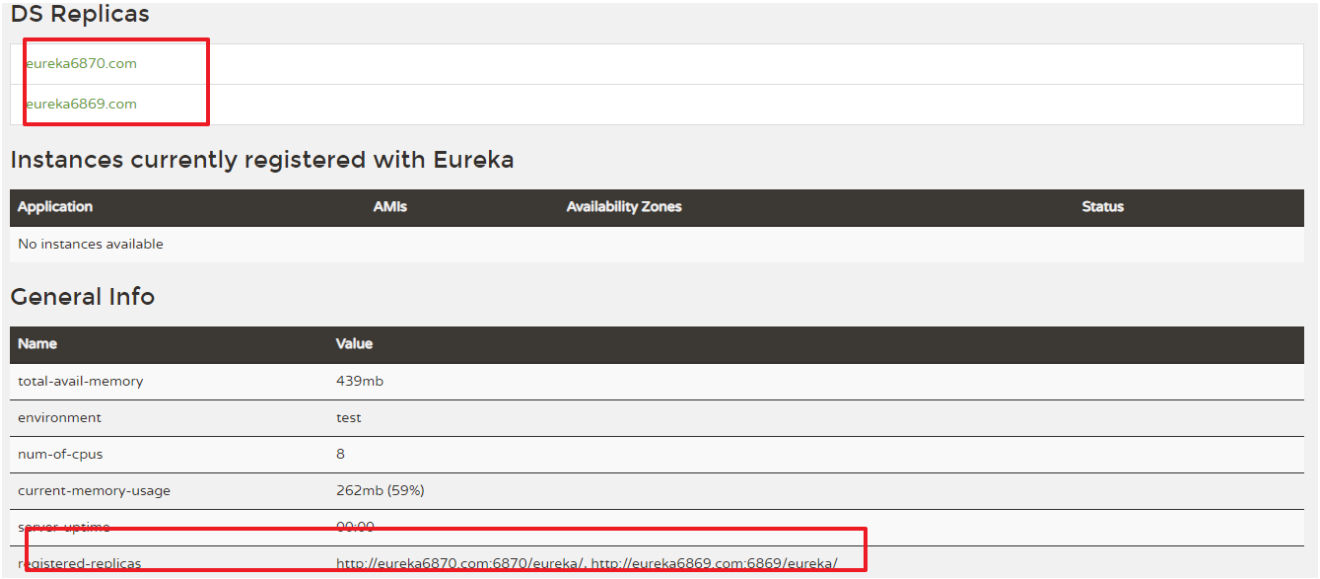
- tensquare_eureka02配置文件

```
server:
  port: 6869
eureka:
  instance:
    hostname: eureka6869.com #eureka服务端的实例名称
  client:
    register-with-eureka: false #是否将自己注册到Eureka服务中，本身就是所有无需注册
    fetch-registry: false #是否从Eureka中获取注册信息
    serviceUrl: #Eureka客户端与Eureka服务端进行交互的地址(单机版本配置为: defaultZone:
http://${eureka.instance.hostname}:${server.port}/eureka/ )
      defaultZone: http://eureka6868.com:6868/eureka/,http://eureka6870.com:6870/eureka/
```

- tensquare_eureka03配置文件

```
server:
  port: 6870
eureka:
  instance:
    hostname: eureka6870.com #eureka服务端的实例名称
  client:
    register-with-eureka: false #是否将自己注册到Eureka服务中，本身就是所有无需注册
    fetch-registry: false #是否从Eureka中获取注册信息
    serviceUrl: #Eureka客户端与Eureka服务端进行交互的地址(单机版本配置为: defaultZone:
http://${eureka.instance.hostname}:${server.port}/eureka/ )
      defaultZone: http://eureka6868.com:6868/eureka/,http://eureka6869.com:6869/eureka/
```

2.4启动三个节点



3,eureka-client连接eureka-server集群

```
eureka:
  client:
    serviceUrl: #Eureka客户端与Eureka服务端进行交互的地址
      defaultZone:
http://eureka6868.com:6868/eureka/,http://eureka6869.com:6869/eureka/,http://eureka6870.com:6870/
eureka/
  instance:
    prefer-ip-address: true #把自己的ip地址报告给Eureka(远程需要,本地测试没有问题)
```

4.作为服务注册中心，Eureka比Zookeeper好在哪里

4.1CAP

- C:Consistency (一致性)

  在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

- A:Availability（可用性):

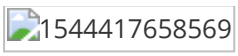
  在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

- P:Partition tolerance（分区容错性）

  以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

  分布式系统不可避免的出现了多个系统通过网络协同工作的场景，结点之间难免会出现网络中断、网延延迟等现
  象，这种现象一旦出现就导致数据被分散在不同的结点上，这就是网络分区。

  CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。因此，根据 CAP 原理将分布式系统分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。



CAP理论就是说在分布式存储系统中，最多只能实现上面的两点。而由于当前的网络硬件肯定会出现延迟丢包等问题，所以分区容错性是我们必须需要实现的。所以我们只能在一致性和可用性之间进行权衡.

CA 传统Oracle数据库

AP 大多数网站架构的选择

CP Redis、Mongodb

4.2Zookeeper和Eureka的区别

- Zookeeper保证CP

当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是zk会出现这样一种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举。问题在于，选举leader的时间太长，30 ~ 120s, 且选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得zk集群失去master节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。

- Eureka保证AP

Eureka看明白了这一点，因此在设计时就优先保证可用性。Eureka各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而Eureka的客户端在向某个Eureka注册或时如果发现连接失败，则会自动切换至其它节点，只要有一台Eureka还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。除此之外，Eureka还有一种自我保护机制，如果在15分钟内超过85%的节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障，此时会出现以下几种情况：

1. Eureka不再从注册列表中移除因为长时间没收到心跳而应该过期的服务
2. Eureka仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上(即保证当前节点依然可用)
3. 当网络稳定时，当前实例新的注册信息会被同步到其它节点中

因此， Eureka可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像zookeeper那样使整个注册服务瘫痪。

二,Zookeeper集群

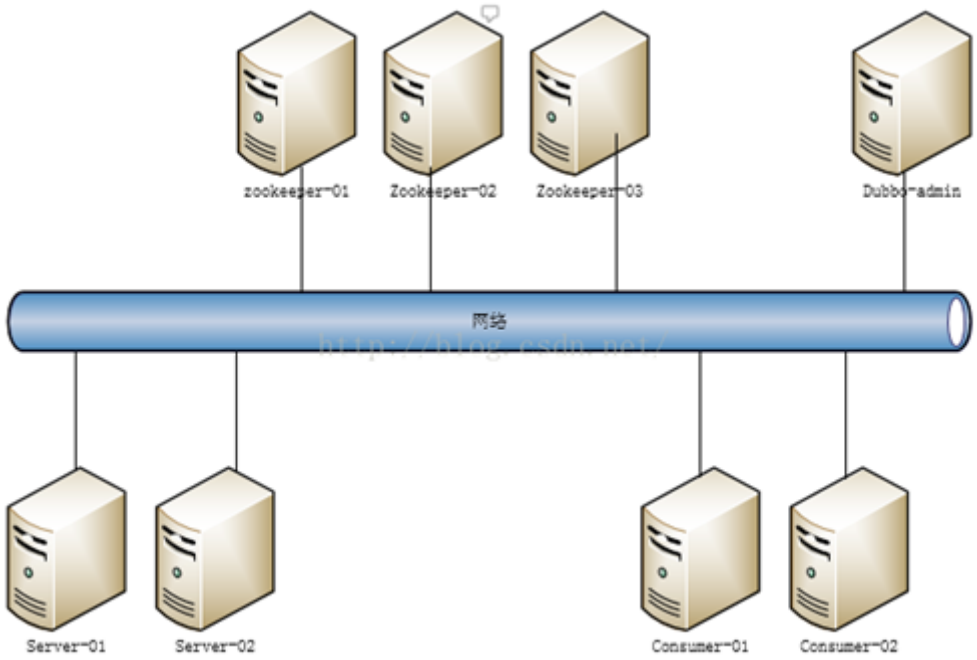
1.Zookeeper集群简介

1.1为什么搭建Zookeeper集群

大部分分布式应用需要一个主控、协调器或者控制器来管理物理分布的子进程。目前，大多数都要开发私有的协调程序，缺乏一个通用机制，协调程序的反复编写浪费，且难以形成通用、伸缩性好的协调器，zookeeper提供通用的分布式锁服务，用以协调分布式应用。所以说zookeeper是分布式应用的协作服务。

zookeeper作为注册中心，服务器和客户端都要访问，如果有大量的并发，肯定会有等待。所以可以通过zookeeper集群解决。

下面是zookeeper集群部署结构图：



1.2Leader选举

Zookeeper的启动过程中leader选举是非常重要而且最复杂的一个环节。那么什么是leader选举呢？zookeeper为什么需要leader选举呢？zookeeper的leader选举的过程又是什么样子的？

首先我们来看看什么是leader选举。其实这个很好理解，leader选举就像总统选举一样，每人一票，获得多数票的人就当选为总统了。在zookeeper集群中也是一样，每个节点都会投票，如果某个节点获得超过半数以上的节点的投票，则该节点就是leader节点了。

以一个简单的例子来说明整个选举的过程.

假设有五台服务器组成的zookeeper集群,它们的id从1-5,同时它们都是最新启动的,也就是没有历史数据,在存放数据量这一点上,都是一样的.假设这些服务器依序启动,来看看会发生什么 。

1) 服务器1启动,此时只有它一台服务器启动了,它发出去的报没有任何响应,所以它的选举状态一直是LOOKING状态

2) 服务器2启动,它与最开始启动的服务器1进行通信,互相交换自己的选举结果,由于两者都没有历史数据,所以id值较大的服务器2胜出,但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是3),所以服务器1,2还是继续保持LOOKING状态.

