hadoop config and management

关于 *hadoop* 配置管理的手册

屈庆磊

quqinglei@icloud.com

2013 年 *7* 月

目录

1 写在前面

此文档或许有存在的意义吧，如果觉得不爽，删掉就 OK 啦，文档目的是注重实际，本来是我的学习笔记，未来希望能写成一本很容易理解又很容易上手的使用手册，面向一些具体的问题进行分析、解决，后期将会从 Java API 的角度，一些特定 mapreduce 案例的编写来讲解 hadoop，如果一切顺利，此文档在后面将对 Hadoop 本身代码进行分析，可此路荆棘，非一时之功，后期将讲解如何优化 Hadoop，并给出一些测试方法和衡量方法。内容大部分来自网络的各位网友们的总结以及 Hadoop 官网给出的帮助文档。本文档将尽可能的短小精悍，未来将控制在 200 页左右，并由易到难，而不是和书本式的照本宣科。此文本身可能出现一些写作上的失误或者错误，如有发现请邮件我，谢谢！

2 环境变量

本例中采用和很多环境变量，如：`JAVA_HOME HADOOP_HOME` 等等，建议在做以下工作的时候，先找一个文本文件，把环境变量都写好，然后在写入`/etc/profile` 文件，当然有些环境变量只是在编译的时候使用，这个你可以根据情况酌情处理，当然环境变量都导出来也是没有什么问题的。

3 配置 JAVA 环境

3.1 配置 JAVA_HOME

需要根据系统位数下载不同的安装包，如：32 位的 jdk-7u25-linux-i586.gz，或者 64 位的 jdk-7u25-linux-x64.tar.gz 建议去甲骨文官网下载。

例子：

```
tar xzv jdk-7u25-linux-i586.gz -C /opt
cd /opt
ln -s jdk1.7.0_25 jdk
echo "export JAVA_HOME=/opt/jdk" >> /etc/profile
source /etc/profile
```

4 Hadoop 安装包

由于在 *Linux* 下有很多安装方式，为了统一，本次采用解压包介绍。 去 *Hadoop* 的官网，找到 *Hadoop* 的下载地址，在稳定版本里挑一个下载，本例中采用的包为：*hadoop-1.1.2.tar.gz*

5 Hadoop 集群安装实例

本例中有三台机器参与，一台 *master*，两台 *slave*，配置比较简单

5.1 说明

192.168.1.5 此台机器作为 master

192.168.1.6 此台机器作为 slave01

192.168.1.7 此台机器作为 slave02

5.2 第二步，配置 hosts

配置 *Hosts*： 编辑 master 机器的/etc/hosts 文件，添加内容为如下所示：

```
192.168.1.5 master
192.168.1.6 slave01
192.168.1.7 slave02
```

同步此配置文件到其它两台机器，保持一致即可。

5.3 第三步，配置 JAVA_HOME 见上述章节

5.4 第四步，建立新用户，并解压安装包

```
useradd -m hadoop
passwd hadoop # 设置密码
su hadoop
cd /home/hadoop
tar xzf hadoop-1.1.2.tar.gz
ln -s hadoop-1.1.2 hadoop
#hadoop 的家路径此时为：/home/hadoop/hadoop
```

5.5 第四步，配置 ssh 无密码访问

其实就是交换公匙

```
# machine master, 使用 hadoop 用户
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
scp ~/.ssh/id_dsa.pub hadoop@192.168.1.6:/home/hadoop/.ssh/authorized_keys_master
scp ~/.ssh/id_dsa.pub hadoop@192.168.1.6:/home/hadoop/.ssh/authorized_keys_master

# slave01, 使用 hadoop 用户
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
cat ~/.ssh/authorized_keys_master >> ~/.ssh/authorized_keys
scp ~/.ssh/id_dsa.pub hadoop@192.168.1.5:/home/hadoop/.ssh/authorized_keys_slave01

# slave02, 使用 hadoop 用户
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
cat ~/.ssh/authorized_keys_master >> ~/.ssh/authorized_keys
scp ~/.ssh/id_dsa.pub hadoop@192.168.1.5:/home/hadoop/.ssh/authorized_keys_slave02

# master, 使用 hadoop 用户
cat ~/.ssh/authorized_keys_slave01 >> ~/.ssh/authorized_keys
cat ~/.ssh/authorized_keys_slave02 >> ~/.ssh/authorized_keys
```

