课后作业：案例——找出潜在好友

2018年8月1日 14:37

案例文件：

tom rose

tom jim

tom smith

tom lucy

rose tom

rose lucy

rose smith

jim tom

jim lucy

smith jim

smith tom

smith rose

比如tom认识rose，但是rose不认识jim，则那么rose-jim就是一对潜在好友，但tom-rose早就认识了，因此不算为潜在好友。

最后的结果为：

jim-rose

lucy-smith

FriendMapper代码：

```
public class FriendMapper extends Mapper<LongWritable, Text, Text,Text>{


    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
    Text>.Context context)
            throws IOException, InterruptedException {
        String line=value.toString();
        String name=line.split(" ")[0];
        String friend=line.split(" ")[1];
```

```
            context.write(new Text(name), new Text(friend));
        }

    }
```

FriendReducer代码：

```
public class FriendReducer extends Reducer<Text,Text,Text,IntWritable>{

    @Override
    protected void reduce(Text key, Iterable<Text> values,
            Reducer<Text, Text, Text, IntWritable>.Context context)
            throws IOException, InterruptedException {

        ArrayList<String> friendsList=new ArrayList<>();

        for(Text value:values){
            friendsList.add(value.toString());
            if(key.toString().compareTo(value.toString())<0){
                context.write(new Text(key+"-"+value),new IntWritable(1));
            }else{
                context.write(new Text(value+"-"+key),new IntWritable(1));
            }

        }

        for(int i=0;i<friendsList.size();i++){
            for(int j=0;j<friendsList.size();j++){
                String f1=friendsList.get(i);
                String f2=friendsList.get(j);
                if(f1.compareTo(f2)<0){
                    context.write(new Text(f1+"-"+f2),new IntWritable(2));
                }
            }
        }
```

```
    }
}
```

SecFriendMapper代码：

```
public class SecFriendMapper extends Mapper<LongWritable,Text,Text,IntWritable>{

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
    IntWritable>.Context context)
            throws IOException, InterruptedException {
        String line=value.toString();

        String friendInfo=line.split("\t")[0];
        int deep=Integer.parseInt(line.split("\t")[1]);

        context.write(new Text(friendInfo), new IntWritable(deep));
    }
}
```

SecFriendReducer代码：

```
public class SecFriendReducer extends Reducer<Text,IntWritable,Text,NullWritable>{

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,
            Reducer<Text, IntWritable, Text, NullWritable>.Context context) throws
            IOException, InterruptedException {

        Boolean flag=true;

        for(IntWritable value:values){
            if(value.get()==1){
                flag=false;
                break;
            }
        }
        if(flag){
            context.write(key, NullWritable.get());
        }
```

```
        }
    }
```

Driver代码：

```
public class Driver {

    public static void main(String[] args) throws Exception {
        Configuration conf=new Configuration();
        Job job=Job.getInstance(conf);

        job.setJarByClass(Driver.class);
        job.setMapperClass(FriendMapper.class);
        job.setReducerClass(FriendReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.setInputPaths(job, new Path("hdfs://192.168.150.137:9000/friend"));
        FileOutputFormat.setOutputPath(job, new
        Path("hdfs://192.168.150.137:9000/friend/result"));

        if(job.waitForCompletion(true)){
            Job job2=Job.getInstance(conf);
            job2.setMapperClass(SecFriendMapper.class);
            job2.setReducerClass(SecFriendReducer.class);

            job2.setMapOutputKeyClass(Text.class);
            job2.setMapOutputValueClass(IntWritable.class);

            job2.setOutputKeyClass(Text.class);
            job2.setOutputValueClass(NullWritable.class);

            FileInputFormat.setInputPaths(job2, new
            Path("hdfs://192.168.150.137:9000/friend/result"));
            FileOutputFormat.setOutputPath(job2, new
```

```
                Path("hdfs://192.168.150.137:9000/friend/secresult"));

                job2.waitForCompletion(true);

            }


        }
    }
```

Zebra案例

HttpAppHost代码:

```
public class HttpAppHost implements Writable{



    private String reportTime;

    private String cellid;

    private int appType;

    private int appSubType;

    private String userIp;

    private int userPort;

    private String appServerIp;

    private int appServerPort;

    private String host;

    private int attempts;

    private int accepts;

    private long trafficUL ;

    private long trafficDL;

    private long retranUL;

    private long retranDL;

    private long transDelay;
```

```
@Override
public void write(DataOutput out) throws IOException {
	out.writeUTF(reportTime);
	out.writeUTF(cellid);
	out.writeInt(appType);
	out.writeInt(appSubType);
	out.writeUTF(userIp);
	out.writeInt(userPort);
	out.writeUTF(appServerIp);
	out.writeInt(appServerPort);
	out.writeUTF(host);
	out.writeInt(attempts);
	out.writeInt(accepts);
	out.writeLong(trafficUL);
	out.writeLong(trafficDL);
	out.writeLong(retranUL);
	out.writeLong(retranDL);
	out.writeLong(transDelay);

}

@Override
public void readFields(DataInput in) throws IOException {
	this.reportTime=in.readUTF();
	this.cellid=in.readUTF();
```

```
        this.appType=in.readInt();
        this.appSubType=in.readInt();
        this.userIp=in.readUTF();
        this.userPort=in.readInt();
        this.appServerIp=in.readUTF();
        this.appServerPort=in.readInt();
        this.host=in.readUTF();
        this.attempts=in.readInt();
        this.accepts=in.readInt();
        this.trafficUL=in.readLong();
        this.trafficDL=in.readLong();
        this.retranUL=in.readLong();
        this.retranDL=in.readLong();
        this.transDelay=in.readLong();

    }
```