3) 服务器3启动,根据前面的理论分析,服务器3成为服务器1,2,3中的老大,而与上面不同的是,此时有三台服务器选举了它,所以它成为了这次选举的leader.

4) 服务器4启动,根据前面的分析,理论上服务器4应该是服务器1,2,3,4中最大的,但是由于前面已经有半数以上的服务器选举了服务器3,所以它只能接收当小弟的命了.

5) 服务器5启动,同4一样,当小弟

2.搭建Zookeeper集群

2.1搭建要求

真实的集群是需要部署在不同的服务器上的，但是在我们测试时同时启动十几个虚拟机内存会吃不消，所以我们通常会搭建伪集群，也就是把所有的服务都搭建在一台虚拟机上，用端口进行区分。

我们这里要求搭建一个三个节点的Zookeeper集群（伪集群）。

2.2准备工作

（1）安装JDK 【此步骤省略】。

（2）Zookeeper压缩包上传到服务器

（3）将Zookeeper解压，创建data目录，将 conf下zoo_sample.cfg 文件改名为 zoo.cfg

（4）建立/usr/local/zookeeper-cluster目录，将解压后的Zookeeper复制到以下三个目录

/usr/local/zookeeper-cluster/zookeeper-1

/usr/local/zookeeper-cluster/zookeeper-2

/usr/local/zookeeper-cluster/zookeeper-3

```
[root@localhost ~]# mkdir /usr/local/zookeeper-cluster
[root@localhost ~]# cp -r  zookeeper-3.4.6 /usr/local/zookeeper-cluster/zookeeper-1
[root@localhost ~]# cp -r  zookeeper-3.4.6 /usr/local/zookeeper-cluster/zookeeper-2
[root@localhost ~]# cp -r  zookeeper-3.4.6 /usr/local/zookeeper-cluster/zookeeper-3
```

（5） 配置每一个Zookeeper 的dataDir（zoo.cfg） clientPort 分别为2181 2182 2183

修改/usr/local/zookeeper-cluster/zookeeper-1/conf/zoo.cfg

```
clientPort=2181
dataDir=/usr/local/zookeeper-cluster/zookeeper-1/data
```

修改/usr/local/zookeeper-cluster/zookeeper-2/conf/zoo.cfg

```
clientPort=2182
dataDir=/usr/local/zookeeper-cluster/zookeeper-2/data
```

修改/usr/local/zookeeper-cluster/zookeeper-3/conf/zoo.cfg

```
clientPort=2183
dataDir=/usr/local/zookeeper-cluster/zookeeper-3/data
```

2.3配置集群

（1）在每个zookeeper的 data 目录下创建一个 myid 文件，内容分别是1、2、3 。这个文件就是记录每个服务器的ID

```
-------知识点小贴士------
如果你要创建的文本文件内容比较简单，我们可以通过echo 命令快速创建文件
格式为：
echo 内容 >文件名
例如我们为第一个zookeeper指定ID为1，则输入命令
```

```
[root@localhost data]# echo 1 >myid
[root@localhost data]# cat myid
1
```

（2）在每一个zookeeper 的 zoo.cfg配置客户端访问端口（clientPort）和集群服务器IP列表。

集群服务器IP列表如下

```
server.1=192.168.25.140:2881:3881
server.2=192.168.25.140:2882:3882
server.3=192.168.25.140:2883:3883
```

解释：server.服务器ID=服务器IP地址：服务器之间通信端口：服务器之间投票选举端口

2.4启动集群

(1)启动集群就是分别启动每个实例

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-1/bin/zkServer.sh start
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-1/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh start
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-3/bin/zkServer.sh start
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-3/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

(2)启动后我们查询一下每个实例的运行状态

先查询第一个服务, Mode为follower表示是跟随者（从）

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-1/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-1/bin/../conf/zoo.cfg
Mode: follower
```

再查询第二个服务Mod 为leader表示是领导者（主）

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Mode: leader
```

查询第三个为跟随者（从）

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-3/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-3/bin/../conf/zoo.cfg
Mode: follower
```

2.5模拟集群异常

（1）首先我们先测试如果是从服务器挂掉，会怎么样

把3号服务器停掉，观察1号和2号，发现状态并没有变化

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-3/bin/zkServer.sh stop
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-3/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Mode: leader
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-1/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-1/bin/../conf/zoo.cfg
Mode: follower
```

由此得出结论，3个节点的集群，从服务器挂掉，集群正常

（2）我们再把1号服务器（从服务器）也停掉，查看2号（主服务器）的状态，发现已经停止运行了。

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-1/bin/zkServer.sh stop
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-1/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Error contacting service. It is probably not running.
```

由此得出结论，3个节点的集群，2个从服务器都挂掉，主服务器也无法运行。因为可运行的机器没有超过集群总数量的半数。

（3）我们再次把1号服务器启动起来，发现2号服务器又开始正常工作了。而且依然是领导者。

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-1/bin/zkServer.sh start
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-1/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Mode: leader
```

（4）我们把3号服务器也启动起来，把2号服务器停掉（汗~~干嘛？领导挂了？）停掉后观察1号和3号的状态。

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh stop
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
[root@localhost ~]#
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-1/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-1/bin/../conf/zoo.cfg
Mode: follower
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-3/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-3/bin/../conf/zoo.cfg
Mode: leader
```

发现新的leader产生了~

由此我们得出结论，当集群中的主服务器挂了，集群中的其他服务器会自动进行选举状态，然后产生新得leader

（5）我们再次测试，当我们把2号服务器重新启动起来（汗~~这是诈尸啊!）启动后，会发生什么？2号服务器会再次成为新的领导吗？我们看结果

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh start
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-2/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-2/bin/../conf/zoo.cfg
Mode: follower
```

```
[root@localhost ~]# /usr/local/zookeeper-cluster/zookeeper-3/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper-cluster/zookeeper-3/bin/../conf/zoo.cfg
Mode: leader
```

我们会发现，2号服务器启动后依然是跟随者（从服务器），3号服务器依然是领导者（主服务器），没有撼动3号服务器的领导地位。

由此我们得出结论，当领导者产生后，再次有新服务器加入集群，不会影响到现任领导者。

3.Dubbo连接zookeeper集群

- 修改服务提供者和服务调用者的spring 配置文件

```
<!-- 指定注册中心地址 -->
<dubbo:registry
    protocol="zookeeper" address="192.168.25.140:2181,192.168.25.140:2182,192.168.25.140:2183">
</dubbo:registry>
```

三, Nginx反向代理,负载均衡

1.Nginx

1.1什么是Nginx

Nginx 是一款高性能的 http 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器。由俄罗斯的程序设计师伊戈尔·西索夫（Igor Sysoev）所开发，官方测试 nginx 能够支支撑 5 万并发链接，并且 cpu、内存等资源消耗却非常低，运行非常稳定。

1.2Nginx应用场景

1、http 服务器。Nginx 是一个 http 服务可以独立提供 http 服务。可以做网页静态服务器。

2、虚拟主机。可以实现在一台服务器虚拟出多个网站。例如个人网站使用的虚拟主机。

3、反向代理，负载均衡。当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群可以使用 nginx 做反向代理。并且多台服务器可以平均分担负载，不会因为某台服务器负载高宕机而某台服务器闲置的情况。

1.3Nginx在Linux下的安装

2.安装和使用Nginx

- 进入http://nginx.org/网站，下载nginx-1.13.9.tar.gz文件

- 把安装包上传到Linux

```
crt中 alt+p
```

- 在 usr/local下新建文件夹 nginx

```
mkdir /usr/local/nginx
```

- 将root下的nginx移动到 /usr/local/nginx

```
mv nginx-1.13.9.tar.gz /usr/local/nginx/
```

- 进入/usr/local/nginx, 解包(不要加z)

```
cd /usr/local/nginx/
tar -xvf nginx-1.13.9.tar.gz
```

- 安装Nginx依赖环境gcc

Nginx是C/C++语言开发，建议在Linux上运行，安装Nginx需要先将官网下载的源码进行编译，编译依赖gcc环境，所以需要安装gcc。一直y(同意)(需要网络),

```
yum install gcc-c++
```

- 连接网络，安装Nginx依赖环境pcre/zlib/openssl. y表示安装过程如有提示，默认选择y

```
yum -y install pcre pcre-devel
yum -y install zlib zlib-devel
yum -y install openssl openssl-devel
```

- 编译和安装nginx

```
cd nginx-1.13.9     进入nginx目录
./configure         配置nginx(在nginx-1.13.9目录中执行这个配置文件)
make                    编译nginx
make install        安装nginx
```

- 进去sbin目录,启动

```
cd   /usr/local/nginx/sbin    进入/usr/local/nginx/sbin这个目录
./nginx                                  启动Nginx
```

- 开放Linux的对外访问的端口80，在默认情况下，Linux不会开放端口号80

```
修改配置文件
    cd /etc/sysconfig
    vi iptables
  复制(yy  p)
    -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
  改成
    -A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
  重启加载防火墙或者重启防火墙
    service iptables reload
    或者
    service iptables restart
```

- 停止Nginx服务器

```
cd   /usr/local/nginx/sbin            进入/usr/local/nginx/sbin这个目录
./nginx   -s   stop                   停止Nginx
```

3.Nginx静态网站部署

3.1 静态网站的部署

eg: 将品优购里面生成的静态页（d:\item）上传到服务器的/usr/local/nginx/html下即可访问

3.2端口绑定

（1）上传静态网站：

将前端静态页cart.html 以及图片样式等资源 上传至 /usr/local/nginx/cart 下

将前端静态页search.html 以及图片样式等资源 上传至 /usr/local/nginx/search 下

（2）修改Nginx 的配置文件：/usr/local/nginx/conf/nginx.conf

```
    server {
        listen       81;
        server_name  localhost;
        location / {
            root   cart;
            index  cart.html;
        }
    }
    server {
        listen       82;
        server_name  localhost;
        location / {
            root   search;
            index  search.html;
        }
    }
```

