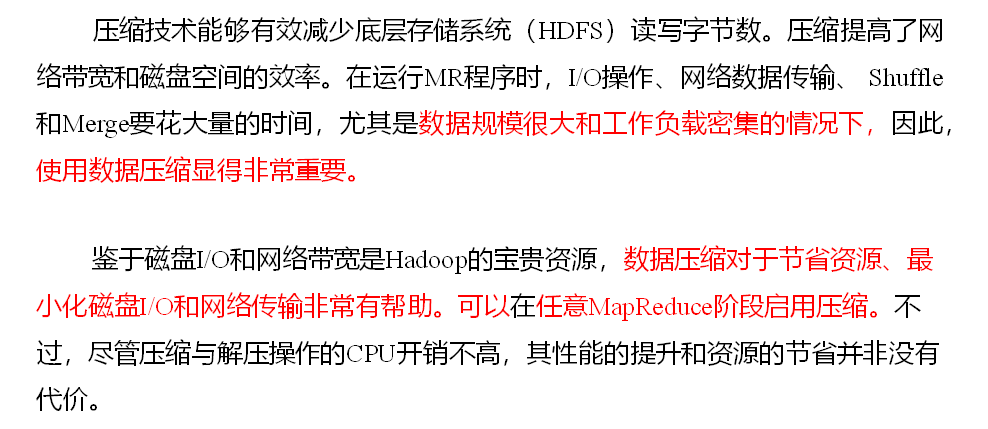
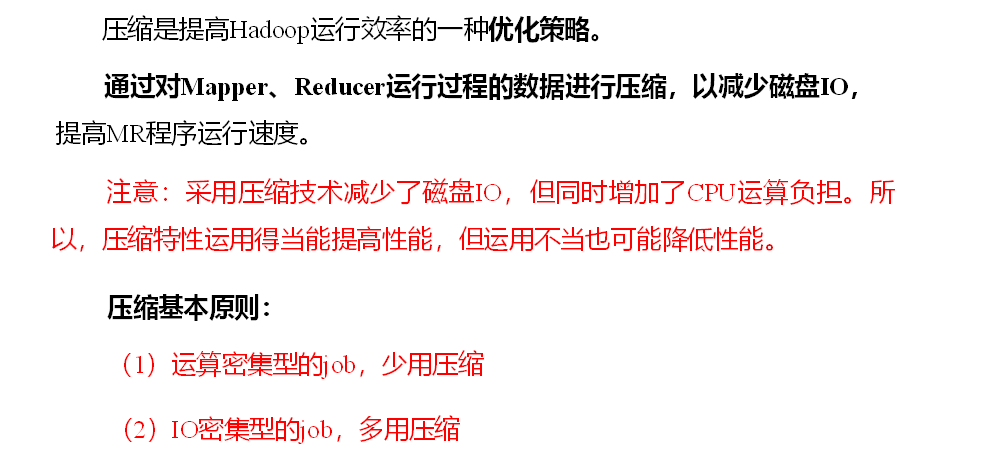
# 第1章 Hadoop数据压缩

## 1.1 概述





## 1.2 MR支持的压缩编码

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 压缩格式 | hadoop自带？ | 算法 | 文件扩展名 | 是否可切片 | 换成压缩格式后，原来的程序是否需要修改 |
| DEFLATE | 是，直接使用 | DEFLATE | .deflate | 否 | 和文本处理一样，不需要修改 |
| Gzip | 是，直接使用 | DEFLATE | .gz | 否 | 和文本处理一样，不需要修改 |
| bzip2 | 是，直接使用 | bzip2 | .bz2 | 是 | 和文本处理一样，不需要修改 |
| LZO | 否，需要安装 | LZO | .lzo | 是 | 需要建索引，还需要指定输入格式 |
| Snappy | 是，直接使用 | Snappy | .snappy | 否 | 和文本处理一样，不需要修改 |

为了支持多种压缩/解压缩算法，Hadoop引入了编码/解码器，如下表所示。

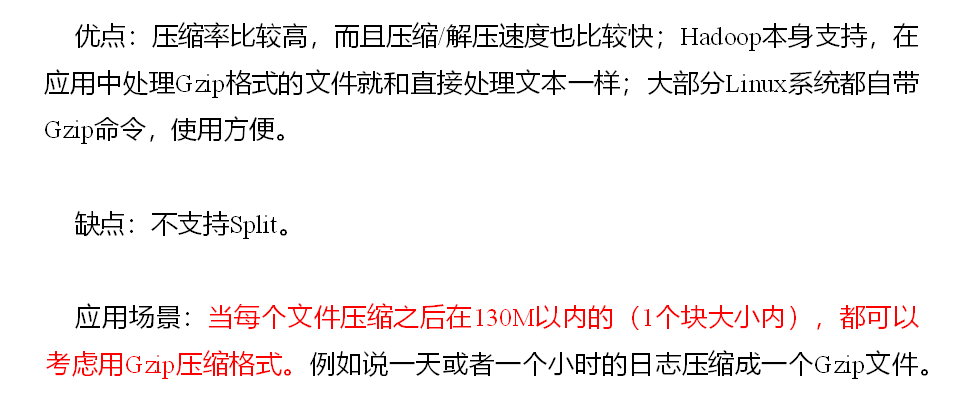
|  |  |
| --- | --- |
| 压缩格式 | 对应的编码/解码器 |
| DEFLATE | org.apache.hadoop.io.compress.DefaultCodec |
| gzip | org.apache.hadoop.io.compress.GzipCodec |
| bzip2 | org.apache.hadoop.io.compress.BZip2Codec |
| LZO | com.hadoop.compression.lzo.LzopCodec |
| Snappy | org.apache.hadoop.io.compress.SnappyCodec |

压缩性能的比较

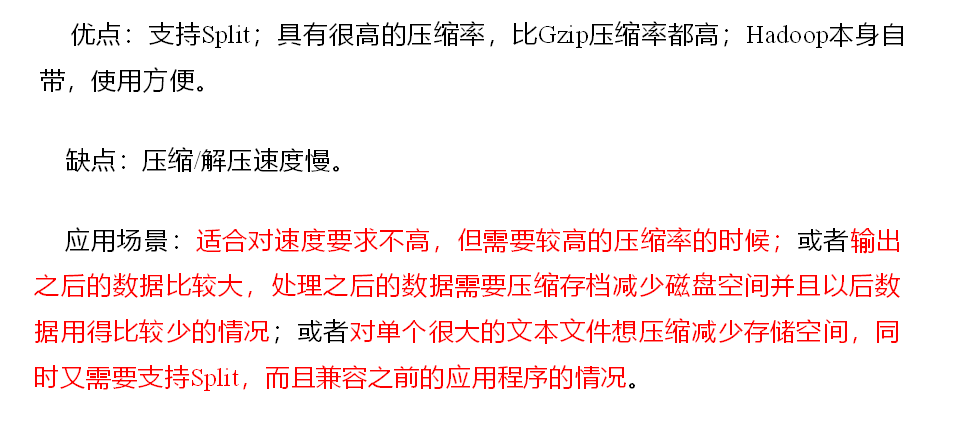
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 压缩算法 | 原始文件大小 | 压缩文件大小 | 压缩速度 | 解压速度 |
| gzip | 8.3GB | 1.8GB | 17.5MB/s | 58MB/s |
| bzip2 | 8.3GB | 1.1GB | 2.4MB/s | 9.5MB/s |
| LZO | 8.3GB | 2.9GB | 49.3MB/s | 74.6MB/s |

