1. zookeeper引言

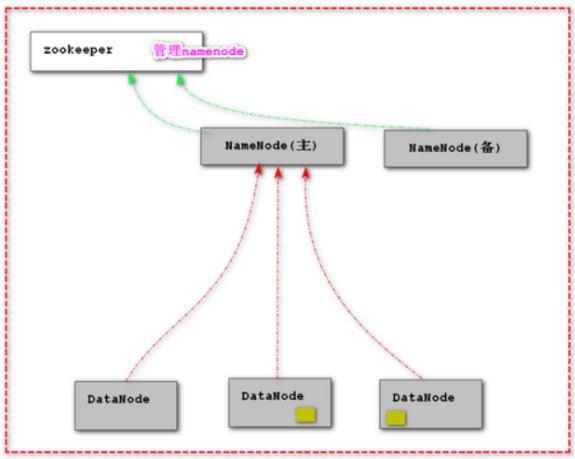
- 简介

> **Apache ZooKeeper**是Apache软件基金会的一个软件项目，大数据集群服务器的管理者协调者。
>
> 简言：ZK就是一个管理多个服务(集群分布式环境下)的`通知机制 Watcher`+`文件系统`
>
> ZNode 文件系统：保存少量，服务器相关的配置文件信息。
>
> Watcher 监听通知机制：注册监听服务器的上下线。

- 特点

> 1. zk集群中的数据内容，完全一致。
> 2. zk作为集群管理者，天生不存在单点问题。
> 3. zk的主机是动态选举出来的。

- 应用场景



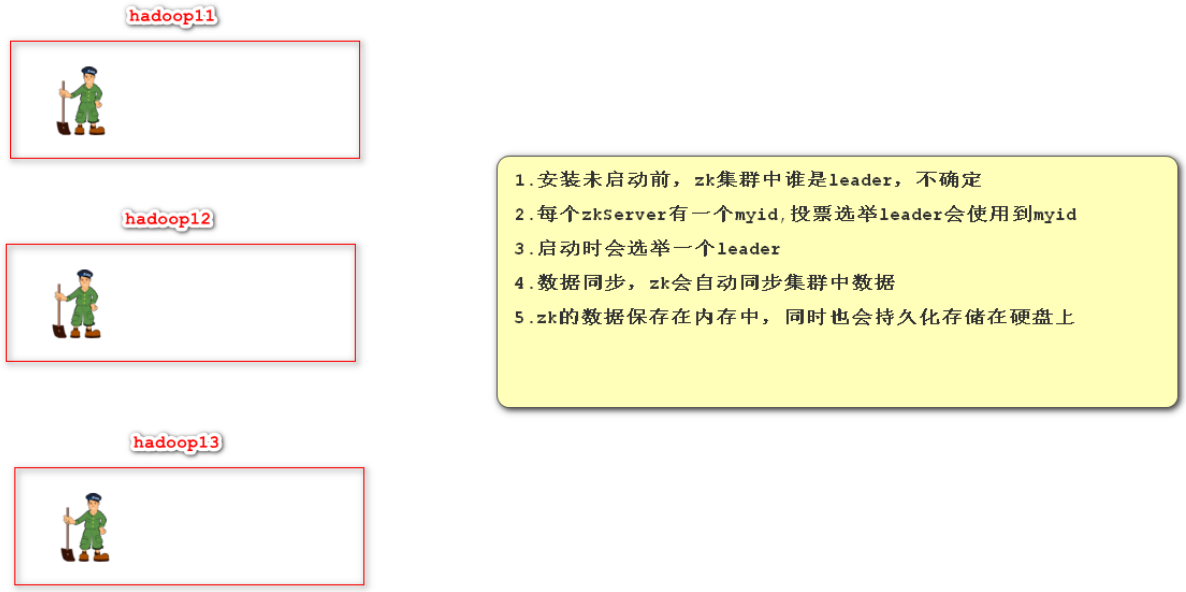
问题：
1. 如果datanode宕机：不会导致block丢失和服务稳定。
   原因：NameNode管理了dataNode
   - 监控datanode上线和下线。
   - 管理datanode内部block的描述信息。

2. 如果NameNode宕机：---单点故障问题。
   导致客户端无法操作HDFS，并且导致HDFS内部集群中所有节点文件存储功能都会丧失。
   解决：
   - 发现namenode宕机(下线)
   - 发现后，立刻 执行指令 将HDFS中namenode备机---转为主机，接管所有datanode管理。

zookeeper：
1. 监听通知机制：监控服务器上下线，在发现上下线后，还能执行一端指令。
2. 文件系统(特殊)：存储服务器配置信息，节点信息...(少量数据)
   例如：服务器ip、服务器状态、服务配置参数。