（3）访问测试：

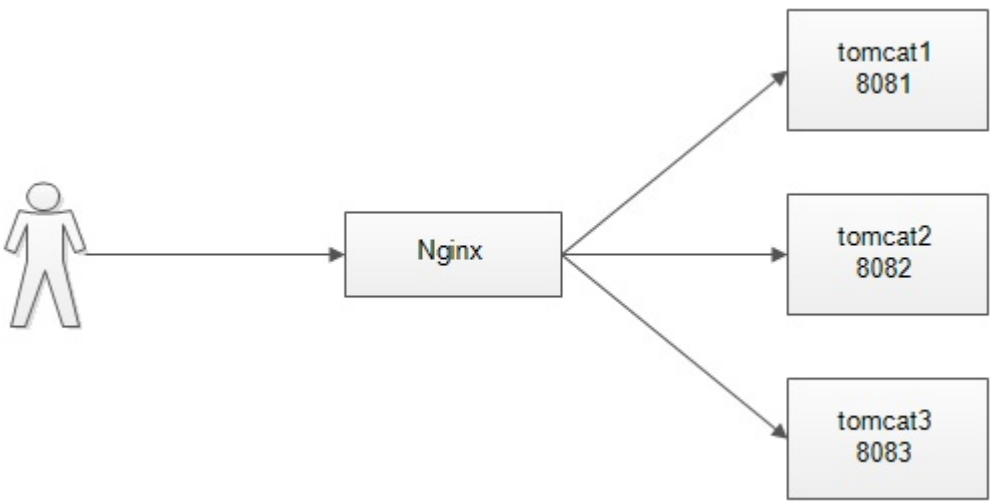
地址栏输入http://192.168.25.141:81 可以看到购物车页面

地址栏输入http://192.168.25.141:82可以看到搜索页面

4.Nginx+tomcat实现集群

4.1配置说明

当我们网站并发量高的时候，一台tomcat无法承受大量并发，可以考虑Nginx+Tomcat集群来实现。咱们这就做一个集群演示。



4.2集群的搭建

- 准备3台tomcat，端口分别是8081、8082、8083，分别在每个tomcat的webapps目录下创建ROOT目录，并创建index.html，分别在html的body里标记1/2/3以示区分。
- 修改Nginx 的配置文件：/usr/local/nginx/conf/nginx.conf

```
    upstream backend{
      server 192.168.25.141:8080 weight=5;
      server 192.168.25.141:8081;
      server 192.168.25.141:8082;
    }


    server {
        listen       80;
        server_name  localhost;

        location / {
             proxy_pass  http://backend;
        }
       }
}
```

5.keepalived+nginx 集群解决单点故障

再牛逼的软件我们也不能保证它一定不挂，为了防止Nginx挂了导致整个服务无法使用的灾难发生，我们这里可以考虑使用Keepalived+Nginx集群实现高可用。

5.1keepalived介绍

Keepalived 是一种高性能的服务器高可用或热备解决方案，Keepalived 可以用来防止服务器单点故障的发生，通过配合 Nginx 可以实现 web 前端服务的高可用。

Keepalived 以 VRRP 协议为实现基础，用 VRRP 协议来实现高可用性(HA)。VRRP(Virtual Router Redundancy Protocol)协议是用于实现路由器冗余的协议，VRRP 协议将两台或多台路由器设备虚拟成一个设备，对外提供虚拟路由器 IP(一个或多个)，而在路由器组内部，如果实际拥有这个对外 IP 的路由器如果工作正常的话就是 MASTER，或者是通过算法选举产生，MASTER 实现针对虚拟路由器 IP 的各种网络功能，如 ARP 请求，ICMP，以及数据的转发等；其他设备不拥有该虚拟 IP，状态是 BACKUP，除了接收 MASTER 的VRRP 状态通告信息外，不执行对外的网络功能。当主机失效时，BACKUP 将接管原先 MASTER 的网络功能。

VRRP 协议使用多播数据来传输 VRRP 数据，VRRP 数据使用特殊的虚拟源 MAC 地址发送数据而不是自身网卡的 MAC 地址，VRRP 运行时只有 MASTER 路由器定时发送 VRRP 通告信息，表示 MASTER 工作正常以及虚拟路由器 IP(组)，BACKUP 只接收 VRRP 数据，不发送数据，如果一定时间内没有接收到 MASTER 的通告信息，各 BACKUP 将宣告自己成为 MASTER，发送通告信息，重新进行 MASTER 选举状态。

5.2准备工作

5.2.1nginx的安装

```
   VIP          |     IP           |       主机名      |      主从
                | 192.168.211.129  |       keep129    |     master
192.168.211.131 |------------------|------------------|--------------
                | 192.168.211.130  |      keep130     |    backup
```

- 在129和130虚拟机上安装nginx，安装过程参考前面学的Nginx。

5.2.2keepalived安装

- 将文件上传到服务器，然后解压安装

```
# cd /usr/local/server
# tar -zxvf keepalived-1.2.18.tar.gz
# mkdir keepalived
# cd keepalived-1.2.18
# ./configure --prefix=/usr/local/server/keepalived
# make && make install
```

- 将 keepalived 安装成 Linux 系统服务,因为没有使用 keepalived 的默认路径安装（默认是/usr/local）,安装完成之后，需要做一些工作复制默认配置文件到默认路径

```
# mkdir /etc/keepalived
# cp /usr/local/server/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/
复制 keepalived 服务脚本到默认的地址
# cp /usr/local/server/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/
# cp /usr/local/server/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
# ln -s /usr/local/sbin/keepalived /usr/sbin/
# ln -s /usr/local/server/keepalived/sbin/keepalived /sbin/
设置 keepalived 服务开机启动
# chkconfig keepalived on
```

5.3配置

- 配置主节点, 找到129节点keepalived的配置文件keepalived.conf(把129虚拟机名字配成keep129)

  cd /usr/local/server/keepalived/etc/keepalived

  ll

```
global_defs {
   router_id keep129;
}

vrrp_script chk_nginx {
 script "/etc/keepalived/nginx_check.sh"
 interval 2
 weight -20
}


vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 129
    mcast_src_ip 192.168.211.129
    priority 100
    nopreempt
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }

    track_script {
     chk_nginx
    }
    virtual_ipaddress {
        192.168.211.131
    }
}
```

- 配置从节点(找到130节点keepalived的配置文件keepalived.conf)

```
global_defs {
   router_id keep130
}

vrrp_script chk_nginx {
 script "/etc/keepalived/nginx_check.sh"
 interval 2
 weight -20
}

vrrp_instance VI_1 {
    state BACKUP
    interface eth2
    virtual_router_id 130
    priority 90
    mcast_src_ip 192.168.211.130
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
    chk_nginx
    }
    virtual_ipaddress {
        192.168.211.131
    }
}
```

- 创建脚本, 在/etc/keepalived目录下创建nginx_check.sh文件

```
#!/bin/bash
A=`ps -C nginx –no-header |wc -l`
if [ $A -eq 0 ];then
    /usr/local/server/nginx/sbin/nginx
    sleep 2
    if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then
        killall keepalived
    fi
fi
```

/usr/local/server/nginx/sbin/nginx 为nginx 的安装目录

附录: 配置说明

```
global_defs {
 ## keepalived 自带的邮件提醒需要开启 sendmail 服务。建议用独立的监控或第三方 SMTP
 router_id keep130 ## 标识本节点的字条串，通常为 hostname
}
## keepalived 会定时执行脚本并对脚本执行的结果进行分析，动态调整 vrrp_instance 的优先级。如果
脚本执行结果为 0，并且 weight 配置的值大于 0，则优先级相应的增加。如果脚本执行结果非 0，并且 weight
配置的值小于 0，则优先级相应的减少。其他情况，维持原本配置的优先级，即配置文件中 priority 对应
的值。
vrrp_script chk_nginx {
 script "/etc/keepalived/nginx_check.sh" ## 检测 nginx 状态的脚本路径
 interval 2 ## 检测时间间隔
 weight -20 ## 如果条件成立，权重-20
}
## 定义虚拟路由，VI_1 为虚拟路由的标示符，自己定义名称
vrrp_instance VI_1 {
 state MASTER ## 主节点为 MASTER，对应的备份节点为 BACKUP
 interface eth1 ## 绑定虚拟 IP 的网络接口，与本机 IP 地址所在的网络接口相同，我的是 eth1
 virtual_router_id 130 ## 虚拟路由的 ID 号，两个节点设置必须一样，可选 IP 最后一段使用，相
同的 VRID 为一个组，他将决定多播的 MAC 地址
 mcast_src_ip 192.168.211.130 ## 本机 IP 地址
 priority 100 ## 节点优先级，值范围 0-254，MASTER 要比 BACKUP 高
nopreempt ## 优先级高的设置 nopreempt 解决异常恢复后再次抢占的问题
advert_int 1 ## 组播信息发送间隔，两个节点设置必须一样，默认 1s
## 设置验证信息，两个节点必须一致
authentication {
 auth_type PASS
 auth_pass 1111 ## 真实生产，按需求对应该过来
}
## 将 track_script 块加入 instance 配置块
 track_script {
 chk_nginx ## 执行 Nginx 监控的服务
}
## 虚拟 IP 池，两个节点设置必须一样
 virtual_ipaddress {
    192.168.199.131 ## 虚拟 ip，可以定义多个
 }
}
```