注意：在 CentOS 下做的同学需要注意，一定要看看 /.ssh/authorized_keys 的权限是不是 644，如果不是 644，请 `chmod 644 authorized_keys`

5.6 第五步，更改 Hadoop 配置文件

```
su hadoop
cd /home/hadoop/hadoop/conf/
# 需要更改：masters, slaves, hadoop-env.sh, core-site.xml, hdfs-site.xml, mapred-site.xml
```

更改结果如下所示：

```
#masters 文件内容
master

#slaves 文件内容
slave01
slave02

#hadoop-env.sh
# 仅仅需要更改 JAVA_HOME 那一行，更改为 JAVA_HOME 的实际路径，本例中我们采用的 JAVA_HOME 为：
export JAVA_HOME=/opt/jdk

#core-site.xml 文件内容
<?xml version="1.0"?>
```

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://master:9000</value>
</property>
</configuration>

#hdfs-site.xml 文件内容
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>dfs.name.dir</name>
<value>/home/hadoop/hadoopfs/name1,/home/hadoop/hadoopfs/name2</value>
</property>

<property>
<name>dfs.data.dir</name>
<value>/home/hadoop/hadoopfs/data1,/home/hadoop/hadoopfs/data2</value>
</property>

<property>
<name>dfs.replication</name>
<value>2</value>
</property>
</configuration>

#mapred-site.xml 文件内容
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>mapred.job.tracker</name>
<value>master:9200</value>
</property>
</configuration>
```

5.7 第六步，把 hadoop 打包，复制到其他两台机器

在配置完一台机器后，直接可以把配置好的 *hadoop* 复制到其他机器上，当然在其他的机器上的位置应该是一样的

5.8 完毕

基本配置完毕，在主机的/home/hadoop/hadoop/bin 下执行：./hadoop namenode -format 格式化文件系统，然后就可以启动 hadoop 集群了，./start-all.sh 即可启动 hadoop 集群

```
cd /home/hadoop/hadoop
./hadoop fs namenode -format
./start-all.sh
```

如果成功，我们可以在其他机器上都能看到 hadoop 的进程，并且在/home/hadoop/ 会看到/home/hadoop/hadoopfs 的路径出现。

6 编译 Hadoop 的 FUSE 模块，机器一定要能上网

如果想在其他机器上挂载 *HDFS*，则需要编译 *fuse_dfs* 模块

6.1 编译前的软件依赖安装

```
# 如果系统是基于 Redhat 的，请使用 yum 安装，如果基于 debian，使用 apt-get 安装，具体如下：
yum install install automake autoconf m4 libtool pkgconfig fuse fuse-devel fuse-libs

apt-get install automake autoconf m4 libtool libextutils-pkgconfig-perl \
        fuse libfuse-dev lrzsz build-essential
```

6.2 下载 ant 包，编译需要它

```
tar xzf apache-ant-1.9.1-bin.tar.gz -C /opt
cd /opt
ln -s apache-ant-1.9.1 ant
```

6.3 设置环境变量

把下面环境变量写到*/etc/profile* 文件里

```
# add below to /etc/profile
export JAVA_HOME=/opt/jdk
export HADOOP_HOME=/home/hadoop/hadoop
export OS_ARCH=i386 #or amd64
export OS_BIT=32 #or 64
export ANT_HOME=/opt/ant
```

```
export PATH=$PATH:$ANT_HOME/bin
export LD_LIBRARY_PATH=$JAVA_HOME/jre/lib/$OS_ARCH/server: \
        $HADOOP_HOME/build/c++/Linux-$OS_ARCH-$OS_BIT/lib:/usr/local/lib:/usr/lib
```

6.3.1 更新环境变量