Mapper代码：

```
public class HttpAppHostMapper extends Mapper<LongWritable,Text,Text,HttpAppHost>{

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
    Text, HttpAppHost>.Context context)
            throws IOException, InterruptedException {
        FileSplit split=(FileSplit) context.getInputSplit();
```

```
String filename=split.getPath().getName();


String line=value.toString();
String[] data=line.split("\\|");
HttpAppHost hah=new HttpAppHost();
//用户上网所在的小区id
hah.setCellid(data[16]);
//appType是应用大类，用数字来表示请求的应用类型是音乐、游戏等
hah.setAppType(Integer.parseInt(data[22]));
//appSubType应用小类，比如微信、qq等
hah.setAppSubType(Integer.parseInt(data[23]));
hah.setUserIp(data[26]);
hah.setUserPort(Integer.parseInt(data[28]));
hah.setAppServerIp(data[30]);
hah.setAppServerPort(Integer.parseInt(data[32]));
hah.setHost(data[58]);
hah.setReportTime(filename.split("_")[1]);
//到此，基础字段信息封装完毕


//appTypecode是请求的状态码，如果是103，代表请求成功
int appTypeCode=Integer.parseInt(data[18]);
//传输的状态码。
String transStatus=data[54];


if(hah.getCellid()==null||hah.getCellid().equals("")){
```

```
        //如果小区是null或空，补齐为9个0
        hah.setCellid("000000000");
        }
//如果请求成功的，尝试成功次数设置为1
if(appTypeCode==103){
hah.setAttempts(1);
}
//如果请求成功，并且传输状态码满足，就设置accept为1。
if(appTypeCode==103
&&"10,11,12,13,14,15,32,33,34,35,36,37,38,48,49,50,51,52,53,54,55,199,20
0,201,202, 203,204,205,206,302,304,306".contains(transStatus)){
        hah.setAccepts(1);
}else{
        hah.setAccepts(0);
}
//如果成功，设置上传流量。用户向互联网（某一个APP服务端）的流量
if(appTypeCode == 103){
        hah.setTrafficUL(Long.parseLong(data[33]));
        }
//设置下传流量，用户从互联网下载的流量
if(appTypeCode == 103){
        hah.setTrafficDL(Long.parseLong(data[34]));
        }
//设置重传上行流量，当重新发起请求服务，产生的流量
if(appTypeCode == 103){
```

```
            hah.setRetranUL(Long.parseLong(data[39]));
            }
        //重传下行流量
        if(appTypeCode == 103){
            hah.setRetranDL(Long.parseLong(data[40]));
            }
        //设置传输总用时
        if(appTypeCode==103){
            hah.setTransDelay(Long.parseLong(data[20]) -
            Long.parseLong(data[19]));
        }
        //结算字段，封装完毕


        //根据业务规则，判断是否同一个用户产生的数据
        String key2=hah.getCellid()+"|"+hah.getAppType()+"|"+
                hah.getAppSubType()+"|"+hah.getUserIp()+"|"+hah.getUserPort()
                +"|"+hah.getAppServerIp()+"|"+hah.getAppServerPort()+"|"+
                hah.getHost()+"|"+hah.getReportTime();



        context.write(new Text(key2),hah);
    }

}
```

Reduce代码：

```
public class HttpAppHostReducer extends
Reducer<Text,HttpAppHost,HttpAppHost,NullWritable>{

    @Override
    protected void reduce(Text key, Iterable<HttpAppHost> values,
            Reducer<Text, HttpAppHost, HttpAppHost, NullWritable>.Context
            context)
            throws IOException, InterruptedException {

        HttpAppHost oldHah=new HttpAppHost();
        String[] basicData=key.toString().split("\\|");
        //封装基础字段
        oldHah.setCellid(basicData[0]);
        oldHah.setAppType(Integer.parseInt(basicData[1]));
        oldHah.setAppSubType(Integer.parseInt(basicData[2]));
        oldHah.setUserIp(basicData[3]);
        oldHah.setUserPort(Integer.parseInt(basicData[4]));
        oldHah.setAppServerIp(basicData[5]);
        oldHah.setAppServerPort(Integer.parseInt(basicData[6]));
        oldHah.setHost(basicData[7]);
        oldHah.setReportTime(basicData[8]);

            for(HttpAppHost hah:values){
                //合并需要累加的字段
```

```
                oldHah.setAccepts(oldHah.getAccepts()+hah.getAccepts());
                oldHah.setAttempts(oldHah.getAttempts()+hah.getAttempts());
                oldHah.setTrafficDL(oldHah.getTrafficDL()+hah.getTrafficDL());
                oldHah.setTrafficUL(oldHah.getTrafficUL()+hah.getTrafficUL());
                oldHah.setRetranUL(oldHah.getRetranUL()+hah.getRetranUL());
                oldHah.setRetranDL(oldHah.getRetranDL()+hah.getRetranDL());
                oldHah.setTransDelay(oldHah.getTransDelay()+hah.getTransDelay(
                ));
            }


            context.write(oldHah, NullWritable.get());

    }

}
```