## 1.3 压缩方式选择

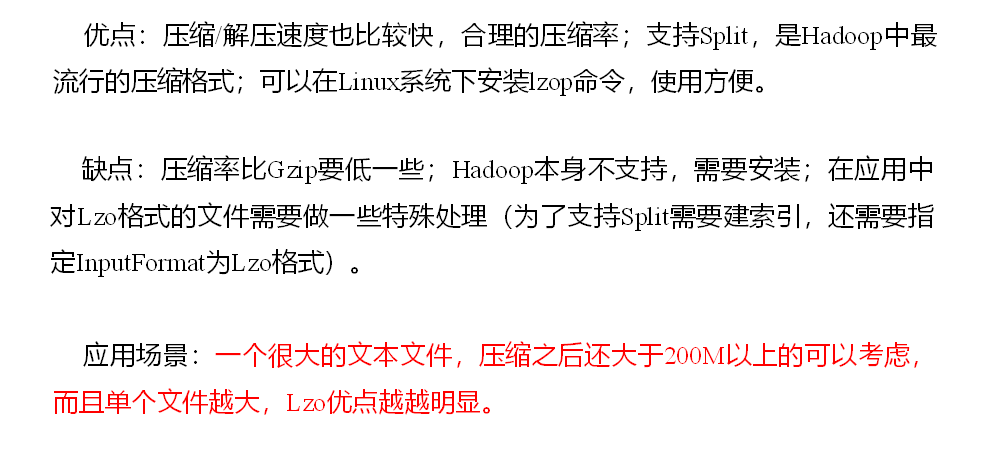
### 1.3.1 Gzip压缩



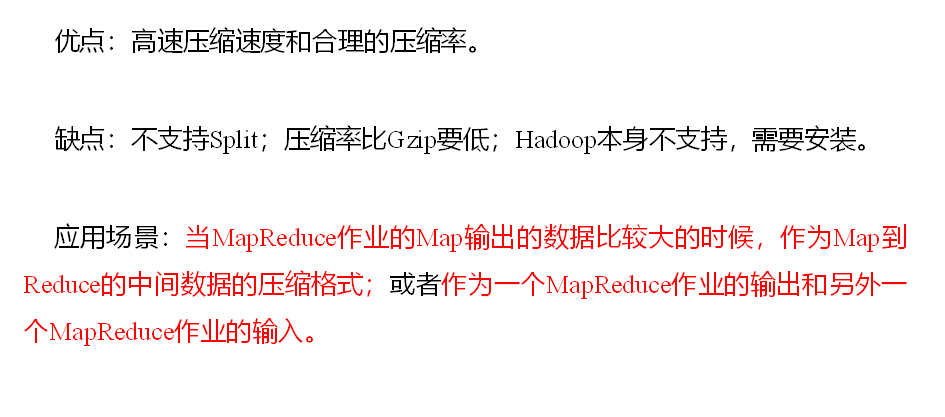
### 1.3.2 Bzip2压缩



### 1.3.3 Lzo压缩



### 1.3.4 Snappy压缩



## 1.4 压缩位置选择

压缩可以在MapReduce作用的任意阶段启用。

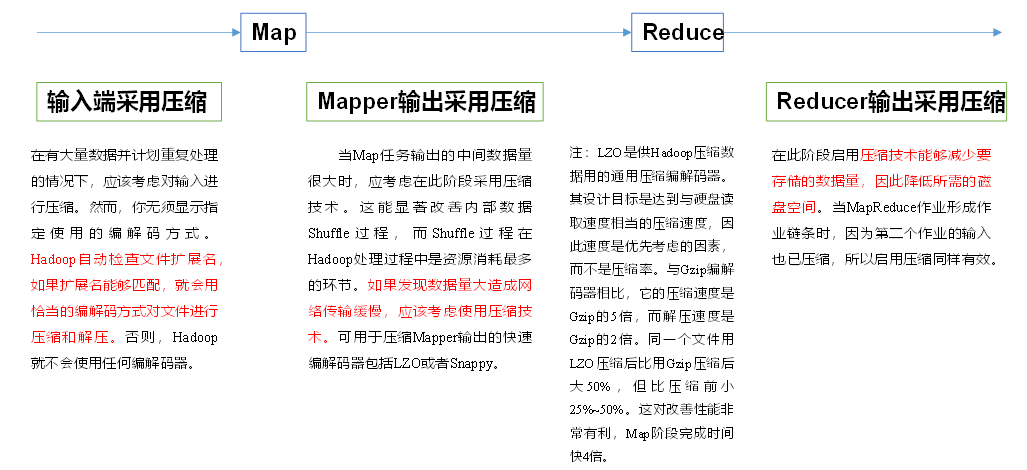


图 MapReduce数据压缩

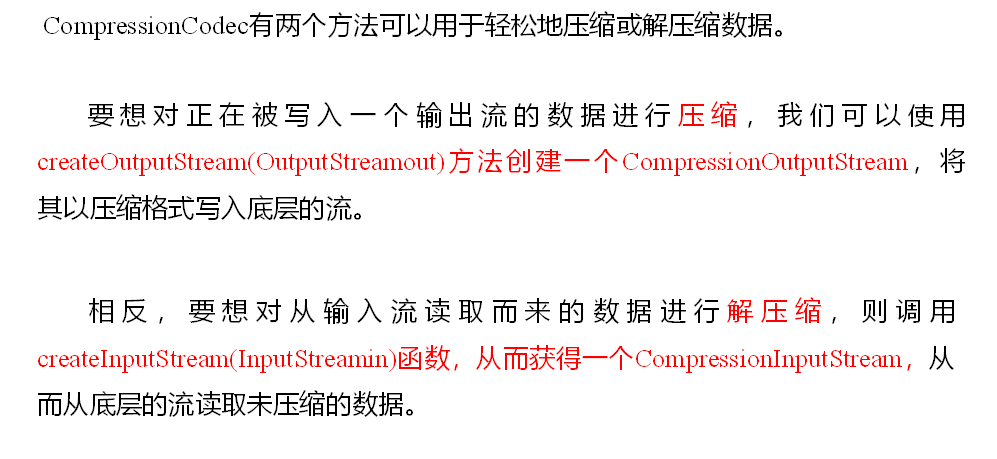
## 1.5 压缩参数配置

要在Hadoop中启用压缩，可以配置如下参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 默认值 | 阶段 | 建议 |
| io.compression.codecs  （在core-site.xml中配置） | 无，这个需要在命令行输入hadoop checknative查看 | 输入压缩 | Hadoop使用文件扩展名判断是否支持某种编解码器 |
| mapreduce.map.output.compress（在mapred-site.xml中配置） | false | mapper输出 | 这个参数设为true启用压缩 |
| mapreduce.map.output.compress.codec（在mapred-site.xml中配置） | org.apache.hadoop.io.compress.DefaultCodec | mapper输出 | 企业多使用LZO或Snappy编解码器在此阶段压缩数据 |
| mapreduce.output.fileoutputformat.compress（在mapred-site.xml中配置） | false | reducer输出 | 这个参数设为true启用压缩 |
| mapreduce.output.fileoutputformat.compress.codec（在mapred-site.xml中配置） | org.apache.hadoop.io.compress.DefaultCodec | reducer输出 | 使用标准工具或者编解码器，如gzip和bzip2 |
| mapreduce.output.fileoutputformat.compress.type（在mapred-site.xml中配置） | RECORD | reducer输出 | SequenceFile输出使用的压缩类型：NONE和BLOCK |

## 1.6 压缩实操案例

### 1.6.1 数据流的压缩和解压缩

测试一下如下压缩方式：

|  |  |
| --- | --- |
| DEFLATE | org.apache.hadoop.io.compress.DefaultCodec |
| gzip | org.apache.hadoop.io.compress.GzipCodec |
| bzip2 | org.apache.hadoop.io.compress.BZip2Codec |

package com.user1.mapreduce.compress;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IOUtils;

import org.apache.hadoop.io.compress.CompressionCodec;

import org.apache.hadoop.io.compress.CompressionCodecFactory;

import org.apache.hadoop.io.compress.CompressionInputStream;

import org.apache.hadoop.io.compress.CompressionOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class TestCompress {

public static void main(String[] args) throws IOException {

compress("D:\\input\\inputcompression\\JaneEyre.txt"

,"org.apache.hadoop.io.compress.BZip2Codec");

//decompress("D:\\input\\inputcompression\\JaneEyre.txt.bz2");

}

//压缩

private static void compress(String filename, String method) throws IOException {

//1 获取输入流

FileInputStream fis = new FileInputStream(new File(filename));

//2 获取输出流

//获取压缩编解码器codec

CompressionCodecFactory factory = new CompressionCodecFactory(new Configuration());

CompressionCodec codec = factory.getCodecByName(method);

//获取普通输出流,文件后面需要加上压缩后缀

FileOutputStream fos = new FileOutputStream(new File(filename + codec.getDefaultExtension()));

//获取压缩输出流,用压缩解码器对fos进行压缩

CompressionOutputStream cos = codec.createOutputStream(fos);

//3 流的对拷

IOUtils.copyBytes(fis,cos,new Configuration());

//4 关闭资源

IOUtils.closeStream(cos);

IOUtils.closeStream(fos);

IOUtils.closeStream(fis);

}

//解压缩

private static void decompress(String filename) throws IOException {

//0 校验是否能解压缩

CompressionCodecFactory factory = new CompressionCodecFactory(new Configuration());

CompressionCodec codec = factory.getCodec(new Path(filename));

if (codec == null) {

System.out.println("cannot find codec for file " + filename);

return;

}

//1 获取输入流

FileInputStream fis = new FileInputStream(new File(filename));

CompressionInputStream cis = codec.createInputStream(fis);

//2 获取输出流

FileOutputStream fos = new FileOutputStream(new File(filename + ".decodec"));

//3 流的对拷

IOUtils.copyBytes(cis,fos,new Configuration());

//4 关闭资源

IOUtils.closeStream(fos);

IOUtils.closeStream(cis);

IOUtils.closeStream(fis);

}

}

### 1.6.2 Map输出端采用压缩

即使你的MapReduce的输入输出文件都是未压缩的文件，你仍然可以对Map任务的中间结果输出做压缩，因为它要写在硬盘并且通过网络传输到Reduce节点，对其压缩可以提高很多性能，这些工作只要设置两个属性即可，我们来看下代码怎么设置。

1）给大家提供的Hadoop源码支持的压缩格式有：BZip2Codec 、DefaultCodec

package com.user1.mapreduce.compress;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.compress.BZip2Codec;

import org.apache.hadoop.io.compress.CompressionCodec;

import org.apache.hadoop.io.compress.GzipCodec;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountDriver {

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

// 开启map端输出压缩

conf.setBoolean("mapreduce.map.output.compress", true);

// 设置map端输出压缩方式

conf.setClass("mapreduce.map.output.compress.codec", BZip2Codec.class,CompressionCodec.class);

Job job = Job.getInstance(conf);

job.setJarByClass(WordCountDriver.class);

job.setMapperClass(WordCountMapper.class);

job.setReducerClass(WordCountReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.setInputPaths(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

boolean result = job.waitForCompletion(true);

System.exit(result ? 0 : 1);

}

}

2）Mapper保持不变

package com.user1.mapreduce.compress;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{

Text k = new Text();

IntWritable v = new IntWritable(1);

@Override

protected void map(LongWritable key, Text value, Context context)throws IOException, InterruptedException {

// 1 获取一行

String line = value.toString();

// 2 切割

String[] words = line.split(" ");

// 3 循环写出

for(String word:words){

k.set(word);

context.write(k, v);

}

}

}

3）Reducer保持不变

package com.user1.mapreduce.compress;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

IntWritable v = new IntWritable();

@Override

protected void reduce(Text key, Iterable<IntWritable> values,

Context context) throws IOException, InterruptedException {

int sum = 0;

// 1 汇总

for(IntWritable value:values){

sum += value.get();

}

v.set(sum);

// 2 输出

context.write(key, v);

}

}

### 1.6.3 Reduce输出端采用压缩

基于WordCount案例处理。

1）修改驱动

package com.user1.mapreduce.compress;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.compress.BZip2Codec;

import org.apache.hadoop.io.compress.DefaultCodec;

import org.apache.hadoop.io.compress.GzipCodec;

import org.apache.hadoop.io.compress.Lz4Codec;

import org.apache.hadoop.io.compress.SnappyCodec;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountDriver {

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf);

job.setJarByClass(WordCountDriver.class);

job.setMapperClass(WordCountMapper.class);

job.setReducerClass(WordCountReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.setInputPaths(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

// 设置reduce端输出压缩开启

FileOutputFormat.setCompressOutput(job, true);

// 设置压缩的方式

FileOutputFormat.setOutputCompressorClass(job, BZip2Codec.class);

// FileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);

// FileOutputFormat.setOutputCompressorClass(job, DefaultCodec.class);

boolean result = job.waitForCompletion(true);

System.exit(result?0:1);