```
source /etc/profile
```

6.3.2 编译 HDFS

```
cd /home/hadoop/hadoop # 进入 hadoop 的家路径
ant compile-c++-libhdfs -Dlibhdfs=1 -Dcompile.c++=1 # 编译 libhdfs，机器一定要能上网，否则无法编译
```

6.4 编译 fuse_dfs

```
# 进入 hadoop 家路径，执行
ln -s c++/Linux-$OS_ARCH-$OS_BIT/lib build/libhdfs
ant compile-contrib -Dlibhdfs=1 -Dfusedfs=1
```

如果以上不出错，则说明问题解决 编译完成后挂载就很简单了，但应该注意以下问题：

(1) 编辑/etc/ld.so.conf 添加 libhdfs.so 文件的路径，或者可以直接把：/home/hadoop/hadoop/c++/Linux-i386-32/lib/ 里的所有文件拷贝到/usr/lib 路径，记得执行 ldconfig

(2) 可以把/home/hadoop/build/contrib/fuse-dfs 路径里的两个文件拷贝到/usr/local/bin/ 但必须更改fuse_dfs_wrapper.sh 的最后一行./fuse_dfs 为：/usr/local/bin/fuse_dfs, fuse_dfs_wrapper.sh 是个脚本，请给予执行权限，并可以更改里面的环境变量为真正的值，如果环境变量已设为全局，则不需要更改。

7 编译并执行经典的例子程序 WordCount

7.1 例子代码

此例子是经过改动的，如果直接使用 *Hadoop* 官网的例子在新版本的 *Hadoop 1.1.2* 会出错，改动处有标记。

```
// WordCount.java
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

        public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
                private final static IntWritable one = new IntWritable(1);
                private Text word = new Text();

                public void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                                String line = value.toString();
                                StringTokenizer tokenizer = new StringTokenizer(line);
                                while (tokenizer.hasMoreTokens()) {
                                        word.set(tokenizer.nextToken());
                                        context.write(word, one);
                                }
                        }
        }

        public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

                public void reduce(Text key, Iterable<IntWritable> values, Context context)
                        throws IOException, InterruptedException {
                                int sum = 0;
                                for (IntWritable val : values) {
                                        sum += val.get();
                                }
                                context.write(key, new IntWritable(sum));
                        }
        }

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

                Job job = new Job(conf, "wordcount");

                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                job.setJarByClass(WordCount.class); // 多加了这一行!!!

                job.setMapperClass(Map.class);
                job.setReducerClass(Reduce.class);

                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
                job.waitForCompletion(true);
        }

}
```

7.2 编译过程

废话少说，直接看 *Makefile* 和文件列表，我觉得模仿是最好的学习方式

```
hadoop@debian:~/testHadoop/wordcount$ tree
.
├── Makefile # 用来编译例子的 Makefile
├── run.sh # 用来执行的，其实里面就是一行命令，我懒得记
├── wordcount_classes # 这是个空的文件夹，用于放编译完的 CLASS 文件
└── WordCount.java # 这个是上面那个例子的源文件

1 directory, 3 files

# Makefile 内容
all: wordcount jarpackage

wordcount:
        javac -classpath ${CLASSPATH} -d  wordcount_classes WordCount.java

jarpackage:
        jar -cvf wordcount.jar -C wordcount_classes/ .

clean:
        rm -rf wordcount_classes/*
        rm -f wordcount.jar

# run.sh 内容
#!/bin/bash

hadoop fs -rmr /user/hadoop/output/ # output 路径不能存在，否则执行失败
hadoop jar wordcount.jar org.myorg.WordCount input output
# input output 默认是在/user/hadoop/input /user/hadoop/output
# 如果 input 路径不是这个请写绝对路径