Driver代码：

```
public class HttpAppHostDriver {

    public static void main(String[] args) throws Exception {
        Configuration conf=new Configuration();
        Job job=Job.getInstance(conf);

        job.setJarByClass(HttpAppHostDriver.class);
        job.setMapperClass(HttpAppHostMapper.class);
```

```
        job.setReducerClass(HttpAppHostReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(HttpAppHost.class);

        job.setOutputKeyClass(HttpAppHost.class);
        job.setOutputValueClass(NullWritable.class);

        FileInputFormat.setInputPaths(job, new
        Path("hdfs://192.168.234.21:9000/zebra"));
        FileOutputFormat.setOutputPath(job, new
        Path("hdfs://192.168.234.21:9000/zebra/result"));

        job.waitForCompletion(true);


    }
}
```

Shuffle（洗牌）—需要整理

2018年8月29日　　15:28

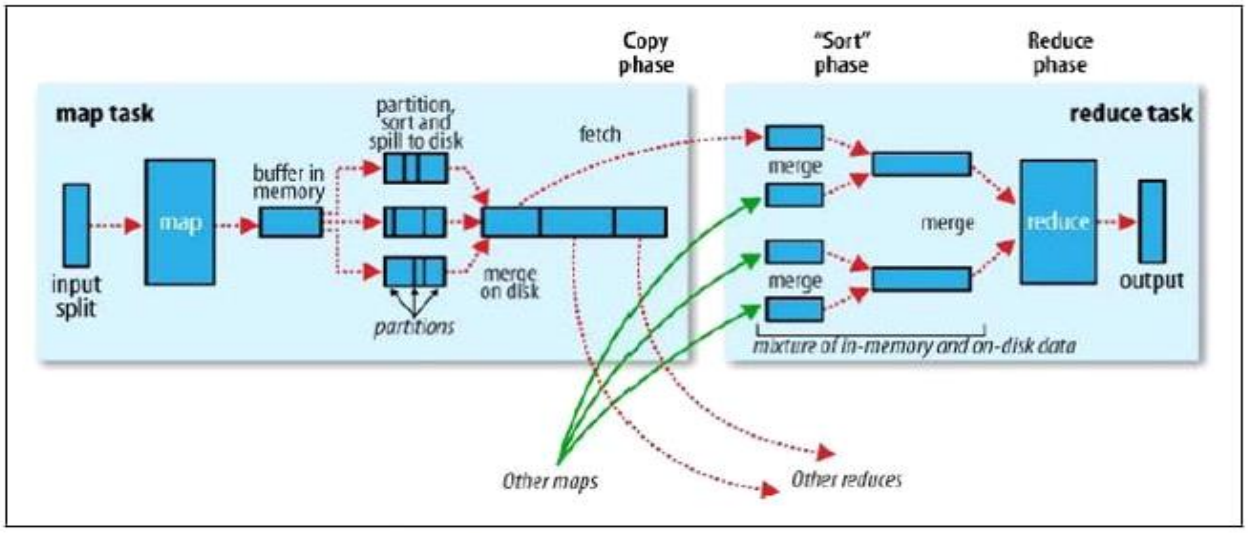


Figure 6-4. Shuffle and sort in MapReduce

案例——Combiner合并

合并的目的是减少Reduce端 迭代的次数
combiner是实现Mapper端进行key的归并，combiner具有类似本地的reduce功能。
如果不用combiner,那么所有的结果都是reduce完成，效率会很低。使用combiner先做合并，然后发往reduce。

案例一：计算利润，含合并

ProfitInfo代码：

```
public class ProfitInfo implements WritableComparable<ProfitInfo>{
	private String name;
	private long profit;

	public ProfitInfo() {
	}



	@Override
	public String toString() {
		return "ProfitInfo [name=" + name + ", profit=" + profit + "]";
	}



	public ProfitInfo(String name, long profit) {
		this.name = name;
		this.profit = profit;
	}



	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public long getProfit() {
		return profit;
	}
	public void setProfit(long profit) {
		this.profit = profit;
	}
	@Override
	public void readFields(DataInput in) throws IOException {
		this.name = in.readUTF();
		this.profit = in.readLong();
```

```
    }
    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(this.name);
        out.writeLong(this.profit);
    }

    @Override
    public int compareTo(ProfitInfo o) {
        return (int) (o.profit - this.profit);
    }
}
```