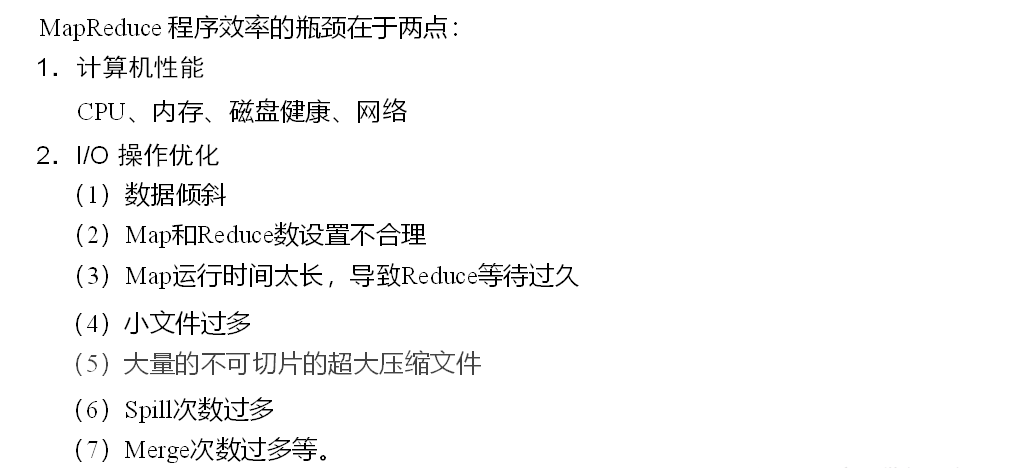
}

}

2）Mapper和Reducer保持不变（详见1.6.2）

# 第2章Hadoop企业优化

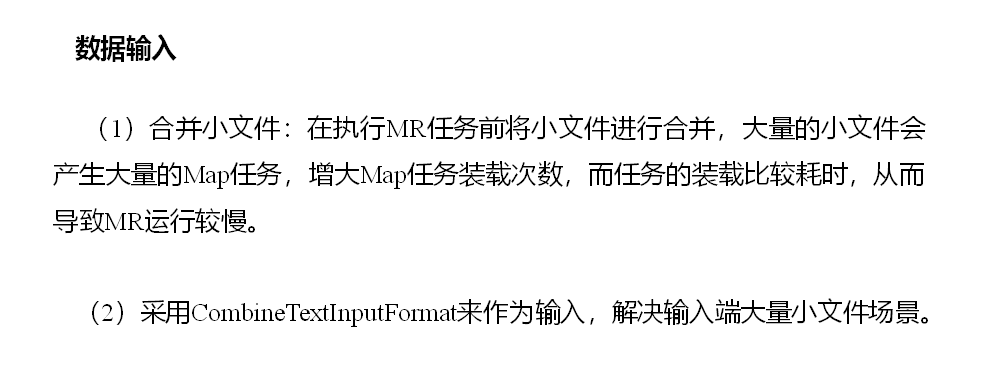
## 2.1 MapReduce 跑的慢的原因



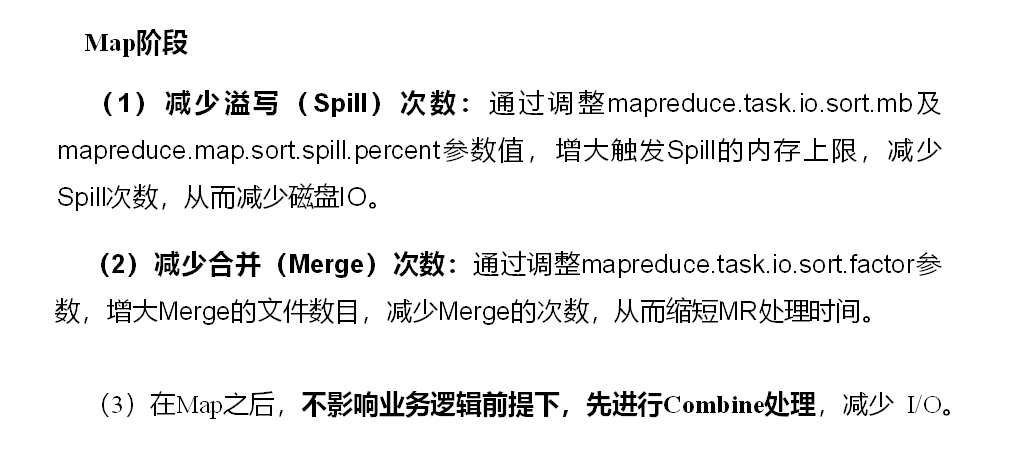
## 2.2 MapReduce优化方法

MapReduce优化方法主要从六个方面考虑：数据输入、Map阶段、Reduce阶段、IO传输、数据倾斜问题和常用的调优参数。

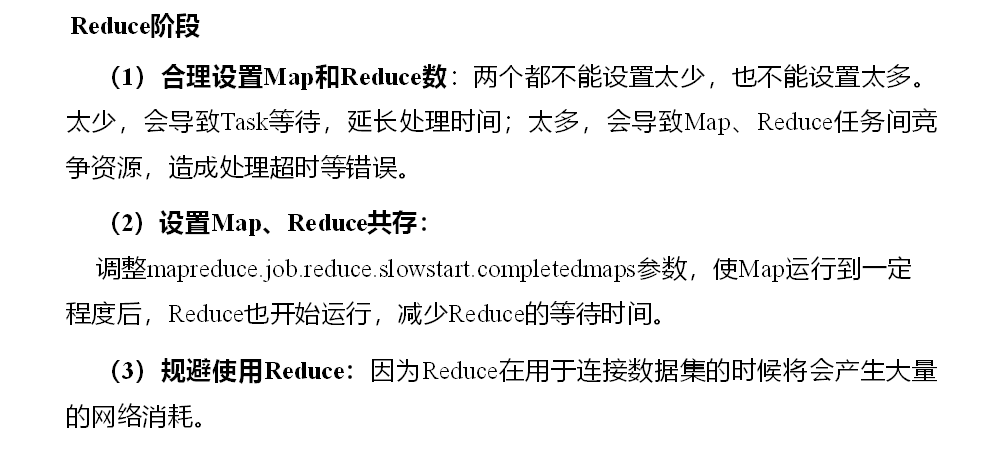
### 2.2.1 数据输入

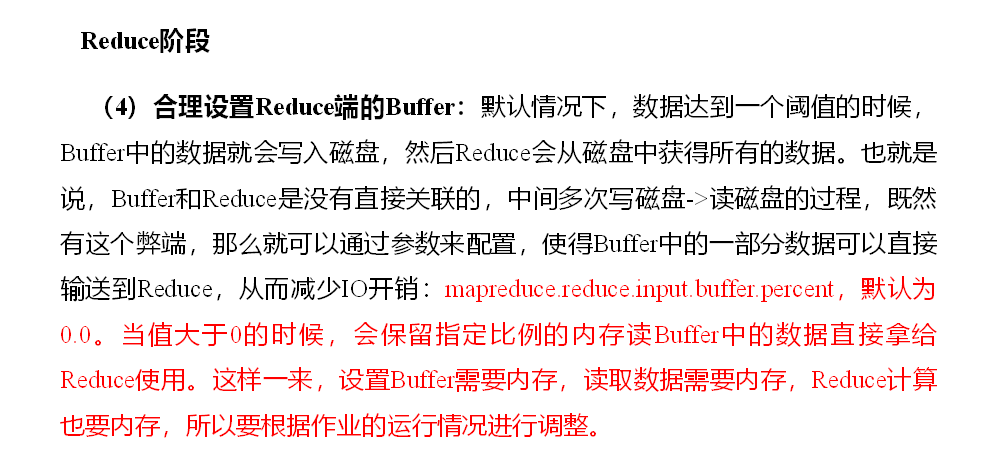


### 2.2.2 Map阶段

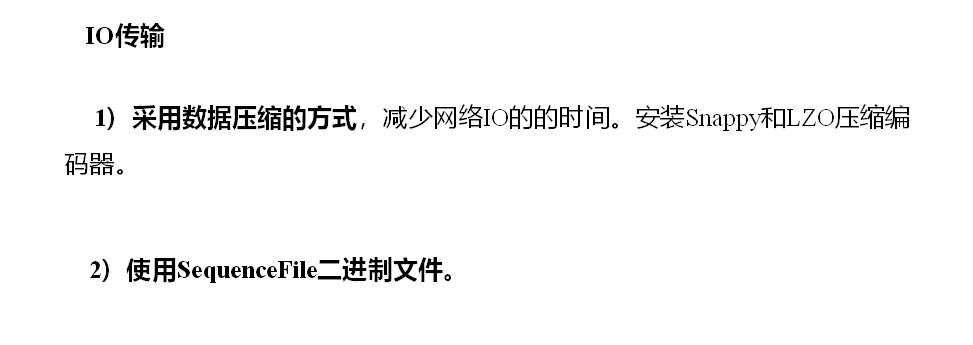


### 2.2.3 Reduce阶段

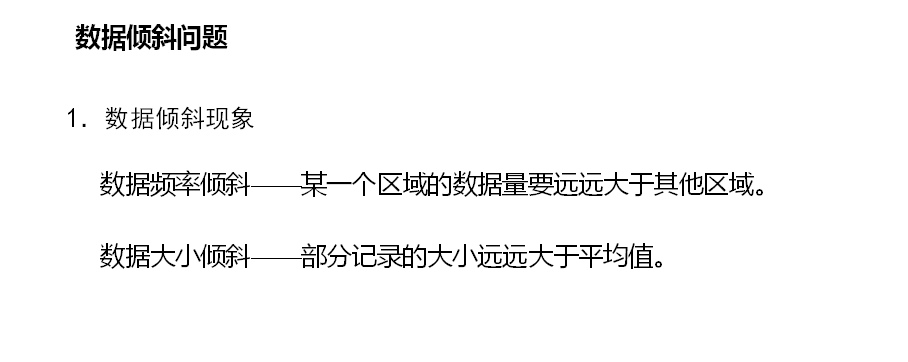


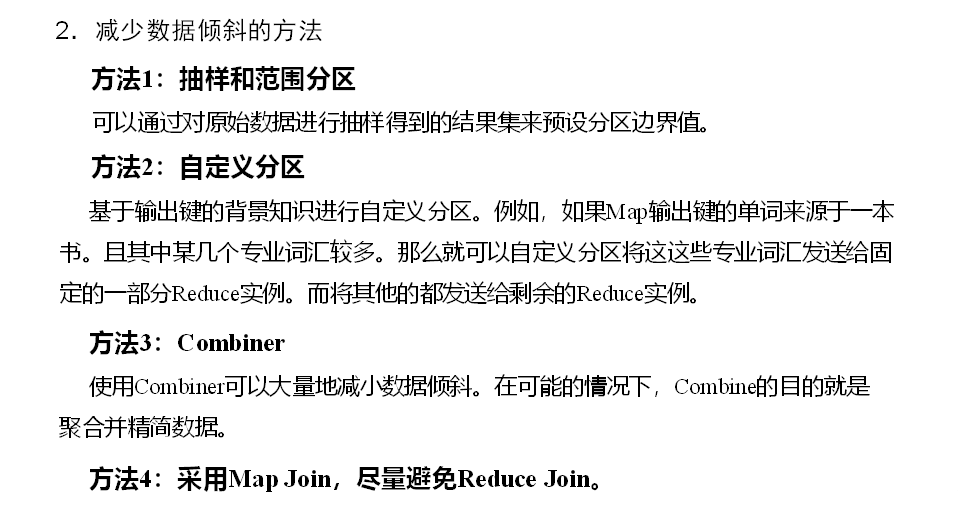


### 2.2.4 I/O传输



### 2.2.5 数据倾斜问题





## 2.3 常用的调优参数

**1）资源相关参数**

（1）以下参数是在用户自己的MR应用程序中配置就可以生效（mapred-default.xml）

|  |  |
| --- | --- |
| 配置参数 | 参数说明 |
| mapreduce.map.memory.mb | 一个MapTask可使用的资源上限（单位:MB），默认为1024。如果MapTask实际使用的资源量超过该值，则会被强制杀死。 |
| mapreduce.reduce.memory.mb | 一个ReduceTask可使用的资源上限（单位:MB），默认为1024。如果ReduceTask实际使用的资源量超过该值，则会被强制杀死。 |
| mapreduce.map.cpu.vcores | 每个MapTask可使用的最多cpu core数目，默认值: 1 |
| mapreduce.reduce.cpu.vcores | 每个ReduceTask可使用的最多cpu core数目，默认值: 1 |
| mapreduce.reduce.shuffle.parallelcopies | 每个Reduce去Map中取数据的并行数。默认值是5 |
| mapreduce.reduce.shuffle.merge.percent | Buffer中的数据达到多少比例开始写入磁盘。默认值0.66 |
| mapreduce.reduce.shuffle.input.buffer.percent | Buffer大小占Reduce可用内存的比例。默认值0.7 |
| mapreduce.reduce.input.buffer.percent | 指定多少比例的内存用来存放Buffer中的数据，默认值是0.0 |

（2）应该在YARN启动之前就配置在服务器的配置文件中才能生效（yarn-default.xml）

|  |  |
| --- | --- |
| 配置参数 | 参数说明 |
| yarn.scheduler.minimum-allocation-mb | 给应用程序Container分配的最小内存，默认值：1024 |
| yarn.scheduler.maximum-allocation-mb | 给应用程序Container分配的最大内存，默认值：8192 |
| yarn.scheduler.minimum-allocation-vcores | 每个Container申请的最小CPU核数，默认值：1 |
| yarn.scheduler.maximum-allocation-vcores | 每个Container申请的最大CPU核数，默认值：32 |
| yarn.nodemanager.resource.memory-mb | 给Containers分配的最大物理内存，默认值：8192 |

（3）Shuffle性能优化的关键参数，应在YARN启动之前就配置好（mapred-default.xml）

|  |  |
| --- | --- |
| 配置参数 | 参数说明 |
| mapreduce.task.io.sort.mb | Shuffle的环形缓冲区大小，默认100m |
| mapreduce.map.sort.spill.percent | 环形缓冲区溢出的阈值，默认80% |

**2）容错相关参数（MapReduce性能优化）**

|  |  |
| --- | --- |
| 配置参数 | 参数说明 |
| mapreduce.map.maxattempts | 每个Map Task最大重试次数，一旦重试次数超过该值，则认为Map Task运行失败，默认值：4。 |
| mapreduce.reduce.maxattempts | 每个Reduce Task最大重试次数，一旦重试次数超过该值，则认为Map Task运行失败，默认值：4。 |
| mapreduce.task.timeout | Task超时时间，经常需要设置的一个参数，该参数表达的意思为：如果一个Task在一定时间内没有任何进入，即不会读取新的数据，也没有输出数据，则认为该Task处于Block状态，可能是卡住了，也许永远会卡住，为了防止因为用户程序永远Block住不退出，则强制设置了一个该超时时间（单位毫秒），默认是600000（10分钟）。如果你的程序对每条输入数据的处理时间过长（比如会访问数据库，通过网络拉取数据等），建议将该参数调大，该参数过小常出现的错误提示是：“AttemptID:attempt\_14267829456721\_123456\_m\_000224\_0 Timed out after 300 secsContainer killed by the ApplicationMaster.”。 |

## 2.4 Hadoop小文件优化方法

### 2.4.1 Hadoop小文件弊端

HDFS上每个文件都要在NameNode上创建对应的元数据，这个元数据的大小约为150byte，这样当小文件比较多的时候，就会产生很多的元数据文件，一方面会大量占用NameNode的内存空间，另一方面就是元数据文件过多，使得寻址索引速度变慢。