# 编译后的 wordcount_classes 路径：
hadoop@debian:~/testHadoop/wordcount$ tree wordcount_classes/
wordcount_classes/
└── org
    └── myorg
        ├── WordCount.class
        ├── WordCount$Map.class
        └── WordCount$Reduce.class
```

```
2 directories, 3 files
```

注意：执行 run.sh 时，注意 org.myorg.WordCount 是否和上面的 wordcount_classes 结构一致！

8 hadoop 配置进阶

8.1 配置垃圾箱

想必大家都犯过类似的错误，不小心把一大堆数据全删掉了，咋办？这个时候你会很崩溃的，如果是在操作系统的文件系统下你还能使用恢复软件，检查 *inode* 节点，恢复数据，但在 *hadoop* 上貌似没那么简单，所以有一部时间机器是很有用的，那么我们可以开启垃圾箱功能，并设置系统自动删除过期数据的超时时间，这样至少在我们想起删错东西的时候可以挽回带来的损失。

8.1.1 配置方法

只需要更改配置文件：*core-site.xml*，*value* 的单位是分钟。

```
<property>
        <name>fs.trash.interval</name>
        <value>1440</value>
</property>
```

8.1.2 还原方法

只需要把文件从 *.Trash* 文件夹里移动到原来的位置即可

```
# 比如我在/user/test/lei/ 路径删除一个文件 abc，那么我应该去哪找呢？
# 其实位置应该在 /user/hadoop/.Trash/Current/user/test/lei/ 路径去找
# 注意上面路径中的 hadoop 其实是用户名，我们直接可以把文件移动过去就可以了

$ hadoop fs -mv /user/hadoop/.Trash/Current/user/test/lei/abc /user/test/lei/abc

# 清理垃圾
$ hadoop fs -rmr /user/hadoop/.Trash
$
```

8.2 磁盘预留剩余空间

这个参数有点脱了裤子放屁，对于一个正常的软件都应该有防止硬盘写满的测试，或许是为了速度吧，在 *Linux* 下如果硬盘写满了，会导致一些问题，比如无法写入问题，然后硬盘看着像只读似的，所以尽量不要让你的硬盘写满，所以这个选项还是有必要的，特别是对于那些硬盘本来就不是多宽裕个兄弟们。

下面是配置代码，直接写到：hdfs-site.xml 文件里就可以啦！value 的单位是字节

```
<property>
<name>dfs.datanode.du.reserved</name>
<value>10737418240</value>
<description>Reserved space in bytes per volume
</description>
</property>
```

9 Hadoop 管理

9.1 文件操作类命令

这行命令和一些 *UNIX* 命令类似，所以比较容易记住，本文档为学习而写，为记忆而写。

cat 查看文件内容，类似于 UNIX 系统中的 cat 命令

```
# Usage: hadoop fs -cat URL [URI ...]
hadoop fs -cat hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2
hadoop fs -cat file://file3 /user/hadoop/file4

# 返回值：0: 正确 -1: 错误
```

chgrp 改变文件所属于的组，使用 -R 参数可以递归执行此路径的所有文件，文件必须属于执行此命令者或者执行命令者为超级用户。

```
# Usage: hadoop fs -chgrp [-R] GROUP URI [URI ...]
```

chmod 更改文件的权限，忘 UNIX 的 chmod 上靠就对了，使用 -R 递归整个路径

```
# Usage: hadoop fs -chmod [-R] [OWNER] [:[GROUP]] URI [URI]
```

chown 改变文件所属于的用户，使用 -R 递归整个路径

```
# Usage: hadoop fs -chown [-R] [OWNER] [:[GROUP]] URI [URI]
```

copyFromLocal 和 put 命令类似，只不过源被限制为本地文件

```
# Usage: hadoop fs -copyFromLocal <localsrc> URI
```

cp 复制命令

```
# Usage: hadoop fs -cp URI [URI ...] <dest>
Example:
hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2
hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir

# 返回值：0: 正确 -1: 错误
```

du 估计文件空间使用情况

```
# Usage: hadoop fs -du URI [URI ...]
Example:
hadoop fs -du /user/hadoop/dir1 /user/hadoop/file2 \
        hdfs://nn.example.com/user/hadoop/dir1