ProfitDriver代码：

```
public class ProfitDriver {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "profitJob");
        job.setJarByClass(cn.tedu.profit.ProfitDriver.class);
        job.setMapperClass(cn.tedu.profit.ProfitMapper.class);
        job.setCombinerClass(ProfitReducer.class);
        job.setReducerClass(cn.tedu.profit.ProfitReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(ProfitInfo.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(ProfitInfo.class);


        FileInputFormat.setInputPaths(job, new
        Path("hdfs://192.168.234.21:9000/profit"));
        FileOutputFormat.setOutputPath(job, new
        Path("hdfs://192.168.234.21:9000/profit/result"));

        if (!job.waitForCompletion(true))
            return;
    }

}
```

ProfitMapper代码：

```
public class ProfitMapper extends Mapper<LongWritable, Text, Text, ProfitInfo> {

    public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
        String line = value.toString();
        String [] attrs = line.split(" ");
        String name = attrs[1];
        long profit = Long.parseLong(attrs[2]) - Long.parseLong(attrs[3]);

        ProfitInfo pi = new ProfitInfo(name,profit);
        context.write(new Text(name), pi);
    }
```

```
}
```

ProfitReducer代码：

```
public class ProfitReducer extends Reducer<Text, ProfitInfo, Text, ProfitInfo> {

	public void reduce(Text key, Iterable<ProfitInfo> values, Context context)
	throws IOException, InterruptedException {

		ProfitInfo pi=new ProfitInfo();
		pi.setName(key.toString());
		int i=0;
		for(ProfitInfo p:values){
			System.out.println(i++);
			pi.setProfit(pi.getProfit()+p.getProfit());
		}

		context.write(key,pi);
	}

}
```

Hadoop2.0高可用集群搭建方案

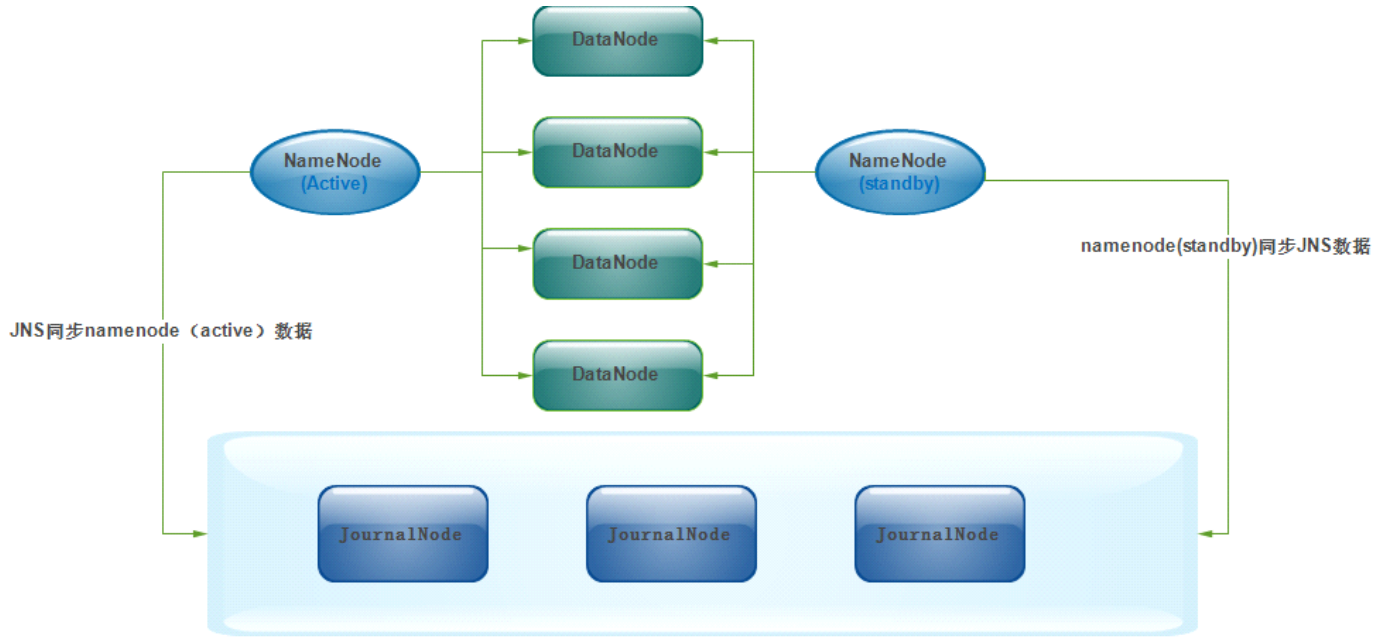
Hadoop1.0版本的单点问题

Hadoop 的namenode好比人的心脏，非常重要，绝对不可以停止工作，在hadoop1,只有一个namenode,如果该namenode数据丢失或停止工作，整个集群就不能恢复了。

hadoop2比hadoop1改进的地方：①高可用的解决方案

hadoop2中，有两个namenode(目前只支持两个,注意，不是secondarynamenode）。每一个都具有相同的职能，一个是active状态，一个是standby待命状态，时刻同步active状态的namenode数据，实现高可用。