5.4启动

- 启动nginx和keepalived

```
[root@localhost sbin]# ./nginx
[root@localhost sbin]# keepalived
```

四,ElasticSearch集群

1.集群相关的概念

1.1集群 cluster

一个集群就是由一个或多个节点组织在一起，它们共同持有整个的数据，并一起提供索引和搜索功能。一个集群由一个唯一的名字标识，这个名字默认就是“elasticsearch”。这个名字是重要的，因为一个节点只能通过指定某个集群的名字，来加入这个集群

1.2节点 node

一个节点是集群中的一个服务器，作为集群的一部分，它存储数据，参与集群的索引和搜索功能。和集群类似，一个节点也是由一个名字来标识的，默认情况下，这个名字是一个随机的漫威漫画角色的名字，这个名字会在启动的时候赋予节点。这个名字对于管理工作来说挺重要的，因为在这个管理过程中，你会去确定网络中的哪些服务器对应于Elasticsearch集群中的哪些节点。

一个节点可以通过配置集群名称的方式来加入一个指定的集群。默认情况下，每个节点都会被安排加入到一个叫做“elasticsearch”的集群中，这意味着，如果你在你的网络中启动了若干个节点，并假定它们能够相互发现彼此，它们将会自动地形成并加入到一个叫做“elasticsearch”的集群中。

在一个集群里，只要你想，可以拥有任意多个节点。而且，如果当前你的网络中没有运行任何Elasticsearch节点，这时启动一个节点，会默认创建并加入一个叫做“elasticsearch”的集群。

1.3分片和复制 shards&replicas

一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间，而任一节点都没有这样大的磁盘空间；或者单个节点处理搜索请求，响应太慢。为了解决这个问题，Elasticsearch提供了将索引划分成多份的能力，这些份就叫做分片。当你创建一个索引的时候，你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引”，这个“索引”可以被放置到集群中的任何节点上。分片很重要，主要有两方面的原因： 1）允许你水平分割/扩展你的内容容量。 2）允许你在分片（潜在地，位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量。

至于一个分片怎样分布，它的文档怎样聚合回搜索请求，是完全由Elasticsearch管理的，对于作为用户的你来说，这些都是透明的。

在一个网络/云的环境里，失败随时都可能发生，在某个分片/节点不知怎么的就处于离线状态，或者由于任何原因消失了，这种情况下，有一个故障转移机制是非常有用并且是强烈推荐的。为此目的，Elasticsearch允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。

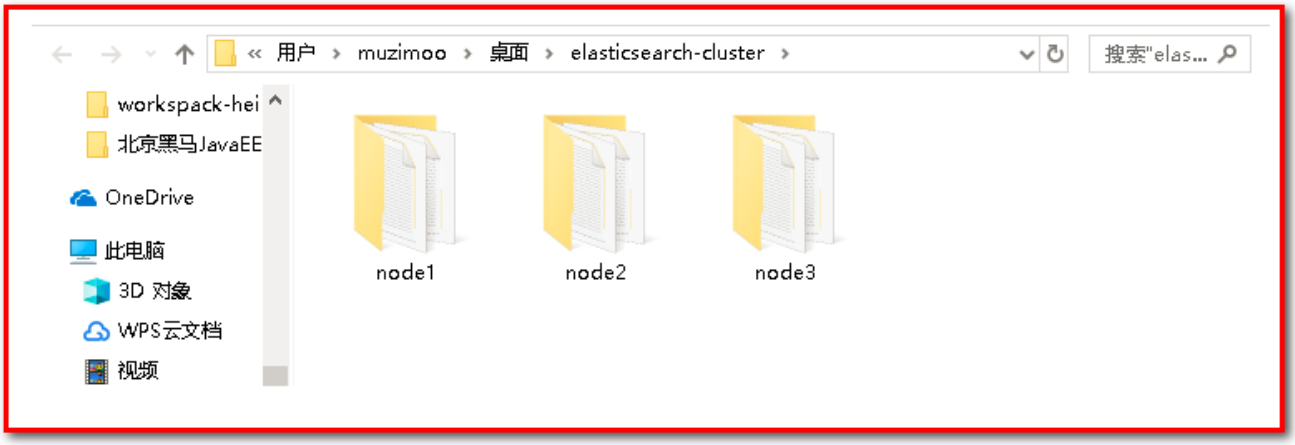
复制之所以重要，有两个主要原因：在分片/节点失败的情况下，提供了高可用性。因为这个原因，注意到复制分片从不与原/主要（original/primary）分片置于同一节点上是非常重要的。扩展你的搜索量/吞吐量，因为搜索可以在所有的复制上并行运行。总之，每个索引可以被分成多个分片。一个索引也可以被复制0次（意思是没有复制）或多次。一旦复制了，每个索引就有了主分片（作为复制源的原来的分片）和复制分片（主分片的拷贝）之别。分片和复制的数量可以在索引创建的时候指定。在索引创建之后，你可以在任何时候动态地改变复制的数量，但是你事后不能改变分片的数量。

默认情况下，Elasticsearch中的每个索引被分片5个主分片和1个复制，这意味着，如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片（1个完全拷贝），这样的话每个索引总共就有10个分片。

2.集群的搭建

2.1准备三台elasticsearch服务器

创建elasticsearch-cluster文件夹，在内部复制三个elasticsearch服务



2.2 修改每台服务器配置

修改elasticsearch-cluster\node*\config\elasticsearch.yml配置文件

- node1节点

```
http.cors.enabled: true
http.cors.allow-origin: "*"


#节点1的配置信息：
#集群名称，保证唯一
cluster.name: my-elasticsearch
#节点名称，必须不一样
node.name: node-1
#必须为本机的ip地址
network.host: 127.0.0.1
#服务端口号，在同一机器下必须不一样
http.port: 9200
#集群间通信端口号，在同一机器下必须不一样
transport.tcp.port: 9300
#设置集群自动发现机器ip集合
discovery.zen.ping.unicast.hosts: ["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
```

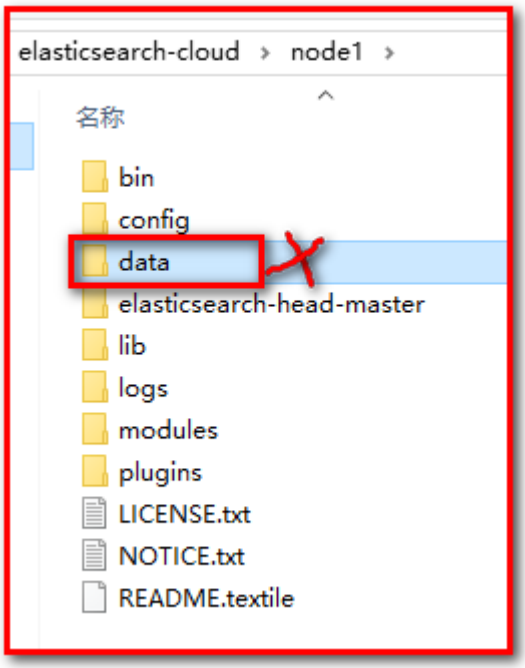
- node2节点

```
http.cors.enabled: true
http.cors.allow-origin: "*"

#节点2的配置信息：
#集群名称，保证唯一
cluster.name: my-elasticsearch
#节点名称，必须不一样
node.name: node-2
#必须为本机的ip地址
network.host: 127.0.0.1
#服务端口号，在同一机器下必须不一样
http.port: 9201
#集群间通信端口号，在同一机器下必须不一样
transport.tcp.port: 9301
#设置集群自动发现机器ip集合
discovery.zen.ping.unicast.hosts: ["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
```

- node3节点

```
http.cors.enabled: true
http.cors.allow-origin: "*"

#节点3的配置信息：
#集群名称，保证唯一
cluster.name: my-elasticsearch
#节点名称，必须不一样
node.name: node-3
#必须为本机的ip地址
network.host: 127.0.0.1
#服务端口号，在同一机器下必须不一样
http.port: 9202
#集群间通信端口号，在同一机器下必须不一样
transport.tcp.port: 9302
#设置集群自动发现机器ip集合
discovery.zen.ping.unicast.hosts: ["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
```