# 返回值：0: 正确 -1: 错误
```

dus 估计某路径的空间使用情况

```
# Usage: hadoop fs -dus <args>
# Example:
hadoop fs -dus /user/hadoop/dir1
```

expunge 清空垃圾箱，或者比喻为清空回收站

```
# Usage: hadoop fs -expunge
```

get 下载文件到本地

```
# Usage: hadoop fs -get [-ignorecrc] [-crc] <src> <localdst>
# Example:
hadoop fs -get /user/hadoop/file localfile
hadoop fs -get hdfs://nn.example.com/user/hadoop/file localfile
```

getmerge 下载之前合并，下载后即为合并的文件，也就是说如果在 HDFS 上有一个路径，里面都是小文件，我们可以使用此命令下载此路径的所有文件并在下载后形成一个合并的文件。[addnl] 就是在合并后的文件结尾加个换行符

```
# Usage: hadoop fs -getmerge <src> <localdst> [addnl]
# Example:
hadoop fs -getmerge /user/hadoop/test localfile
```

ls 莫要多讲，和 UNIX 下的 ls 功能一样

```
# Usage: hadoop fs -ls <args>
# Example:
hadoop fs -ls /user/hadoop/test hdfs://nn.example.com/user/hadoop/dir1

# 返回值：0: 正确 -1: 错误
```

lsr UNIX 下的 ls -R 功能一样，递归列出本路径所有文件

```
# Usage: hadoop fs -lsr <args>
```

mkdir UNIX 下的 mkdir 功能一样，创建路径

```
# Usage: hadoop fs -mkdir <paths>
# Example:
hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
hadoop fs -mkdir hdfs://nn1.example.com/user/hadoop/dir \
         hdfs://nn2.example.com/user/hadoop/dir2

# 返回值：0: 正确 -1: 错误
```

mv UNIX 下的 mv 功能一样，移动文件

```
# Usage: hadoop fs -mv URI [URI ...] <dest>
# Example:
hadoop fs -mv /user/hadoop/file1 /user/hadoop/file2
hadoop fs -mv hdfs://nn.example.com/file1 hdfs://nn.example.com/file2 \
        hdfs://nn.example.com/file3 hdfs://nn.example.com/dir1

# 返回值：0: 正确 -1: 错误
```

put 上传文件到 HDFS

```
# Usage: hadoop fs -put <localsrc> ... <dst>
# Example:
hadoop fs -put localfile /user/hadoop/hadoopfile
hadoop fs -put localfile1 localfile2 /user/hadoop/hadoopdir
hadoop fs -put localfile hdfs://nn.example.com/hadoop/hadoopfile
hadoop fs -put - hdfs://nn.example.com/hadoop/hadoopfile # 从标准输入读取，并上传

# 返回值：0: 正确 -1: 错误
```

rm 删除文件

```
# Usage: hadoop fs -rm URI [URI ...]
# Example:
hadoop fs -rm hdfs://nn.example.com/hadoop/hadoopfile /user/hadoop/emptydir

# 返回值：0: 正确 -1: 错误
```

rmr 递归删除文件，和 UNIX 中的 rm -r 差不多

```
# Usage: hadoop fs -rmr URI [URI ...]
# Example:
hadoop fs -rmr /user/hadoop/dir
hadoop fs -rmr hdfs://nn.example.com/hadoop/hadoopfile /user/hadoop/emptydir

# 返回值：0: 正确 -1: 错误
```

setrep 设置文件的复制因子数，说白了就是手动设置文件的副本数，使用 -R 可以递归设置一个路径下的所有文件

```
# Usage: hadoop fs -setrep [-R] <path>
# Example:
hadoop fs -setrep -w 3 -R /user/hadoop/dir1 # -w 参数指的是副本数量