要想实现数据的实现共享，新HDFS采用了一种共享机制，Quorum Journal Node（JournalNode）集群。NFS是操作系统层面的，JournalNode是hadoop层面的（主流做法）

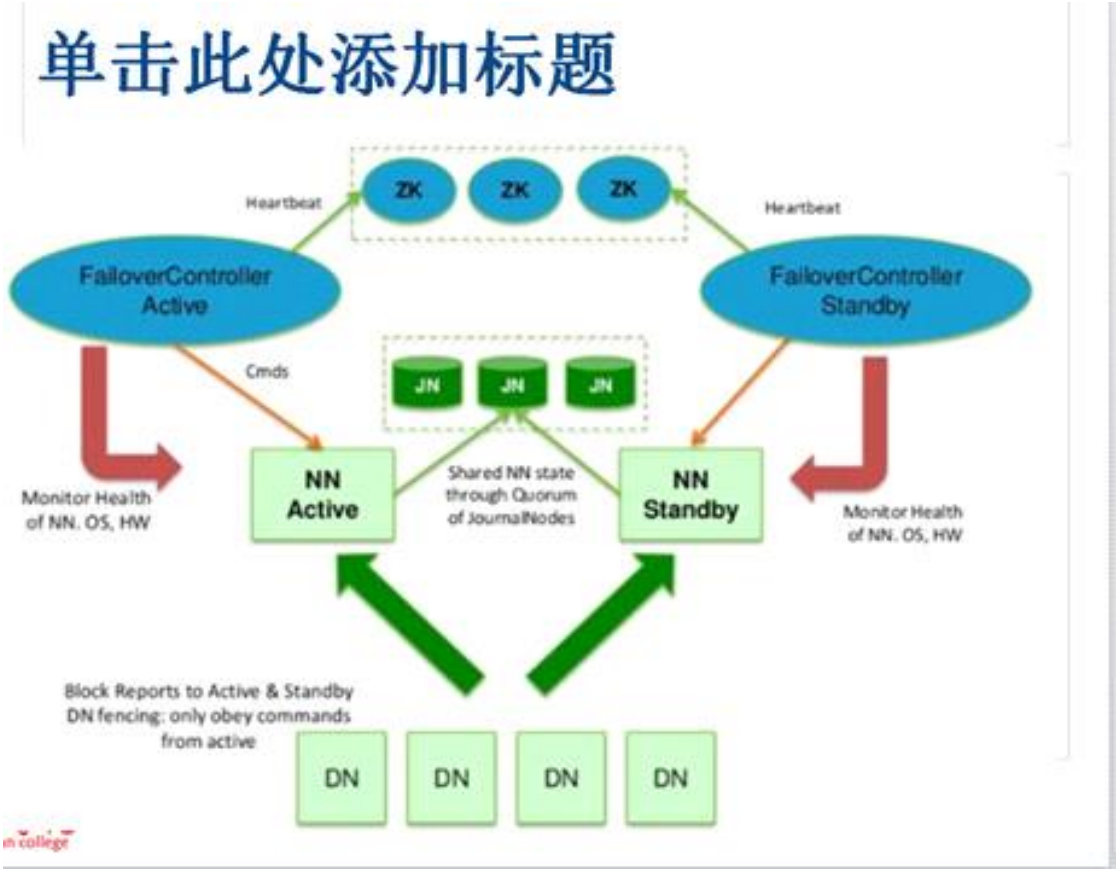


Hadoop2.0实现思想

两个NameNode为了数据同步，会通过一组称作JournalNodes的独立进程进行相互通信。当active状态的NameNode的命名空间有任何修改时，会告知大部分的JournalNodes进程。standby状态的NameNode有能力读取JNs中的变更信息，并且一直监控edit log的变化，把变化应用于自己的命名空间。standby可以确保在集群出错时，命名空间状态已经完全同步了

1.4 NameNode之间的故障切换

对于HA集群而言，确保同一时刻只有一个NameNode处于active状态是至关重要的。否则，两个NameNode的数据状态就会产生分歧，可能丢失数据，或者产生错误的结果。为了保证这点，这就需要利用使用ZooKeeper了。首先HDFS集群中的两个NameNode都在ZooKeeper中注册，当active状态的NameNode出故障时，ZooKeeper能检测到这种情况，它就会自动把standby状态的NameNode切换为active状态。



Hadoop2.0 HA集群搭建步骤

集群节点分配

- [ ] Park01
  - Zookeeper
  - NameNode (active)
  - Resourcemanager (active)

- [ ] Park02
  - Zookeeper
  - NameNode (standby)

- [ ] Park03
  - Zookeeper
  - ResourceManager (standby)