注意：这里需要先删除node1，node2，node3的data文件，防止数据冲突

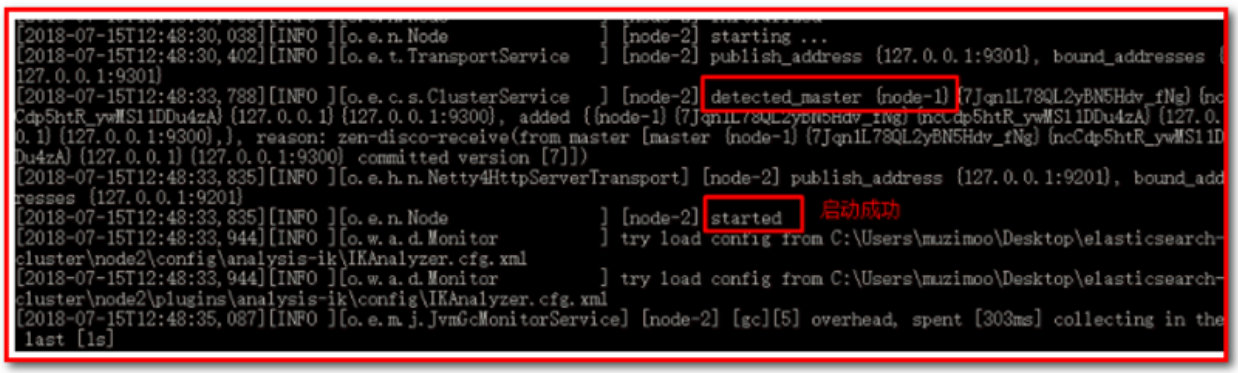


2.3启动各个节点服务器

双击elasticsearch-cluster\node*\bin\elasticsearch.bat

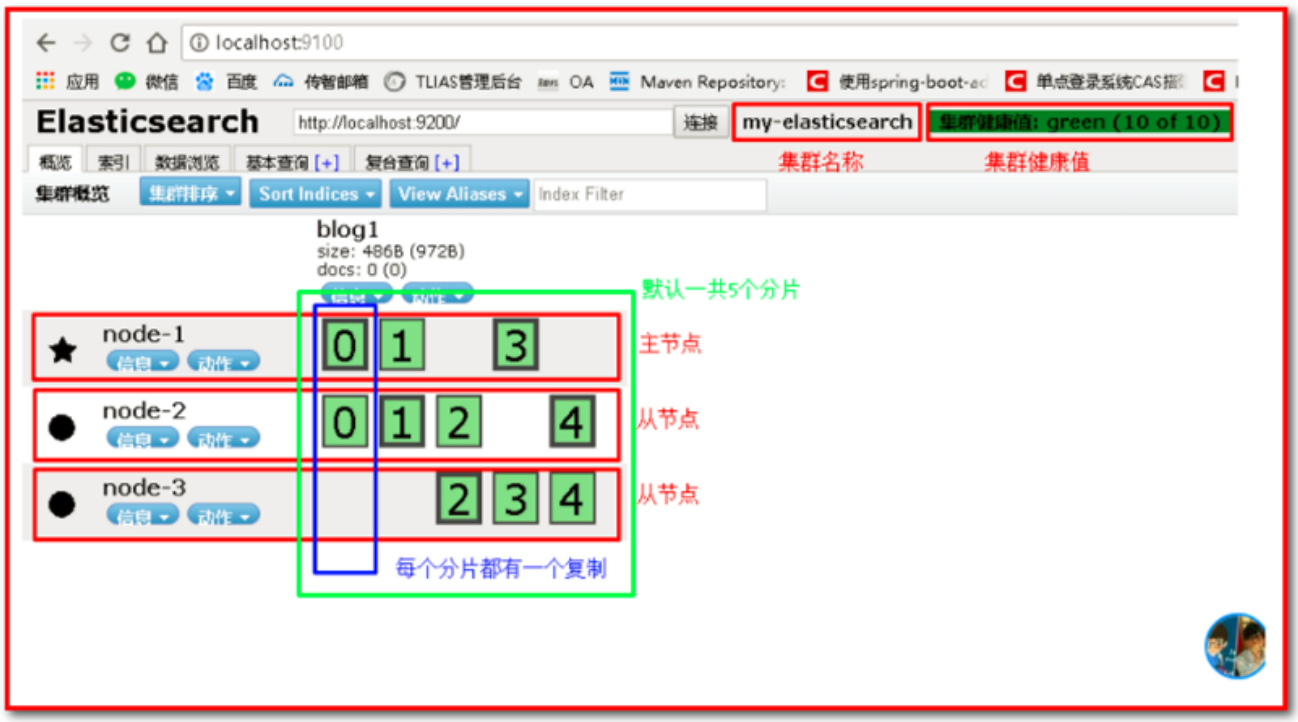
- 启动节点1

- 启动节点2



- 启动节点3

2.4创建索引,查看



五,SolrCloud

1. SolrCloud简介

1.1什么是SolrCloud

SolrCloud(solr 云)是 Solr 提供的分布式搜索方案，当你需要大规模，容错，分布式索引和检索能力时使用 SolrCloud。当一个系统的索引数据量少的时候是不需要使用 SolrCloud的，当索引量很大，搜索请求并发很高，这时需要使用 SolrCloud 来满足这些需求。

SolrCloud 是基于 Solr 和Zookeeper的分布式搜索方案，它的主要思想是使用 Zookeeper作为集群的配置信息中心。

它有几个特色功能：

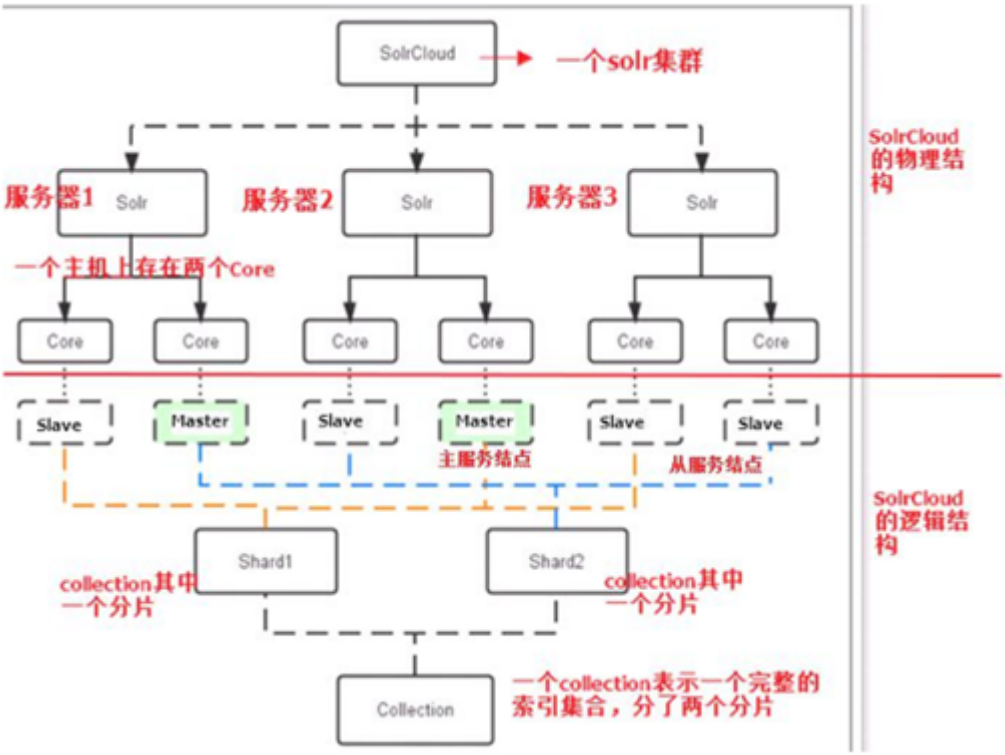
1）集中式的配置信息

2）自动容错

3）近实时搜索

4）查询时自动负载均衡

1.2SolrCloud系统架构

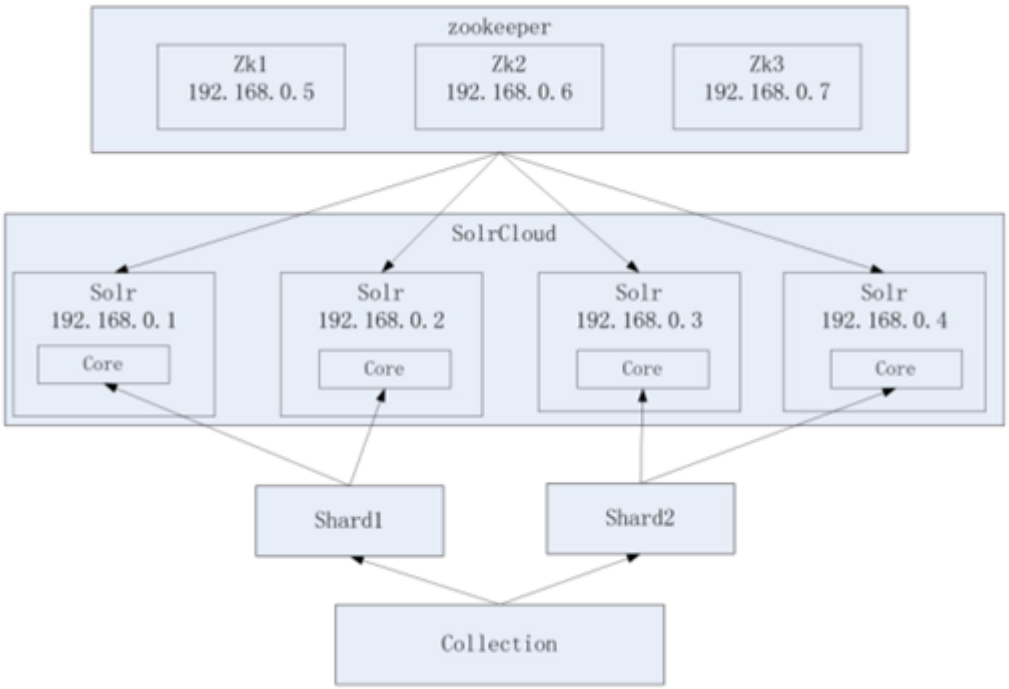


【1】物理结构
三个 Solr 实例（ 每个实例包括两个 Core），组成一个 SolrCloud。
【2】逻辑结构
索引集合包括两个 Shard（shard1 和 shard2），shard1 和 shard2 分别由三个 Core 组成，其中一个 Leader 两个 Replication，Leader 是由 zookeeper 选举产生，zookeeper 控制每个shard上三个 Core 的索引数据一致，解决高可用问题。
用户发起索引请求分别从 shard1 和 shard2 上获取，解决高并发问题。
（1）Collection

Collection 在 SolrCloud 集群中是一个逻辑意义上的完整的索引结构。它常常被划分为一个或多个 Shard（分片），它们使用相同的配置信息。
比如：针对商品信息搜索可以创建一个 collection。
collection=shard1+shard2+....+shardX
（2） Core
每个 Core 是 Solr 中一个独立运行单位，提供 索引和搜索服务。一个 shard 需要由一个Core 或多个 Core 组成。
由于 collection 由多个 shard 组成所以 collection 一般由多个 core 组成。
（3）Master 或 Slave
Master 是 master-slave 结构中的主结点（通常说主服务器），Slave 是 master-slave 结构中的从结点（通常说从服务器或备服务器）。同一个 Shard 下 master 和 slave 存储的数据是一致的，这是为了达到高可用目的。
（4）Shard
Collection 的逻辑分片。每个 Shard 被化成一个或者多个 replication，通过选举确定哪个是 Leader。