小文件过多，在进行MR计算时，会生成过多切片，需要启动过多的MapTask。每个MapTask处理的数据量小，导致MapTask的处理时间比启动时间还小，白白消耗资源。

### 2.4.2 Hadoop小文件解决方案

1. 小文件优化的方向：

（1）在数据采集的时候，就将小文件或小批数据合成大文件再上传HDFS。

（2）在业务处理之前，在HDFS上使用MapReduce程序对小文件进行合并。

（3）在MapReduce处理时，可采用CombineTextInputFormat提高效率。

（4）开启uber模式，实现jvm重用

1. Hadoop Archive

是一个高效的将小文件放入HDFS块中的文件存档工具，能够将多个小文件打包成一个HAR文件，从而达到减少NameNode的内存使用

1. SequenceFile

SequenceFile是由一系列的二进制k/v组成，如果为key为文件名，value为文件内容，可将大批小文件合并成一个大文件

1. CombineTextInputFormat

CombineTextInputFormat用于将多个小文件在切片过程中生成一个单独的切片或者少量的切片。

1. 开启uber模式，实现jvm重用。默认情况下，每个Task任务都需要启动一个jvm来运行，如果Task任务计算的数据量很小，我们可以让同一个Job的多个Task运行在一个Jvm中，不必为每个Task都开启一个Jvm.

开启uber模式，在mapred-site.xml中添加如下配置

<!-- 开启uber模式 -->

<property>

<name>mapreduce.job.ubertask.enable</name>

<value>true</value>

</property>

<!-- uber模式中最大的mapTask数量，可向下修改 -->

<property>

<name>mapreduce.job.ubertask.maxmaps</name>

<value>9</value>

</property>

<!-- uber模式中最大的reduce数量，可向下修改 -->

<property>

<name>mapreduce.job.ubertask.maxreduces</name>

<value>1</value>

</property>

<!-- uber模式中最大的输入数据量，默认使用dfs.blocksize 的值，可向下修改 -->

<property>

<name>mapreduce.job.ubertask.maxbytes</name>

<value></value>

</property>

# 第3章 Hadoop新特性

## 3.1 Hadoop2.x新特性

### 3.1.1 集群间数据拷贝

1）scp实现两个远程主机之间的文件复制

scp -r hello.txt [root@hadoop103:/user/user1/hello.txt](mailto:root@hadoop103:/user/atguigu/hello.txt) // 推 push

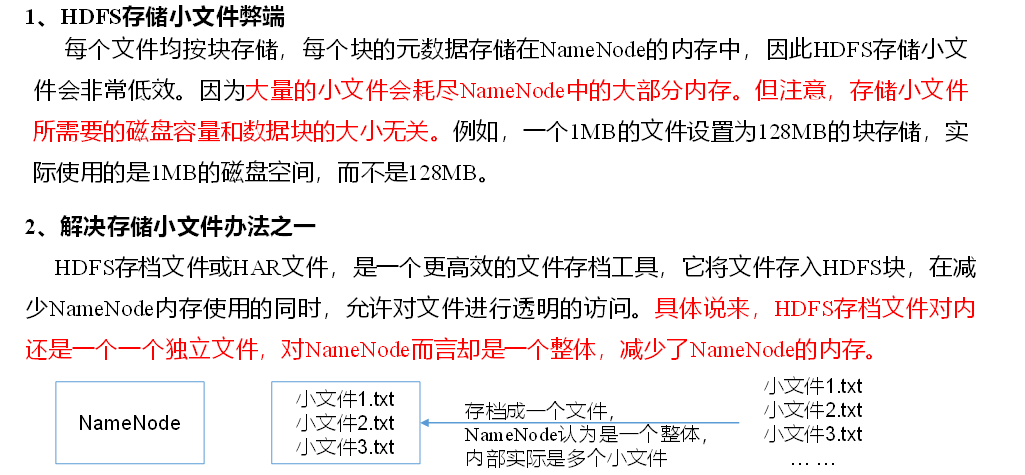
scp -r [root@hadoop103:/user/user1/hello.txt hello.txt](mailto:root@hadoop103:/user/atguigu/hello.txt%20%20hello.txt) // 拉 pull

scp -r [root@hadoop103:/user/user1/hello.txt](mailto:root@hadoop103:/user/atguigu/hello.txt) root@hadoop104:/user/user1 //是通过本地主机中转实现两个远程主机的文件复制；如果在两个远程主机之间ssh没有配置的情况下可以使用该方式。

2）采用distcp命令实现两个Hadoop集群之间的递归数据复制

[user1@hadoop102 hadoop-3.1.3]$ bin/hadoop distcp hdfs://hadoop102:9820/user/user1/hello.txt hdfs://hadoop105:9820/user/user1/hello.txt

### 3.1.2 小文件存档



1）案例实操

（1）需要启动YARN进程

[user1@hadoop102 hadoop-3.1.3]$ start-yarn.sh

（2）归档文件

把/user/user1/input目录里面的所有文件归档成一个叫input.har的归档文件，并把归档后文件存储到/user/user1/output路径下。

[user1@hadoop102 hadoop-3.1.3]$ hadoop archive -archiveName input.har -p /user/user1/input /user/user1/output

（3）查看归档

[user1@hadoop102 hadoop-3.1.3]$ hadoop fs -ls /user/user1/output/input.har

[user1@hadoop102 hadoop-3.1.3]$ hadoop fs -ls har:///user/user1/output/input.har

（4）解归档文件

[user1@hadoop102 hadoop-3.1.3]$ hadoop fs -cp har:/// user/user1/output/input.har/\* /user/user1

### 3.1.3 回收站

开启回收站功能，可以将删除的文件在不超时的情况下，恢复原数据，起到防止误删除、备份等作用。

1）回收站参数设置及工作机制

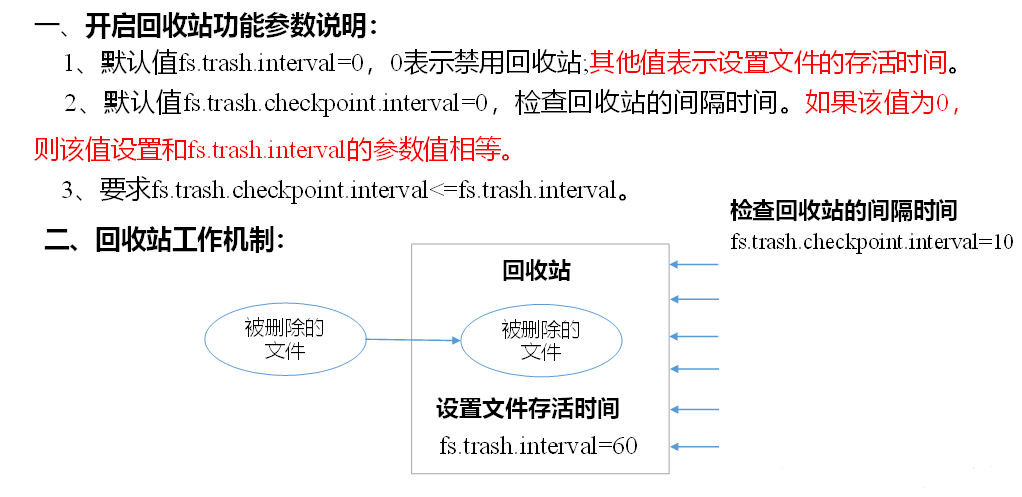


图 回收站

2）启用回收站

修改core-site.xml，配置垃圾回收时间为1分钟。

<property>

<name>fs.trash.interval</name>

<value>1</value>

</property>

3）查看回收站

回收站目录在hdfs集群中的路径：/user/user1/.Trash/….

4）通过程序删除的文件不会经过回收站，需要调用moveToTrash()才进入回收站

Trash trash = New Trash(conf);

trash.moveToTrash(path);

5）通过网页上直接删除的文件也不会走回收站。

6）只有在命令行利用hadoop fs -rm命令删除的文件才会走回收站。

[user1@hadoop102 hadoop-3.1.3]$ hadoop fs -rm -r /user/user1/input

2020-07-14 16:13:42,643 INFO fs.TrashPolicyDefault: Moved: 'hdfs://hadoop102:9820/user/user1/input' to trash at: hdfs://hadoop102:9820/user/user1/.Trash/Current/user/user1/input

7）恢复回收站数据

[user1@hadoop102 hadoop-3.1.3]$ hadoop fs -mv

/user/user1/.Trash/Current/user/user1/input /user/user1/input

## 3.2 Hadoop3.x新特性

### 3.2.1 多NN的HA架构

HDFS NameNode高可用性的初始实现为单个活动NameNode和单个备用NameNode，将edits复制到三个JournalNode。该体系结构能够容忍系统中一个NN或一个JN的故障。

但是，某些部署需要更高程度的容错能力。Hadoop3.x允许用户运行多个备用NameNode。例如，通过配置三个NameNode和五个JournalNode，群集能够容忍两个节点而不是一个节点的故障。