# 返回值：0: 正确 -1: 错误
```

stat 查看路径信息

```
# Usage: hadoop fs -stat URI [URI ...]
# Example:
hadoop fs -stat path

# 返回值：0: 正确 -1: 错误
```

tail 查看文件最后字节到标准输出，和 UNIX 下的 tail 类似

```
# Usage: hadoop fs -tail pathname

# 返回值：0: 正确 -1: 错误
```

test 查看文件是否存在，是否为 0，或者是否是路径，使用 -d 选项时，如果返回值是 0 则是路径，如果返回值是 1 则是文件，-1 则说明路径不存在

```
# Usage: hadoop fs -test -[ezd] URI
# -e 测试文件是否存在，如果存在返回 0
# -z 测试文件是否为空，如果空返回 0
# -d 测试文件是否是路径，如果是返回 0，如果是文件返回 1，其他返回-1，在 UNIX 返回值上看应该是 255
# Example:
hadoop fs -test -e filename
```

text 以文本方式查看文件，支持 zip 以及 TextRecordInputStream[tbd] 格式的文件

```
# Usage: hadoop fs -text <src>
```

touchz 创建个空文件

```
# Usage: hadoop fs -touchz URI [URI ...]
# Example:
hadoop -touchz pathname

# 返回值：0: 正确 -1: 错误
```

9.2 限额管理

如果很多人使用 *hadoop*，设置限额是很有意义的，*Hadoop quota* 的设定是针对路径，而不是针对账号所以管理上最好每个账号只能写入一个目录，关于权限管理请查看 *9.3*

(1) 设定方式，设定方式有两种，一种是命令空间[1]限定，另外一种是空间大小设置。

(2) 预设是没有任何限额设置的，可以使用 `hadoop fs -count -q pathname` 查看

```
hadoop fs -count -q /user/hadoop/
# 结果如下，为了能全部显示出来，把 tab 换成了空格，所以就这样啦，哈哈
# 下面有一个表格，对这几个字段进行详细的解释，请查看下面的表
none inf none inf 2 0 0 hdfs://namenode:9000/user/hadoop
```

序号	文字	字段名	解释
1	none	QUOTA	允许创建文件或文件夹的个数
2	inf	REMAINING_QUOTA	创建文件或文件夹个数剩余
3	none	SPACE_QUOTA	允许创建文件总共大小限制
4	inf	REMAINING_SPACE_QUOTA	剩余创建文件总共大小限制
5	2	DIR_COUNT	路径个数
6	0	FILE_COUNT	文件个数
7	0	CONTENT_SIZE	总共文件大小

[1]说白了就是允许你建多少个文件

计算公式，第一个是命名空间计算公式，第二个为空间计算公式

$$QUOTA - (DIR\_COUNT + FILE\_COUNT) = REMAINING\_QUOTA$$
$$SPACE\_QUOTA - CONTENT\_SIZE = REMAINING\_SPACE\_QUOTA$$

下面是一些配置命令的例子：

```
hadoop fs -count -q /user/hadoop # 查看配置限额

# 设置此路径允许创建文件和文件夹的最大数目
# 如果要写入的档案数已经超过设定值，会有错误信息 NSQuotaExceededException
hadoop dfsadmin -setQuota 10000 /user/hadoop/test/

# 查看上一步的设定
hadoop fs -count /user/hadoop/test/

# 设置空间最大允许大小，m,g,t 分别代表 MB,GB,TB
# 如果写入的档案数已经超过目前设定值，则会有：DSQuotaExceededException
hadoop dfsadmin -setSpaceQuota 1g /user/hadoop/test/

# 清除掉设定
hadoop dfsadmin -clrSpaceQuota /user/hadoop/test/
```

(3) 备注：路径改名限额设置依然生效，设定后如果档案量超过预设，依然会存在大小为 0 的这些上传文件，这点 hadoop 做的不咋地，这个可以试验下，或许新版本解决此问题，不得而知，我没有进行测试。还有其他 BUG，这个看你的 hadoop 版本了，不过到对使用影响不是很大。

9.3 权限管理详解