2.搭建SolrCloud

2.1搭建要求



Zookeeper 作为集群的管理工具

1、集群管理：容错、负载均衡。

2、配置文件的集中管理

3、集群的入口

需要实现 zookeeper 高可用，需要搭建zookeeper集群。建议是奇数节点。需要三个 zookeeper 服务器。

搭建 solr 集群需要 7 台服务器（搭建伪分布式，建议虚拟机的内存 1G 以上）：

需要三个 zookeeper 节点

需要四个 tomcat 节点。

2.2准备工作

环境准备

CentOS-6.5-i386-bin-DVD1.iso

jdk-7u72-linux-i586.tar.gz

apache-tomcat-7.0.47.tar.gz

zookeeper-3.4.6.tar.gz

solr-4.10.3.tgz

步骤：

（1）搭建Zookeeper集群（我们在上一小节已经完成）

（2）将已经部署完solr 的tomcat的上传到linux

（3）在linux中创建文件夹/usr/local/solr-cloud 创建4个tomcat实例

```
[root@localhost ~]# mkdir /usr/local/solr-cloud
[root@localhost ~]# cp -r tomcat-solr /usr/local/solr-cloud/tomcat-1
[root@localhost ~]# cp -r tomcat-solr /usr/local/solr-cloud/tomcat-2
[root@localhost ~]# cp -r tomcat-solr /usr/local/solr-cloud/tomcat-3
[root@localhost ~]# cp -r tomcat-solr /usr/local/solr-cloud/tomcat-4
```

（4）将本地的solrhome上传到linux

（5）在linux中创建文件夹/usr/local/solrhomes ,将solrhome复制4份

```
[root@localhost ~]# mkdir /usr/local/solrhomes
[root@localhost ~]# cp -r solrhome /usr/local/solrhomes/solrhome-1
[root@localhost ~]# cp -r solrhome /usr/local/solrhomes/solrhome-2
[root@localhost ~]# cp -r solrhome /usr/local/solrhomes/solrhome-3
[root@localhost ~]# cp -r solrhome /usr/local/solrhomes/solrhome-4
```

（6）修改每个solr的 web.xml 文件, 关联solrhome

```
    <env-entry>
       <env-entry-name>solr/home</env-entry-name>
       <env-entry-value>/usr/local/solrhomes/solrhome-1</env-entry-value>
       <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
```

（7）修改每个tomcat的原运行端口8085 8080 8009 ， 分别为

8185 8180 8109

8285 8280 8209

8385 8380 8309

8485 8480 8409

------ 说明 ------
8005端口是用来关闭TOMCAT服务的端口。
8080端口，负责建立HTTP连接。在通过浏览器访问Tomcat服务器的Web应用时，使用的就是这个连接器。
8009端口，负责和其他的HTTP服务器建立连接。在把Tomcat与其他HTTP服务器集成时，就需要用到这个连接器。

2.3配置集群

（1）修改每个 tomcat实例 bin 目录下的 catalina.sh 文件

把此配置添加到catalina.sh中(第234行) ：

```
JAVA_OPTS="-DzkHost=192.168.25.140:2181,192.168.25.140:2182,192.168.25.140:2183"
```

JAVA_OPTS ,顾名思义,是用来设置JVM相关运行参数的变量 . 此配置用于在tomcat启动时找到 zookeeper集群。

（2）配置 solrCloud 相关的配置。每个solrhome 下都有一个 solr.xml，把其中的 ip 及端口号配置好（是对应的tomcat的IP和端口）。

solrhomes/solrhome-1/solr.xml

```
  <solrcloud>
    <str name="host">192.168.25.140</str>
    <int name="hostPort">8180</int>
    <str name="hostContext">${hostContext:solr}</str>
    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
  </solrcloud>
```

solrhomes/solrhome-2/solr.xml

```
 <solrcloud>
    <str name="host">192.168.25.140</str>
    <int name="hostPort">8280</int>
    <str name="hostContext">${hostContext:solr}</str>
    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
  </solrcloud>
```

solrhomes/solrhome-3/solr.xml

```
<solrcloud>
    <str name="host">192.168.25.140</str>
    <int name="hostPort">8380</int>
    <str name="hostContext">${hostContext:solr}</str>
    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
  </solrcloud>
```

solrhomes/solrhome-4/solr.xml

```
  <solrcloud>
    <str name="host">192.168.25.140</str>
    <int name="hostPort">8480</int>
    <str name="hostContext">${hostContext:solr}</str>
    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
  </solrcloud>
```

（3）让 zookeeper 统一管理配置文件。需要把solrhome下collection1/conf 目录上传到

zookeeper。上传任意 solrhome 中的配置文件即可。

我们需要使用solr给我们提供的工具上传配置文件：

solr-4.10.3/example/scripts/cloud-scripts/zkcli.sh

将solr-4.10.3压缩包上传到linux，解压，然后进入solr-4.10.3/example/scripts/cloud-scripts目录，执行下列命令

```
./zkcli.sh -zkhost 192.168.25.140:2181,192.168.25.140:2182,192.168.25.140:2183 -cmd upconfig
-confdir /usr/local/solrhomes/solrhome-1/collection1/conf -confname myconf
```

参数解释

-zkhost ： 指定zookeeper地址列表

-cmd ： 指定命令。upconfig 为上传配置的命令

-confdir : 配置文件所在目录

-confname : 配置名称

2.4启动集群

（1）启动每个 tomcat 实例。要保证 zookeeper集群是启动状态。

```
----  知识点小贴士 -----
如果你想让某个文件夹下都可以执行，使用以下命令实现
chmod -R 777 solr-cloud
```

（2）访问集群

地址栏输入 http://192.168.25.140:8180/solr ，可以看到Solr集群版的界面

下图表示的是，一个主节点，三个从节点。

3. SpringDataSolr连接SolrCloud

在SolrJ中提供一个叫做CloudSolrServer的类，它是SolrServer的子类，用于连接solrCloud

它的构造参数就是zookeeper的地址列表，另外它要求要指定defaultCollection属性（默认的 collection名称）

我们现在修改springDataSolrDemo工程的配置文件，把原来的solr-server注销，替换为CloudSolrServer .指定构造参数为地址列表，设置默认 collection名称

```
<!-- solr服务器地址
<solr:solr-server id="solrServer" url="http://192.168.25.129:8080/solr" />
 -->
<bean id="solrServer" class="org.apache.solr.client.solrj.impl.CloudSolrServer">
    <constructor-arg value="192.168.25.140:2181,192.168.25.140:2182,192.168.25.140:2183" />
    <property name="defaultCollection" value="collection1"></property>
</bean>
```

4.分片配置

（1）创建新的 Collection 进行分片处理。

在浏览器输入以下地址，可以按照我们的要求 创建新的Collection

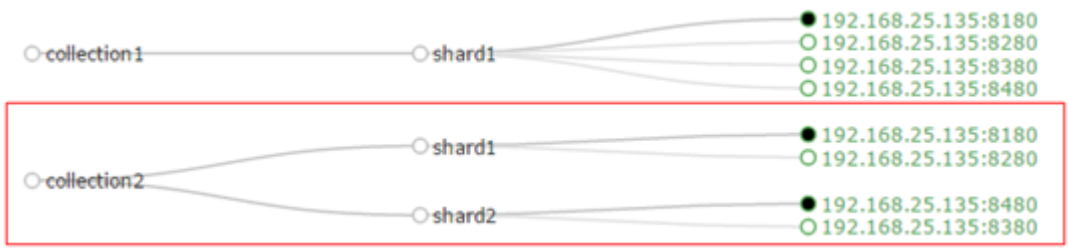
```
http://192.168.25.140:8180/solr/admin/collections?
action=CREATE&name=collection2&numShards=2&replicationFactor=2
```

参数：

name:将被创建的集合的名字
numShards:集合创建时需要创建逻辑碎片的个数
replicationFactor:分片的副本数。

看到这个提示表示成功

```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">5912</int>
  </lst>
  <lst name="success">
    <lst>
      <lst name="responseHeader">
        <int name="status">0</int>
        <int name="QTime">4468</int>
      </lst>
      <str name="core">collection2_shard1_replica1</str>
    </lst>
```



（2）删除不用的 Collection。执行以下命令

```
http://192.168.25.140:8480/solr/admin/collections?action=DELETE&name=collection1
```

```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">699</int>
  </lst>
  <lst name="success">
    <lst name="192.168.25.135:8480_solr">
      <lst name="responseHeader">
        <int name="status">0</int>
        <int name="QTime">126</int>
      </lst>
    </lst>
```