### 3.2.2 纠删码

HDFS中的默认3副本方案在存储空间和其他资源（例如，网络带宽）中具有200％的开销。但是，对于I / O活动相对较低暖和冷数据集，在正常操作期间很少访问其他块副本，但仍会消耗与第一个副本相同的资源量。

纠删码（Erasure Coding）能够在不到50% 的数据冗余情况下提供和3副本相同的容错能力，因此，使用纠删码作为副本机制的改进是自然而然的。

查看集群支持的纠删码策略：hdfs ec -listPolicies

# 第4章 Hadoop HA高可用

## 4.1 HA概述

（1）所谓HA（High Availablity），即高可用（7\*24小时不中断服务）。

（2）实现高可用最关键的策略是消除单点故障。HA严格来说应该分成各个组件的HA机制：HDFS的HA和YARN的HA。

（3）Hadoop2.0之前，在HDFS集群中NameNode存在单点故障（SPOF）。

（4）NameNode主要在以下两个方面影响HDFS集群

* NameNode机器发生意外，如宕机，集群将无法使用，直到管理员重启
* NameNode机器需要升级，包括软件、硬件升级，此时集群也将无法使用

HDFS HA功能通过配置多个NameNodes(Active/Standby)实现在集群中对NameNode的热备来解决上述问题。如果出现故障，如机器崩溃或机器需要升级维护，这时可通过此种方式将NameNode很快的切换到另外一台机器。

## 4.2 HDFS-HA工作机制

通过多个NameNode消除单点故障

### 4.2.1 HDFS-HA工作要点

1）元数据管理方式需要改变

内存中各自保存一份元数据；

Edits日志只有Active状态的NameNode节点可以做写操作；

所有的NameNode都可以读取Edits；

共享的Edits放在一个共享存储中管理（qjournal和NFS两个主流实现）；

2）需要一个状态管理功能模块

实现了一个zkfailover，常驻在每一个namenode所在的节点，每一个zkfailover负责监控自己所在NameNode节点，利用zk进行状态标识，当需要进行状态切换时，由zkfailover来负责切换，切换时需要防止brain split现象的发生。

3）必须保证两个NameNode之间能够ssh无密码登录

4）隔离（Fence），即同一时刻仅仅有一个NameNode对外提供服务

### 4.2.2 HDFS-HA自动故障转移工作机制

自动故障转移为HDFS部署增加了两个新组件：ZooKeeper和ZKFailoverController（ZKFC）进程，如图3-20所示。ZooKeeper是维护少量协调数据，通知客户端这些数据的改变和监视客户端故障的高可用服务。HA的自动故障转移依赖于ZooKeeper的以下功能：

**1．故障检测**

集群中的每个NameNode在ZooKeeper中维护了一个会话，如果机器崩溃，ZooKeeper中的会话将终止，ZooKeeper通知另一个NameNode需要触发故障转移。

**2．现役NameNode选择**

ZooKeeper提供了一个简单的机制用于唯一的选择一个节点为active状态。如果目前现役NameNode崩溃，另一个节点可能从ZooKeeper获得特殊的排外锁以表明它应该成为现役NameNode。

ZKFC是自动故障转移中的另一个新组件，是ZooKeeper的客户端，也监视和管理NameNode的状态。每个运行NameNode的主机也运行了一个ZKFC进程，ZKFC负责：

**1）健康监测**

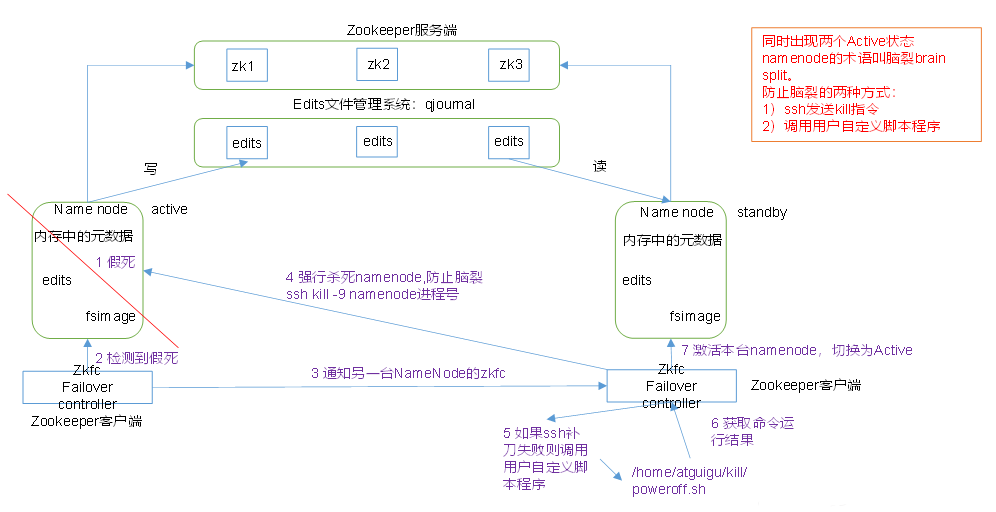
ZKFC使用一个健康检查命令定期地ping与之在相同主机的NameNode，只要该NameNode及时地回复健康状态，ZKFC认为该节点是健康的。如果该节点崩溃，冻结或进入不健康状态，健康监测器标识该节点为非健康的。

**2）ZooKeeper会话管理**

当本地NameNode是健康的，ZKFC保持一个在ZooKeeper中打开的会话。如果本地NameNode处于active状态，ZKFC也保持一个特殊的znode锁，该锁使用了ZooKeeper对短暂节点的支持，如果会话终止，锁节点将自动删除。

**3）基于ZooKeeper的选择**

如果本地NameNode是健康的，且ZKFC发现没有其它的节点当前持有znode锁，它将为自己获取该锁。如果成功，则它已经赢得了选择，并负责运行故障转移进程以使它的本地NameNode为Active。



## 4.3 HDFS-HA集群配置

### 4.3.1 环境准备

（1）修改IP

（2）修改主机名及主机名和IP地址的映射

（3）关闭防火墙

（4）ssh免密登录

（5）安装JDK，配置环境变量等

### 4.3.2 规划集群

|  |  |  |
| --- | --- | --- |
| hadoop102 | hadoop103 | hadoop104 |
| NameNode | NameNode | NameNode |
| JournalNode | JournalNode | JournalNode |
| DataNode | DataNode | DataNode |
| ZKFC | ZKFC | ZKFC |
| ZK | ZK | ZK |
|  | ResourceManager |  |
| NodeManager | NodeManager | NodeManager |

### 4.3.3 配置Zookeeper集群

1）集群规划

在hadoop102、hadoop103和hadoop104三个节点上部署Zookeeper。

2）解压安装

（1）解压Zookeeper安装包到/opt/module/目录下

[user1@hadoop102 software]$ tar -zxvf zookeeper-3.5.7.tar.gz -C /opt/module/

（2）在/opt/module/zookeeper-3.5.7/这个目录下创建zkData

[user1@hadoop102 zookeeper-3.5.7]$ mkdir -p zkData

（3）重命名/opt/module/zookeeper-3.4.14/conf这个目录下的zoo\_sample.cfg为zoo.cfg

[user1@hadoop102 conf]$ mv zoo\_sample.cfg zoo.cfg

3）配置zoo.cfg文件

（1）具体配置

dataDir=/opt/module/zookeeper-3.5.7/zkData

增加如下配置

#######################cluster##########################

server.2=hadoop102:2888:3888

server.3=hadoop103:2888:3888

server.4=hadoop104:2888:3888

（2）配置参数解读

Server.A=B:C:D。

A是一个数字，表示这个是第几号服务器；

B是这个服务器的IP地址；

C是这个服务器与集群中的Leader服务器交换信息的端口；

D是万一集群中的Leader服务器挂了，需要一个端口来重新进行选举，选出一个新的Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

集群模式下配置一个文件myid，这个文件在dataDir目录下，这个文件里面有一个数据就是A的值，Zookeeper启动时读取此文件，拿到里面的数据与zoo.cfg里面的配置信息比较从而判断到底是哪个server。

4）集群操作

（1）在/opt/module/zookeeper-3.5.7/zkData目录下创建一个myid的文件

[user1@hadoop102 zkData]$ touch myid

添加myid文件，注意一定要在linux里面创建，在notepad++里面很可能乱码

（2）编辑myid文件

[user1@hadoop102 zkData]$ vi myid

在文件中添加与server对应的编号：如2

（3）拷贝配置好的zookeeper到其他机器上

[user1@hadoop102 module]$ scp -r zookeeper-3.5.7/ [user1@hadoop103:/opt/module/](mailto:root@hadoop103.atguigu.com:/opt/app/)

[user1@hadoop102 module]$ scp -r zookeeper-3.5.7/ [user1@hadoop104:/opt/module/](mailto:root@hadoop104.atguigu.com:/opt/app/)