- [ ] Park04
  - DataNode
  - NodeManager
  - JournalNode

- [ ] Park05
  - DataNode
  - NodeManager
  - JournalNode

- [ ] Park06
  - DataNode
  - NodeManager
  - JournalNode

安装步骤

- [ ] 0.永久关闭每台机器的防火墙

  执行：service iptables stop

  再次执行：chkconfig iptables off

- [ ] 1.为每台机器配置主机名以及hosts文件

  配置主机名=》执行： vim /etc/sysconfig/network =》然后执行 hostname 主机名=》达到不重启生效目的

  配置hosts文件=》执行：vim /etc/hosts

📖 **示例：**

```
127.0.0.1   localhost
::1         localhost
192.168.234.21 hadoop01
192.168.234.22 hadoop02
192.168.234.23 hadoop03
192.168.234.24 hadoop04
192.168.234.25 hadoop05
192.168.234.26 hadoop06
```

- [ ] 2.通过远程命令将配置好的hosts文件 scp到其他5台节点上

  执行：scp /etc/hosts hadoop02: /etc

- [ ] 3.为每台机器配置ssh免秘钥登录

  执行：ssh-keygen

  ssh-copy-id root@hadoop01 （分别发送到6台节点上）

- [ ] 4.前三台机器安装和配置zookeeper

  配置conf目录下的zoo.cfg以及创建myid文件

  （zookeeper集群安装具体略）

- [ ] 5.为每台机器安装jdk和配置jdk环境

- [ ] 6.为每台机器配置主机名,**然后每台机器重启，（如果不重启，也可以配合：hostname hadoop01生效）**

  执行： vim /etc/sysconfig/network 进行编辑

- [ ] 7.安装和配置01节点的hadoop

- [ ] 配置hadoop-env.sh

  配置jdk安装所在目录

  配置hadoop配置文件所在目录

```
# The java implementation to use.
export JAVA_HOME=/home/software/jdk1.8

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol.  Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}
```

- [ ] 8.配置core-site.xml

```
<configuration>
<!--用来指定hdfs的老大，ns为固定属性名，表示两个namenode-->
<property>
<name>fs.defaultFS</name>
<value>hdfs://ns</value>
</property>
<!--用来指定hadoop运行时产生文件的存放目录-->
<property>
<name>hadoop.tmp.dir</name>
<value>/home/software/hadoop-2.7.1/tmp</value>
</property>
<!--执行zookeeper地址-->
<property>
<name>ha.zookeeper.quorum</name>
<value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>
```

```
</property>
</configuration>
```

9.配置01节点的hdfs-site.xml

配置

```
<configuration>
<!--执行hdfs的nameservice为ns,和core-site.xml保持一致-->
<property>
<name>dfs.nameservices</name>
<value>ns</value>
</property>
<!--ns下有两个namenode,分别是nn1,nn2-->
<property>
<name>dfs.ha.namenodes.ns</name>
<value>nn1,nn2</value>
</property>
<!--nn1的RPC通信地址-->
<property>
<name>dfs.namenode.rpc-address.ns.nn1</name>
<value>hadoop01:9000</value>
</property>
<!--nn1的http通信地址-->
<property>
<name>dfs.namenode.http-address.ns.nn1</name>
<value>hadoop01:50070</value>
</property>
<!--nn2的RPC通信地址-->
<property>
<name>dfs.namenode.rpc-address.ns.nn2</name>
<value>hadoop02:9000</value>
</property>
<!--nn2的http通信地址-->
<property>
<name>dfs.namenode.http-address.ns.nn2</name>
<value>hadoop02:50070</value>
</property>
<!--指定namenode的元数据在JournalNode上的存放位置,这样，namenode2可以从jn集群里获
取
    最新的namenode的信息，达到热备的效果-->
<property>
<name>dfs.namenode.shared.edits.dir</name>
<value>qjournal://hadoop04:8485;hadoop05:8485;hadoop06:8485/ns</value>
</property>
<!--指定JournalNode存放数据的位置-->
<property>
<name>dfs.journalnode.edits.dir</name>
<value>/home/software/hadoop-2.7.1/journal</value>
</property>
<!--开启namenode故障时自动切换-->
<property>
<name>dfs.ha.automatic-failover.enabled</name>
<value>true</value>
</property>
<!--配置切换的实现方式-->
```

```
<property>
<name>dfs.client.failover.proxy.provider.ns</name>
<value>
org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
</property>
<!--配置隔离机制-->
<property>
<name>dfs.ha.fencing.methods</name>
<value>sshfence</value>
</property>
<!--配置隔离机制的ssh登录秘钥所在的位置-->
<property>
<name>dfs.ha.fencing.ssh.private-key-files</name>
<value>/root/.ssh/id_rsa</value>
</property>

<!--配置namenode数据存放的位置,可以不配置，如果不配置，默认用的是
     core-site.xml里配置的hadoop.tmp.dir的路径-->
<property>
<name>dfs.namenode.name.dir</name>
<value>file:///home/software/hadoop-2.7.1/tmp/namenode</value>
</property>
<!--配置datanode数据存放的位置,可以不配置，如果不配置，默认用的是
          core-site.xml里配置的hadoop.tmp.dir的路径-->
<property>
<name>dfs.datanode.data.dir</name>
<value>file:///home/software/hadoop-2.7.1/tmp/datanode</value>
</property>

<!--配置block副本数量-->
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
<!--设置hdfs的操作权限，false表示任何用户都可以在hdfs上操作文件-->
<property>
<name>dfs.permissions</name>
<value>false</value>
</property>

</configuration>
```