5.模拟集群异常测试

（1）停止第一个tomcat节点，看查询是否能正常工作 -- 能！因为还有从节点

（2）停止第三个tomcat节点，看看查询能够正常工作 -- 不能，因为整个一片数据全没了，无法正常工作。

（3）恢复第三个tomcat节点，看看能否正常工作。恢复时间会比较长，大概2分半到3分钟之间。请耐心等待。

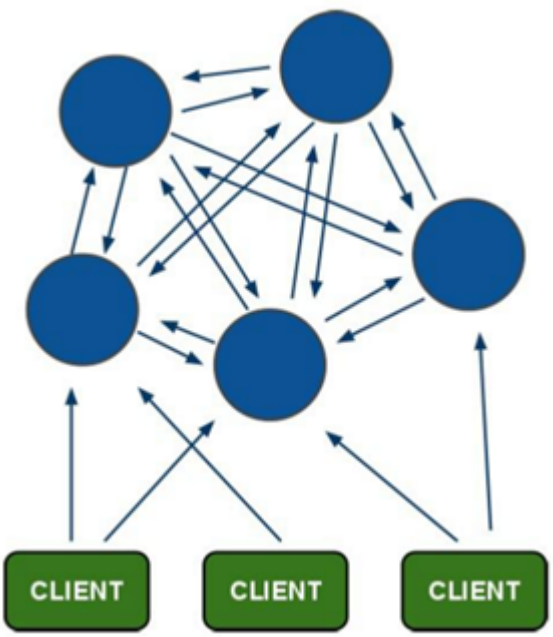
六,Redis Cluster

1. Redis-Cluster简介

1.1什么是Redis-Cluster

为何要搭建Redis集群。Redis是在内存中保存数据的，而我们的电脑一般内存都不大，这也就意味着Redis不适合存储大数据，适合存储大数据的是Hadoop生态系统的Hbase或者是MogoDB。Redis更适合处理高并发，一台设备的存储能力是很有限的，但是多台设备协同合作，就可以让内存增大很多倍，这就需要用到集群。

Redis集群搭建的方式有多种，例如使用客户端分片、Twemproxy、Codis等，但从redis 3.0之后版本支持redis-cluster集群，它是Redis官方提出的解决方案，Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。其redis-cluster架构图如下：



客户端与 redis 节点直连,不需要中间 proxy 层.客户端不需要连接集群所有节点连接集群中任何一个可用节点即可。

所有的 redis 节点彼此互联(PING-PONG 机制),内部使用二进制协议优化传输速度和带宽.

1.2分布存储机制-槽

（1）redis-cluster 把所有的物理节点映射到[0-16383]slot 上,cluster 负责维护
node<->slot<->value

（2）Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点。

例如三个节点：槽分布的值如下

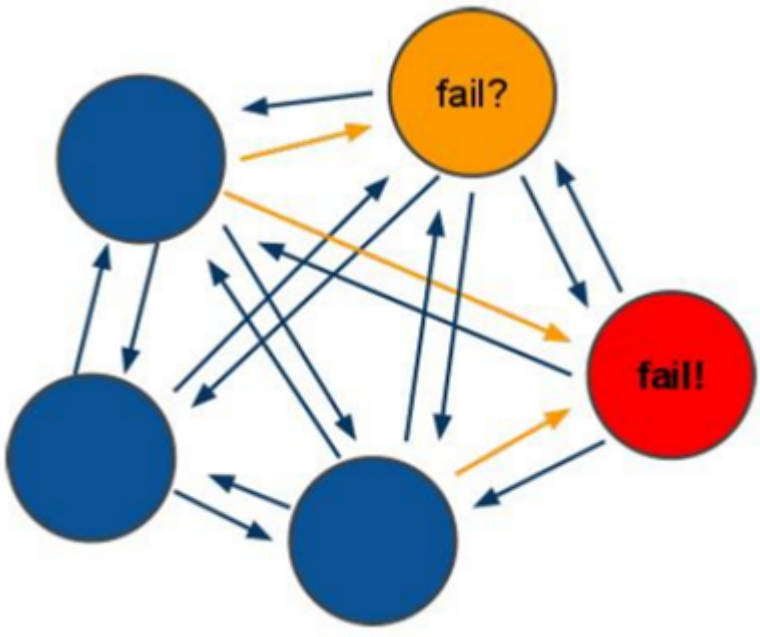
SERVER1: 0-5460
SERVER2: 5461-10922
SERVER3: 10923-16383

1.3容错机制-投票

（1）选举过程是集群中所有master参与,如果半数以上master节点与故障节点通信超过(cluster-node-timeout),认为该节点故障，自动触发故障转移操作. 故障节点对应的从节点自动升级为主节点

（2）什么时候整个集群不可用(cluster_state:fail)?

如果集群任意master挂掉,且当前master没有slave.集群进入fail状态,也可以理解成集群的slot映射[0-16383]不完成时进入fail状态.



2.搭建Redis-Cluster

2.1搭建要求

需要 6 台 redis 服务器。搭建伪集群。

需要 6 个 redis 实例。

需要运行在不同的端口 7001-7006

2.2准备工作

（1）安装gcc 【此步省略】

Redis 是 c 语言开发的。安装 redis 需要 c 语言的编译环境。如果没有 gcc 需要在线安装。

```
yum install gcc-c++
```

（2）使用yum命令安装 ruby （我们需要使用ruby脚本来实现集群搭建）【此步省略】

```
yum install ruby
yum install rubygems
```

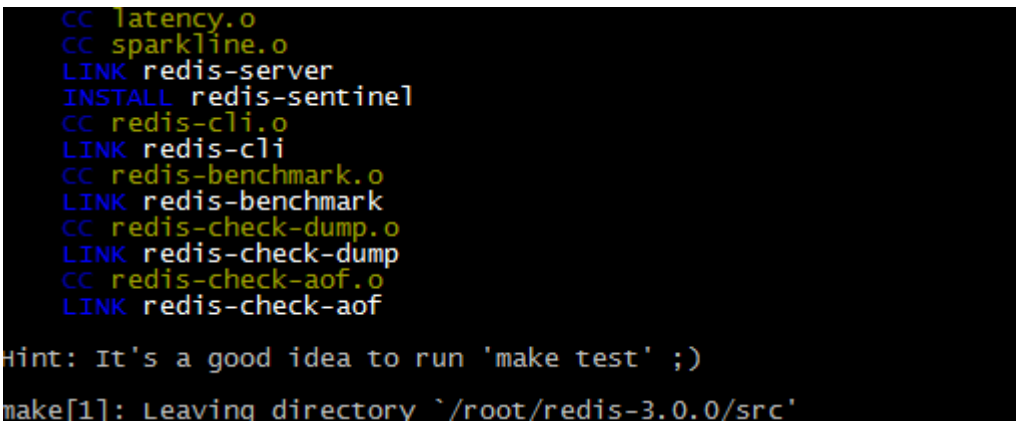
```
----- 知识点小贴士 -----
Ruby，一种简单快捷的面向对象（面向对象程序设计）脚本语言，在20世纪90年代由日本人松本行弘(Yukihiro Matsumoto)开发，遵守GPL协议和Ruby License。它的灵感与特性来自于 Perl、Smalltalk、Eiffel、Ada以及 Lisp 语言。由 Ruby 语言本身还发展出了JRuby（Java平台）、IronRuby（.NET平台）等其他平台的 Ruby 语言替代品。
Ruby的作者于1993年2月24日开始编写Ruby，直至1995年12月才正式公开发布于fj（新闻组）。因为Perl发音与6月诞生石pearl（珍珠）相同，因此Ruby以7月诞生石ruby（红宝石）命名
RubyGems简称gems，是一个用于对 Ruby组件进行打包的 Ruby 打包系统
```

（3）将redis源码包上传到 linux 系统 ，解压redis源码包

（4）编译redis源码 ，进入redis源码文件夹

```
make
```

看到以下输出结果，表示编译成功



（5）创建目录/usr/local/redis-cluster目录， 安装6个redis实例，分别安装在以下目录

/usr/local/redis-cluster/redis-1

/usr/local/redis-cluster/redis-2

/usr/local/redis-cluster/redis-3

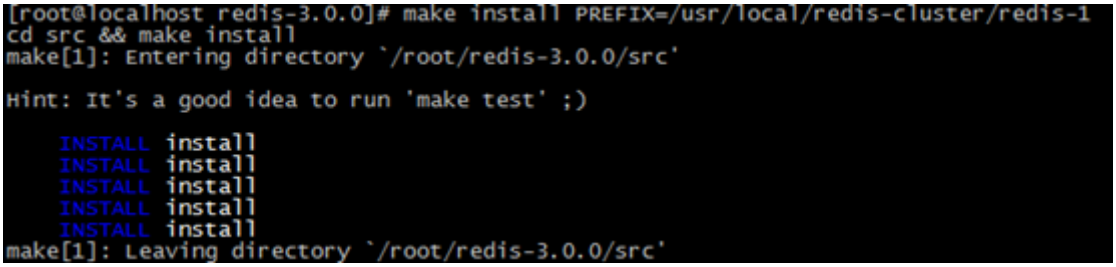
/usr/local/redis-cluster/redis-4

/usr/local/redis-cluster/redis-5

/usr/local/redis-cluster/redis-6

以第一个redis实例为例，命令如下

```
make install PREFIX=/usr/local/redis-cluster/redis-1
```



出现此提示表示成功，按此方法安装其余5个redis实例

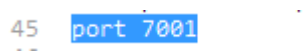
（6）复制配置文件 将 /redis-3.0.0/redis.conf 复制到redis下的bin目录下

```
[root@localhost redis-3.0.0]# cp redis.conf /usr/local/redis-cluster/redis-1/bin
[root@localhost redis-3.0.0]# cp redis.conf /usr/local/redis-cluster/redis-2/bin
[root@localhost redis-3.0.0]# cp redis.conf /usr/local/redis-cluster/redis-3/bin
[root@localhost redis-3.0.0]# cp redis.conf /usr/local/redis-cluster/redis-4/bin
[root@localhost redis-3.0.0]# cp redis.conf /usr/local/redis-cluster/redis-5/bin
[root@localhost redis-3.0.0]# cp redis.conf /usr/local/redis-cluster/redis-6/bin
```