并分别修改myid文件中内容为3、4

（4）分别启动zookeeper

[user1@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh start

[user1@hadoop103 zookeeper-3.5.7]$ bin/zkServer.sh start

[user1@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh start

（5）查看状态

[user1@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg

Mode: follower

[user1@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg

Mode: leader

[user1@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg

Mode: follower

### 4.3.4 配置HDFS-HA集群

1）官方地址：<http://hadoop.apache.org/>

2）在opt目录下创建一个ha文件夹

[user1@hadoop102 ~]$ cd /opt

[user1@hadoop102 opt]$ sudo mkdir ha

[user1@hadoop102 opt]$ sudo chown user1:user1 /opt/ha

3）将/opt/module/下的 hadoop-3.1.3拷贝到/opt/ha目录下（记得删除data 和 log目录）

[user1@hadoop102 opt]$ cp -r /opt/module/hadoop-3.1.3 /opt/ha/

4）配置hadoop-env.sh

export JAVA\_HOME=/opt/module/jdk1.8.0\_212

5）配置core-site.xml

<configuration>

<!-- 把多个NameNode的地址组装成一个集群mycluster -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://mycluster</value>

</property>

<!-- 指定hadoop运行时产生文件的存储目录 -->

<property>

<name>hadoop.tmp.dir</name>

<value>/opt/ha/hadoop-3.1.3/data</value>

</property>

</configuration>

6）配置hdfs-site.xml

<configuration>

<!-- NameNode数据存储目录 -->

<property>

<name>dfs.namenode.name.dir</name>

<value>file://${hadoop.tmp.dir}/name</value>

</property>

<!-- DataNode数据存储目录 -->

<property>

<name>dfs.datanode.data.dir</name>

<value>file://${hadoop.tmp.dir}/data</value>

</property>

<!-- JournalNode数据存储目录 -->

<property>

<name>dfs.journalnode.edits.dir</name>

<value>${hadoop.tmp.dir}/jn</value>

</property>

<!-- 完全分布式集群名称 -->

<property>

<name>dfs.nameservices</name>

<value>mycluster</value>

</property>

<!-- 集群中NameNode节点都有哪些 -->

<property>

<name>dfs.ha.namenodes.mycluster</name>

<value>nn1,nn2,nn3</value>

</property>

<!-- NameNode的RPC通信地址 -->

<property>

<name>dfs.namenode.rpc-address.mycluster.nn1</name>

<value>hadoop102:8020</value>

</property>

<property>

<name>dfs.namenode.rpc-address.mycluster.nn2</name>

<value>hadoop103:8020</value>

</property>

<property>

<name>dfs.namenode.rpc-address.mycluster.nn3</name>

<value>hadoop104:8020</value>

</property>

<!-- NameNode的http通信地址 -->

<property>

<name>dfs.namenode.http-address.mycluster.nn1</name>

<value>hadoop102:9870</value>

</property>

<property>

<name>dfs.namenode.http-address.mycluster.nn2</name>

<value>hadoop103:9870</value>

</property>

<property>

<name>dfs.namenode.http-address.mycluster.nn3</name>

<value>hadoop104:9870</value>

</property>

<!-- 指定NameNode元数据在JournalNode上的存放位置 -->

<property>

<name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://hadoop102:8485;hadoop103:8485;hadoop104:8485/mycluster</value>

</property>

<!-- 访问代理类：client用于确定哪个NameNode为Active -->

<property>

<name>dfs.client.failover.proxy.provider.mycluster</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<!-- 配置隔离机制，即同一时刻只能有一台服务器对外响应 -->

<property>

<name>dfs.ha.fencing.methods</name>

<value>sshfence</value>

</property>

<!-- 使用隔离机制时需要ssh秘钥登录-->

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/home/user1/.ssh/id\_rsa</value>

</property>

</configuration>

7）分发配置好的hadoop环境到其他节点

### 4.3.5 启动HDFS-HA集群

1）将HADOOP\_HOME环境变量更改到HA目录(三台机器)

[user1@hadoop102 ~]$ sudo vim /etc/profile.d/my\_env.sh

将HADOOP\_HOME部分改为如下

##HADOOP\_HOME

export HADOOP\_HOME=/opt/ha/hadoop-3.1.3

export PATH=$PATH:$HADOOP\_HOME/bin

export PATH=$PATH:$HADOOP\_HOME/sbin

2）在各个JournalNode节点上，输入以下命令启动journalnode服务

[user1@hadoop102 ~]$ hdfs --daemon start journalnode

[user1@hadoop103 ~]$ hdfs --daemon start journalnode

[user1@hadoop104 ~]$ hdfs --daemon start journalnode

3）在[nn1]上，对其进行格式化，并启动

[user1@hadoop102 ~]$ hdfs namenode -format

[user1@hadoop102 ~]$ hdfs --daemon start namenode

4）在[nn2]和[nn3]上，同步nn1的元数据信息

[user1@hadoop103 ~]$ hdfs namenode -bootstrapStandby

[user1@hadoop104 ~]$ hdfs namenode -bootstrapStandby

5）启动[nn2]和[nn3]

[user1@hadoop103 ~]$ hdfs --daemon start namenode

[user1@hadoop104 ~]$ hdfs --daemon start namenode

6）查看web页面显示

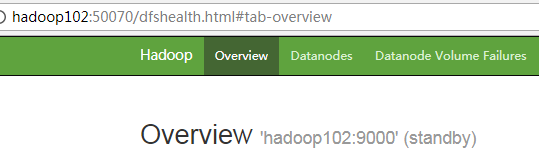


图 hadoop102(standby)

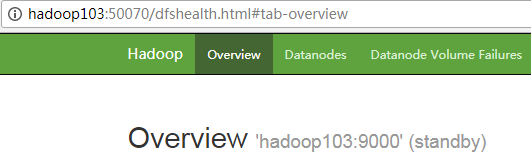


图 hadoop103(standby)

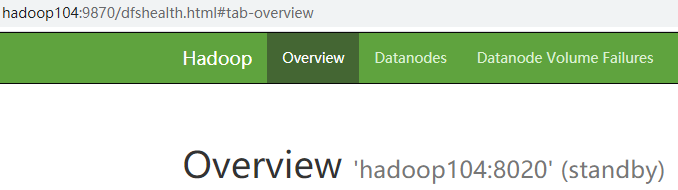


图 hadoop104(standby)

7）在所有节点上，启动datanode

[user1@hadoop102 ~]$ hdfs --daemon start datanode

[user1@hadoop103 ~]$ hdfs --daemon start datanode

[user1@hadoop104 ~]$ hdfs --daemon start datanode

8）将[nn1]切换为Active

[user1@hadoop102 ~]$ hdfs haadmin -transitionToActive nn1

9）查看是否Active

[user1@hadoop102 ~]$ hdfs haadmin -getServiceState nn1

### 4.3.6 配置HDFS-HA自动故障转移

1）具体配置

（1）在hdfs-site.xml中增加

<!-- 启用nn故障自动转移 -->

<property>

<name>dfs.ha.automatic-failover.enabled</name>

<value>true</value>

</property>

（2）在core-site.xml文件中增加

<!-- 指定zkfc要连接的zkServer地址 -->

<property>

<name>ha.zookeeper.quorum</name>

<value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value>

</property>

（3）修改后分发配置文件

[user1@hadoop102 etc]$ pwd

/opt/ha/hadoop-3.1.3/etc

[user1@hadoop102 etc]$ xsync hadoop/

2）启动

（1）关闭所有HDFS服务：

[user1@hadoop102 ~]$ stop-dfs.sh

（2）启动Zookeeper集群：

[user1@hadoop102 ~]$ zkServer.sh start

[user1@hadoop103 ~]$ zkServer.sh start

[user1@hadoop104 ~]$ zkServer.sh start

（3）启动Zookeeper以后，然后再初始化HA在Zookeeper中状态：

[user1@hadoop102 ~]$ hdfs zkfc -formatZK

（4）启动HDFS服务：

[user1@hadoop102 ~]$ start-dfs.sh

（5）可以去zkCli.sh客户端查看Namenode选举锁节点内容：

[zk: localhost:2181(CONNECTED) 7] get -s /hadoop-ha/mycluster/ActiveStandbyElectorLock

myclusternn2 hadoop103 �>(�>

cZxid = 0x10000000b

ctime = Tue Jul 14 17:00:13 CST 2020

mZxid = 0x10000000b

mtime = Tue Jul 14 17:00:13 CST 2020

pZxid = 0x10000000b

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x40000da2eb70000

dataLength = 33

numChildren = 0

3）验证

（1）将Active NameNode进程kill，查看网页端三台Namenode的状态变化

[user1@hadoop102 ~]$ kill -9 namenode的进程id

### 4.3.7 解决NN连接不上JN的问题

自动故障转移配置好以后，然后使用start-dfs.sh群起脚本启动hdfs集群，有可能会遇到NameNode起来一会后，进程自动关闭的问题。查看NameNode日志，报错信息如下：

2020-08-17 10:11:40,658 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 0 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:40,659 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 0 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:40,659 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 0 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:41,660 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 1 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:41,660 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 1 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:41,665 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 1 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:42,661 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 2 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:42,661 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 2 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:42,667 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 2 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:43,662 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 3 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:43,662 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 3 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:43,668 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 3 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:44,663 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 4 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:44,663 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 4 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:44,670 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 4 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:45,467 INFO org.apache.hadoop.hdfs.qjournal.client.QuorumJournalManager: Waited 6001 ms (timeout=20000 ms) for a response for selectStreamingInputStreams. No responses yet.