- [ ] 10.配置mapred-site.xml

📖 配置代码：

```
<configuration>
<property>
<!--指定mapreduce运行在yarn上-->
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

- [ ] 11.配置yarn-site.xml

📖 配置代码：

```
<configuration>
```

```
 <!-- 开启YARN HA -->
<property>
<name>yarn.resourcemanager.ha.enabled</name>
<value>true</value>
</property>

  <!-- 指定两个resourcemanager的名称 -->
<property>
<name>yarn.resourcemanager.ha.rm-ids</name>
<value>rm1,rm2</value>
</property>

<!-- 配置rm1，rm2的主机 -->
<property>
<name>yarn.resourcemanager.hostname.rm1</name>
<value>hadoop01</value>
</property>

<property>
<name>yarn.resourcemanager.hostname.rm2</name>
<value>hadoop03</value>
</property>

<!--开启yarn恢复机制-->
<property>
<name>yarn.resourcemanager.recovery.enabled</name>
<value>true</value>
</property>

<!--执行rm恢复机制实现类-->
<property>
<name>yarn.resourcemanager.store.class</name>
<value>
org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>
</property>

<!-- 配置zookeeper的地址 -->
<property>
<name>yarn.resourcemanager.zk-address</name>
<value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>
<description>For multiple zk services, separate them with comma</description>
</property>

<!-- 指定YARN HA的名称 -->
<property>
<name>yarn.resourcemanager.cluster-id</name>
<value>yarn-ha</value>
</property>

<property>
<!--指定yarn的老大 resoucemanager的地址-->
<name>yarn.resourcemanager.hostname</name>
<value>hadoop03</value>
</property>
<property>
<!--NodeManager获取数据的方式-->
```

```
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
</configuration>
```

- [ ] 12.配置slaves文件

📖 配置代码：

```
hadoop04
hadoop05
hadoop06
```

- [ ] 13.配置hadoop的环境变量（可不配）

```
JAVA_HOME=/home/software/jdk1.8
HADOOP_HOME=/home/software/hadoop-2.7.1
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
export JAVA_HOME PATH CLASSPATH HADOOP_HOME
```

- [ ] 14.根据配置文件，创建相关的文件夹，用来存放对应数据

在hadoop-2.7.1目录下创建:

①journal目录

②创建tmp目录

③在tmp目录下，分别创建namenode目录和datanode目录

- [ ] 15.通过scp 命令，将hadoop安装目录远程copy到其他5台机器上

比如向hadoop02节点传输：

scp -r hadoop-2.7.1 hadoop02:/home/software

Hadoop集群启动

- [ ] 16.启动zookeeper集群

在Zookeeper安装目录的bin目录下执行：sh zkServer.sh start

- [ ] 17.格式化zookeeper

在zk的leader节点上执行：

hdfs zkfc -formatZK，这个指令的作用是在zookeeper集群上生成hadoop-ha节点（ns节点）

```
16/10/10 20:29:35 INFO zookeeper.ClientCnxn: Session establishment complete on server hadoop02/192.168
.234.22:2181, sessionid = 0x257b18226890000, negotiated timeout = 5000
16/10/10 20:29:35 INFO ha.ActiveStandbyElector: Session connected.
16/10/10 20:29:35 INFO ha.ActiveStandbyElector: Successfully created /hadoop-ha/ns in ZK.
```

注：18--24步可以用一步来替代：进入hadoop安装目录的sbin目录，执行：start-dfs.sh 。但建议还是按部就班来执行，比较可靠。

- [ ] 18.启动journalnode集群

在01、02、03节点上执行：

切换到hadoop安装目录的bin目录下，执行：

```
sh hadoop-daemons.sh start journalnode
```

然后执行jps命令查看：

```
[root@hadoop06 ~]# jps
2375 JournalNode
2429 Jps
```

- [ ] 19.格式化01节点的namenode

在01节点上执行：

hadoop namenode -format

- [ ] 20.启动01节点的namenode

在01节点上执行：

hadoop-daemon.sh start namenode

```
[root@hadoop01 hadoop-2.7.1]# jps
6465 Jps
3649 QuorumPeerMain
6388 NameNode
```

- [ ] 21.把02节点的 namenode节点变为standby namenode节点

在02节点上执行：

```
hdfs namenode -bootstrapStandby
```

```
16/10/10 20:50:14 INFO common.Storage: Storage directory /home/software/hadoop-2.7.1/tmp/namenode has
been successfully formatted.
16/10/10 20:50:15 INFO namenode.TransferFsImage: Opening connection to http://hadoop01:50070/imagetran
sfer?getimage=1&txid=0&storageInfo=-63:1223489324:0:CID-273d2cc0-2e0c-4082-a909-a4eb1e7f078f
16/10/10 20:50:15 INFO namenode.TransferFsImage: Image Transfer timeout configured to 60000 millisecon
ds
16/10/10 20:50:15 INFO namenode.TransferFsImage: Transfer took 0.00s at 0.00 KB/s
16/10/10 20:50:15 INFO namenode.TransferFsImage: Downloaded file fsimage.ckpt_0000000000000000000 size
 351 bytes.
```

- [ ] 22.启动02节点的namenode节点

在02节点上执行：