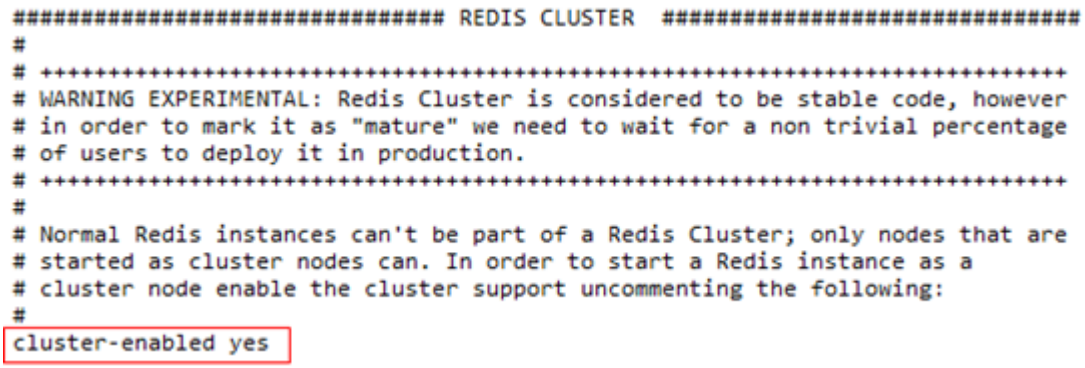
2.3配置集群

（1）修改每个redis节点的配置文件redis.conf

修改运行端口为7001 （7002 7003 .....）



将cluster-enabled yes 前的注释去掉(632行)



（2）启动每个redis实例

以第一个实例为例，命令如下

```
cd /usr/local/redis-cluster/redis-1/bin/
./redis-server redis.conf
```

把其余的5个也启动起来，然后查看一下是不是都启动起来了

```
[root@localhost ~]# ps -ef | grep redis
root     15776 15775  0 08:19 pts/1    00:00:00 ./redis-server *:7001 [cluster]
root     15810 15784  0 08:22 pts/2    00:00:00 ./redis-server *:7002 [cluster]
root     15831 15813  0 08:23 pts/3    00:00:00 ./redis-server *:7003 [cluster]
root     15852 15834  0 08:23 pts/4    00:00:00 ./redis-server *:7004 [cluster]
root     15872 15856  0 08:24 pts/5    00:00:00 ./redis-server *:7005 [cluster]
root     15891 15875  0 08:24 pts/6    00:00:00 ./redis-server *:7006 [cluster]
root     15926 15895  0 08:24 pts/7    00:00:00 grep redis
```

（3）上传redis-3.0.0.gem，安装 ruby用于搭建redis集群的脚本。

```
[root@localhost ~]# gem install redis-3.0.0.gem
Successfully installed redis-3.0.0
1 gem installed
Installing ri documentation for redis-3.0.0...
Installing RDoc documentation for redis-3.0.0...
```

（4）使用 ruby 脚本搭建集群。

进入redis源码目录中的src目录 执行下面的命令

```
./redis-trib.rb create --replicas 1 192.168.25.140:7001 192.168.25.140:7002 192.168.25.140:7003
192.168.25.140:7004 192.168.25.140:7005 192.168.25.140:7006
```

出现下列提示信息

```
>>> Creating cluster
Connecting to node 192.168.25.140:7001: OK
Connecting to node 192.168.25.140:7002: OK
Connecting to node 192.168.25.140:7003: OK
Connecting to node 192.168.25.140:7004: OK
Connecting to node 192.168.25.140:7005: OK
Connecting to node 192.168.25.140:7006: OK
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
192.168.25.140:7001
192.168.25.140:7002
192.168.25.140:7003
Adding replica 192.168.25.140:7004 to 192.168.25.140:7001
Adding replica 192.168.25.140:7005 to 192.168.25.140:7002
Adding replica 192.168.25.140:7006 to 192.168.25.140:7003
M: 1800237a743c2aa918ade045a28128448c6ce689 192.168.25.140:7001
   slots:0-5460 (5461 slots) master
M: 7cb3f7d5c60bfbd3ab28800f8fd3bf6de005bf0d 192.168.25.140:7002
   slots:5461-10922 (5462 slots) master
M: 436e88ec323a2f8bb08bf09f7df07cc7909fcf81 192.168.25.140:7003
   slots:10923-16383 (5461 slots) master
S: c2a39a94b5f41532cd83bf6643e98fc277c2f441 192.168.25.140:7004
   replicates 1800237a743c2aa918ade045a28128448c6ce689
S: b0e38d80273515c84b1a01820d8ecee04547d776 192.168.25.140:7005
   replicates 7cb3f7d5c60bfbd3ab28800f8fd3bf6de005bf0d
S: 03bf6bd7e3e6eece5a02043224497c2c8e185132 192.168.25.140:7006
   replicates 436e88ec323a2f8bb08bf09f7df07cc7909fcf81
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join....
>>> Performing Cluster Check (using node 192.168.25.140:7001)
M: 1800237a743c2aa918ade045a28128448c6ce689 192.168.25.140:7001
   slots:0-5460 (5461 slots) master
M: 7cb3f7d5c60bfbd3ab28800f8fd3bf6de005bf0d 192.168.25.140:7002
   slots:5461-10922 (5462 slots) master
M: 436e88ec323a2f8bb08bf09f7df07cc7909fcf81 192.168.25.140:7003
   slots:10923-16383 (5461 slots) master
M: c2a39a94b5f41532cd83bf6643e98fc277c2f441 192.168.25.140:7004
   slots: (0 slots) master
   replicates 1800237a743c2aa918ade045a28128448c6ce689
M: b0e38d80273515c84b1a01820d8ecee04547d776 192.168.25.140:7005
   slots: (0 slots) master
   replicates 7cb3f7d5c60bfbd3ab28800f8fd3bf6de005bf0d
M: 03bf6bd7e3e6eece5a02043224497c2c8e185132 192.168.25.140:7006
   slots: (0 slots) master
   replicates 436e88ec323a2f8bb08bf09f7df07cc7909fcf81
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

3.连接Redis-Cluster

3.1客户端工具连接

Redis-cli 连接集群：

```
redis-cli -p 主机ip -p 端口（集群中任意端口） -c
```

-c：代表连接的是 redis 集群

测试值的存取:

（1）从本地连接到集群redis 使用7001端口 加 -c 参数

（2）存入name值为abc ，系统提示此值被存入到了7002端口所在的redis （槽是5798）

（3）提取name的值，可以提取。

（4）退出（quit）

（5）再次以7001端口进入 ，不带-c

（6）查询name值，无法获取，因为值在7002端口的redis上

（7）我们以7002端口进入，获取name值发现是可以获取的,而以其它端口进入均不能获取

3.2SpringDataRedis连接Redis集群

修改配置文件applicationContext-redis-cluster.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- 加载配置属性文件 -->
<context:property-placeholder ignore-unresolvable="true" location="classpath:properties/redis-
cluster-config.properties" />
<bean id="redis-clusterConfiguration" class="org.springframework.data.redis.connection.redis-
clusterConfiguration">
    <property name="maxRedirects" value="${redis.maxRedirects}"></property>
    <property name="clusterNodes">
    <set>
        <bean class="org.springframework.data.redis.connection.redis-clusterNode">
            <constructor-arg name="host" value="${redis.host1}"></constructor-arg>
            <constructor-arg name="port" value="${redis.port1}"></constructor-arg>
        </bean>
        <bean class="org.springframework.data.redis.connection.redis-clusterNode">
            <constructor-arg name="host" value="${redis.host2}"></constructor-arg>
            <constructor-arg name="port" value="${redis.port2}"></constructor-arg>
        </bean>
        <bean class="org.springframework.data.redis.connection.redis-clusterNode">
            <constructor-arg name="host" value="${redis.host3}"></constructor-arg>
            <constructor-arg name="port" value="${redis.port3}"></constructor-arg>
        </bean>
         <bean class="org.springframework.data.redis.connection.redis-clusterNode">
            <constructor-arg name="host" value="${redis.host4}"></constructor-arg>
            <constructor-arg name="port" value="${redis.port4}"></constructor-arg>
          </bean>
          <bean class="org.springframework.data.redis.connection.redis-clusterNode">
            <constructor-arg name="host" value="${redis.host5}"></constructor-arg>
            <constructor-arg name="port" value="${redis.port5}"></constructor-arg>
          </bean>
          <bean class="org.springframework.data.redis.connection.redis-clusterNode">
            <constructor-arg name="host" value="${redis.host6}"></constructor-arg>
            <constructor-arg name="port" value="${redis.port6}"></constructor-arg>
          </bean>
       </set>
     </property>
</bean>
<bean id="jedisPoolConfig"   class="redis.clients.jedis.JedisPoolConfig">
       <property name="maxIdle" value="${redis.maxIdle}" />
       <property name="maxTotal" value="${redis.maxTotal}" />
</bean>
<bean id="jeidsConnectionFactory"
class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"  >
        <constructor-arg ref="redis-clusterConfiguration" />
        <constructor-arg ref="jedisPoolConfig" />

</bean>
```

```
<bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate">
        <property name="connectionFactory" ref="jeidsConnectionFactory" />
</bean>
</beans>
```

添加属性文件redis-cluster-config.properties

```
#cluster configuration
redis.host1=192.168.25.140
redis.port1=7001

redis.host2=192.168.25.140
redis.port2=7002

redis.host3=192.168.25.140
redis.port3=7003

redis.host4=192.168.25.140
redis.port4=7004

redis.host5=192.168.25.140
redis.port5=7005

redis.host6=192.168.25.140
redis.port6=7006

redis.maxRedirects=3
redis.maxIdle=100
redis.maxTotal=600
```

4.模拟集群异常测试

关闭节点命令

```
./redis-cli -p 端口 shutdown
```

（1）测试关闭7001 和7004, 看看会发生什么。

（2）测试关闭7001、7002、7003 会发生什么。