2020-08-17 10:11:45,664 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 5 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:45,664 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 5 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:45,672 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 5 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:46,469 INFO org.apache.hadoop.hdfs.qjournal.client.QuorumJournalManager: Waited 7003 ms (timeout=20000 ms) for a response for selectStreamingInputStreams. No responses yet.

2020-08-17 10:11:46,665 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 6 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:46,665 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 6 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:46,673 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 6 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:47,470 INFO org.apache.hadoop.hdfs.qjournal.client.QuorumJournalManager: Waited 8004 ms (timeout=20000 ms) for a response for selectStreamingInputStreams. No responses yet.

2020-08-17 10:11:47,666 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 7 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:47,667 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 7 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:47,674 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 7 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:48,471 INFO org.apache.hadoop.hdfs.qjournal.client.QuorumJournalManager: Waited 9005 ms (timeout=20000 ms) for a response for selectStreamingInputStreams. No responses yet.

2020-08-17 10:11:48,668 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 8 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:48,668 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 8 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:48,675 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 8 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:49,669 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop102/192.168.6.102:8485. Already tried 9 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:49,673 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop104/192.168.6.104:8485. Already tried 9 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:49,676 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hadoop103/192.168.6.103:8485. Already tried 9 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)

2020-08-17 10:11:49,678 WARN org.apache.hadoop.hdfs.server.namenode.FSEditLog: Unable to determine input streams from QJM to [192.168.6.102:8485, 192.168.6.103:8485, 192.168.6.104:8485]. Skipping.

org.apache.hadoop.hdfs.qjournal.client.QuorumException: Got too many exceptions to achieve quorum size 2/3. 3 exceptions thrown:

192.168.6.103:8485: Call From hadoop102/192.168.6.102 to hadoop103:8485 failed on connection exception: java.net.ConnectException: 拒绝连接; For more details see: http://wiki.apache.org/hadoop/ConnectionRefused

192.168.6.102:8485: Call From hadoop102/192.168.6.102 to hadoop102:8485 failed on connection exception: java.net.ConnectException: 拒绝连接; For more details see: http://wiki.apache.org/hadoop/ConnectionRefused

192.168.6.104:8485: Call From hadoop102/192.168.6.102 to hadoop104:8485 failed on connection exception: java.net.ConnectException: 拒绝连接; For more details see: http://wiki.apache.org/hadoop/ConnectionRefused

查看报错日志，可分析出报错原因是因为NameNode连接不上JournalNode，而利用jps命令查看到三台JN都已经正常启动，为什么NN还是无法正常连接到JN呢？这是因为start-dfs.sh群起脚本默认的启动顺序是先启动NN，再启动DN，然后再启动JN，并且默认的rpc连接参数是重试次数为10，每次重试的间隔是1s，也就是说启动完NN以后的10s中内，JN还启动不起来，NN就会报错了。

Core-default.xml里面有两个参数如下：

<!-- NN连接JN重试次数，默认是10次 -->

<property>

<name>ipc.client.connect.max.retries</name>

<value>10</value>

</property>

<!-- 重试时间间隔，默认1s -->

<property>

<name>ipc.client.connect.retry.interval</name>

<value>1000</value>

</property>

解决方案：遇到上述问题后，可以稍等片刻，等JN成功启动后，手动启动下三台NN：

[user1@hadoop102 ~]$ hdfs --daemon start namenode

[user1@hadoop103 ~]$ hdfs --daemon start namenode

[user1@hadoop104 ~]$ hdfs --daemon start namenode

也可以在core-site.xml里面适当调大上面的两个参数：

<!-- NN连接JN重试次数，默认是10次 -->

<property>

<name>ipc.client.connect.max.retries</name>

<value>20</value>

</property>

<!-- 重试时间间隔，默认1s -->

<property>

<name>ipc.client.connect.retry.interval</name>

<value>5000</value>

</property>

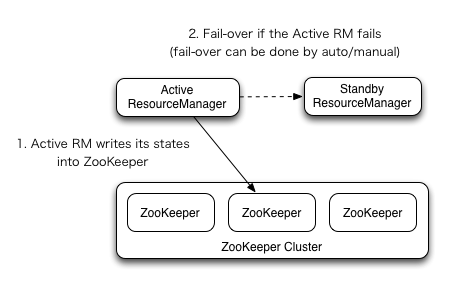
## 4.4 YARN-HA配置

### 4.4.1 YARN-HA工作机制

1）官方文档：

[http://hadoop.apache.org/docs/r3.1.3/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html](http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html)

2）YARN-HA工作机制



### 4.4.2 配置YARN-HA集群

1）环境准备

（1）修改IP

（2）修改主机名及主机名和IP地址的映射

（3）关闭防火墙

（4）ssh免密登录

（5）安装JDK，配置环境变量等

（6）配置Zookeeper集群

2）规划集群

|  |  |  |
| --- | --- | --- |
| hadoop102 | hadoop103 | hadoop104 |
| NameNode | NameNode | NameNode |
| JournalNode | JournalNode | JournalNode |
| DataNode | DataNode | DataNode |
| ZKFC | ZKFC | ZKFC |
| ZK | ZK | ZK |
| ResourceManager | ResourceManager |  |
| NodeManager | NodeManager | NodeManager |

3）具体配置

（1）yarn-site.xml

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<!-- 启用resourcemanager ha -->

<property>

<name>yarn.resourcemanager.ha.enabled</name>

<value>true</value>

</property>

<!-- 声明两台resourcemanager的地址 -->

<property>

<name>yarn.resourcemanager.cluster-id</name>

<value>cluster-yarn1</value>

</property>

<!--指定resourcemanager的逻辑列表-->

<property>

<name>yarn.resourcemanager.ha.rm-ids</name>

<value>rm1,rm2</value>

</property>

<!-- ========== rm1的配置 ========== -->

<!-- 指定rm1的主机名 -->

<property>

<name>yarn.resourcemanager.hostname.rm1</name>

<value>hadoop102</value>

</property>

<!-- 指定rm1的web端地址 -->

<property>

<name>yarn.resourcemanager.webapp.address.rm1</name>

<value>hadoop102:8088</value>

</property>

<!-- 指定rm1的内部通信地址 -->

<property>

<name>yarn.resourcemanager.address.rm1</name>

<value>hadoop102:8032</value>

</property>

<!-- 指定AM向rm1申请资源的地址 -->

<property>

<name>yarn.resourcemanager.scheduler.address.rm1</name>

<value>hadoop102:8030</value>

</property>

<!-- 指定供NM连接的地址 -->

<property>

<name>yarn.resourcemanager.resource-tracker.address.rm1</name>

<value>hadoop102:8031</value>

</property>

<!-- ========== rm2的配置 ========== -->

<!-- 指定rm2的主机名 -->

<property>

<name>yarn.resourcemanager.hostname.rm2</name>

<value>hadoop103</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address.rm2</name>

<value>hadoop103:8088</value>

</property>

<property>

<name>yarn.resourcemanager.address.rm2</name>

<value>hadoop103:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address.rm2</name>

<value>hadoop103:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address.rm2</name>

<value>hadoop103:8031</value>

</property>

<!-- 指定zookeeper集群的地址 -->

<property>

<name>yarn.resourcemanager.zk-address</name>

<value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value>

</property>

<!-- 启用自动恢复 -->

<property>

<name>yarn.resourcemanager.recovery.enabled</name>

<value>true</value>

</property>

<!-- 指定resourcemanager的状态信息存储在zookeeper集群 -->

<property>

<name>yarn.resourcemanager.store.class</name> <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>

<!-- 环境变量的继承 -->

<property>

<name>yarn.nodemanager.env-whitelist</name>

<value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLASSPATH\_PREPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>

</property>

</configuration>

（2）同步更新其他节点的配置信息，分发配置文件

[user1@hadoop102 etc]$ xsync hadoop/

4）启动hdfs

[user1@hadoop102 ~]$ start-dfs.sh

5）启动YARN

（1）在hadoop102或者hadoop103中执行：

[user1@hadoop102 ~]$ start-yarn.sh

（2）查看服务状态

[user1@hadoop102 ~]$ yarn rmadmin -getServiceState rm1

（3）可以去zkCli.sh客户端查看ResourceManager选举锁节点内容：

[user1@hadoop102 ~]$ zkCli.sh

[zk: localhost:2181(CONNECTED) 16] get -s /yarn-leader-election/cluster-yarn1/ActiveStandbyElectorLock

cluster-yarn1rm1

cZxid = 0x100000022

ctime = Tue Jul 14 17:06:44 CST 2020

mZxid = 0x100000022

mtime = Tue Jul 14 17:06:44 CST 2020

pZxid = 0x100000022

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x30000da33080005

dataLength = 20

numChildren = 0

（4）web端查看hadoop102:8088和hadoop103:8088的YARN的状态，和NameNode对比，查看区别