```
hadoop-daemon.sh start namenode
```

- [ ] 23.在01,02,03节点上启动datanode节点

在01,02,03节点上执行：`hadoop-daemon.sh start datanode`

- [ ] 24.启动zkfc（启动FalioverControllerActive)

在01,02节点上执行：

hadoop-daemon.sh start zkfc

```
[root@hadoop01 hadoop-2.7.1]# jps
3649 QuorumPeerMain
6388 NameNode
6790 DFSZKFailoverController
6856 Jps
```

- [ ] 25.在01节点上启动 主Resourcemanager

在01节点上执行：start-yarn.sh

```
[root@hadoop03 ~]# start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/software/hadoop-2.7.1/logs/yarn-root-resourcemanager-hadoop
03.out
hadoop06: starting nodemanager, logging to /home/software/hadoop-2.7.1/logs/yarn-root-nodemanager-hado
op06.out
hadoop04: starting nodemanager, logging to /home/software/hadoop-2.7.1/logs/yarn-root-nodemanager-hado
op04.out
hadoop05: starting nodemanager, logging to /home/software/hadoop-2.7.1/logs/yarn-root-nodemanager-hado
op05.out
[root@hadoop03 ~]# jps
2533 ResourceManager
2602 Jps
```

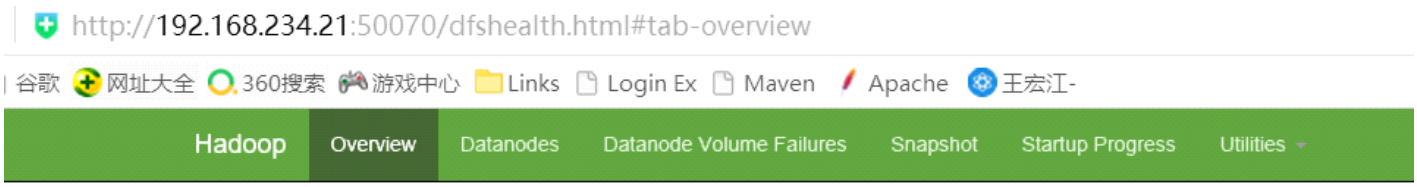
启动成功后，04,05,06节点上应该有nodemanager 的进程

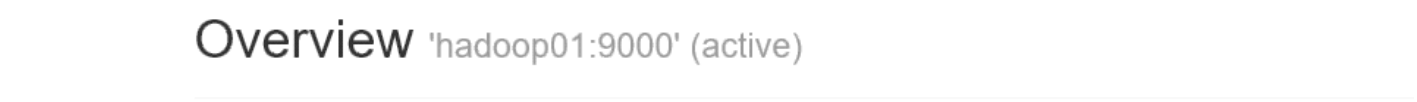
- [ ] 26.在03节点上启动副 Resoucemanager

在03节点上执行：`yarn-daemon.sh start resourcemanager`

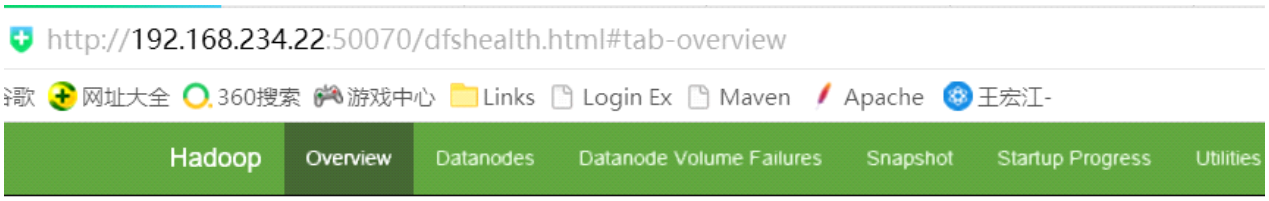
- [ ] 27.测试

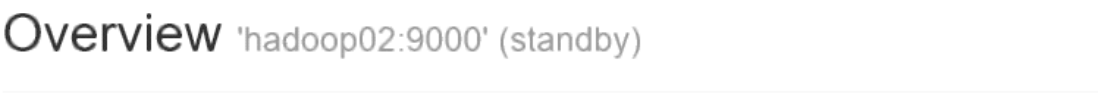
输入地址：http://192.168.234.21:50070，查看namenode的信息，是active状态的





输入地址：http://192.168.234.22:50070，查看namenode的信息，是standby状态





然后停掉01节点的namenode,此时返现standby的namenode变为active。

- [ ] 28.查看yarn的管理地址

http://192.168.234.21:8088（节点01的8088端口）