服务管理者---自身稳定性非常高

2. 集群安装



1.准备三台服务器

(hadoop11/hadoop12/hadoop13)

```
0. 设置ip
1. 安装jdk
2. 配置java环境变量
3. 关闭防火墙
4. 设置hostname
5. 设置hosts(3台彼此之间集群互通)
```

2.安装

```
# 1. 解压
[root@hadoop11 modules]# tar zxvf zookeeper-3.4.6.tar.gz -C /opt/installs/
# 2. 修改文件名
[root@hadoop11 installs]# mv zookeeper-3.4.6/ zookeeper3.4.6
# 3. 编辑环境变量配置文件
[root@hadoop11 zookeeper3.4.6]# vim /etc/profile


# ------------------下面是添加的内容------
# zookeeper
export PATH=$PATH:/opt/installs/zookeeper3.4.6/bin/


# 4. 重新加载profile配置
source /etc/profile
```

3.标记主机号

```
# 1. zk目录下新建一个data目录
  作为后续zk的数据存放位置
  [root@hadoop11 zookeeper3.4.6]# mkdir data
# 2. 在data下，新建一个myid文件。
  [root@hadoop11 zookeeper3.4.6]# cd data
# 3. 里面内容填写当前zk节点的编号
  [root@hadoop11 data]# echo 11 > myid
```

4.初始化配置文件

```
# 1. 拷贝zoo.cfg文件
  [root@hadoop11 conf]# cp zoo_sample.cfg zoo.cfg
# 2. 配置zoo.cfg
-------------以下是内容--------------
#Zookeeper使用的基本时间，服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个tickTime时间就会发送一个心跳，时间单位为毫秒。
tickTime=2000 #心跳时间周期(单位：毫秒)
#集群中的Follower跟随者服务器与Leader领导者服务器之间初始连接时能容忍的最多心跳数（tickTime的数量），用它来限定集群中的Zookeeper服务器连接到Leader的时限。
initLimit=10 #启动zk时候的时间延迟最大值(10倍心跳)
##Leader发送心跳包给集群中所有Follower，若Follower在syncLimit时间内没有响应，那么Leader就认为该follower已经挂掉了，单位：tickTime
syncLimit=5 # zk的主机和从机之间的通信访问的最大延迟(5倍心跳)
dataDir=/opt/installs/zookeeper3.4.6/data/ #zk的数据存储位置
clientPort=2181 # zk的客户端访问zk的端口号

# server.myid=zk的ip:2888:3888
server.11=hadoop11:2888:3888
server.12=hadoop12:2888:3888
server.13=hadoop13:2888:3888
# 2888(内部数据通信的端口)  #3888(选举投票使用的端口)
```

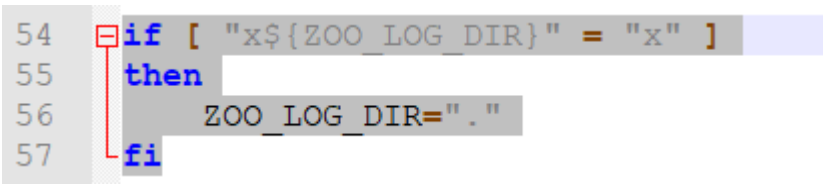
5.同步配置文件

```
# 1. 同步zookeeper的软件及其内部配置文件信息
  scp -r zookeeper3.4.6 root@hadoop12:/opt/installs/
  scp -r zookeeper3.4.6 root@hadoop13:/opt/installs/
# 2. 同步zookeeper的环境变量文件/etc/profile
  scp -r /etc/profile root@hadoop12:/etc
  scp -r /etc/profile root@hadoop13:/etc
# 3. 重新加载其他节点上的zk的环境变量
  source /etc/profile
# 4. 修改其他节点上的myid的zk编号。
```

6.zk服务器管理命令

```
# 1. 启动
    zkServer.sh start
    [root@hadoop11 data]# jps
    2610 QuorumPeerMain # zk节点的进程。
    3500 Jps
# 2. 查看状态
    zkServer.sh status

# 3. 停止
     zkServer.sh stop

# 4. 客户端
    zkCli.sh 登录本机的zk
    zkCli.sh -server ip:2181 登录指定ip的zk主机
```

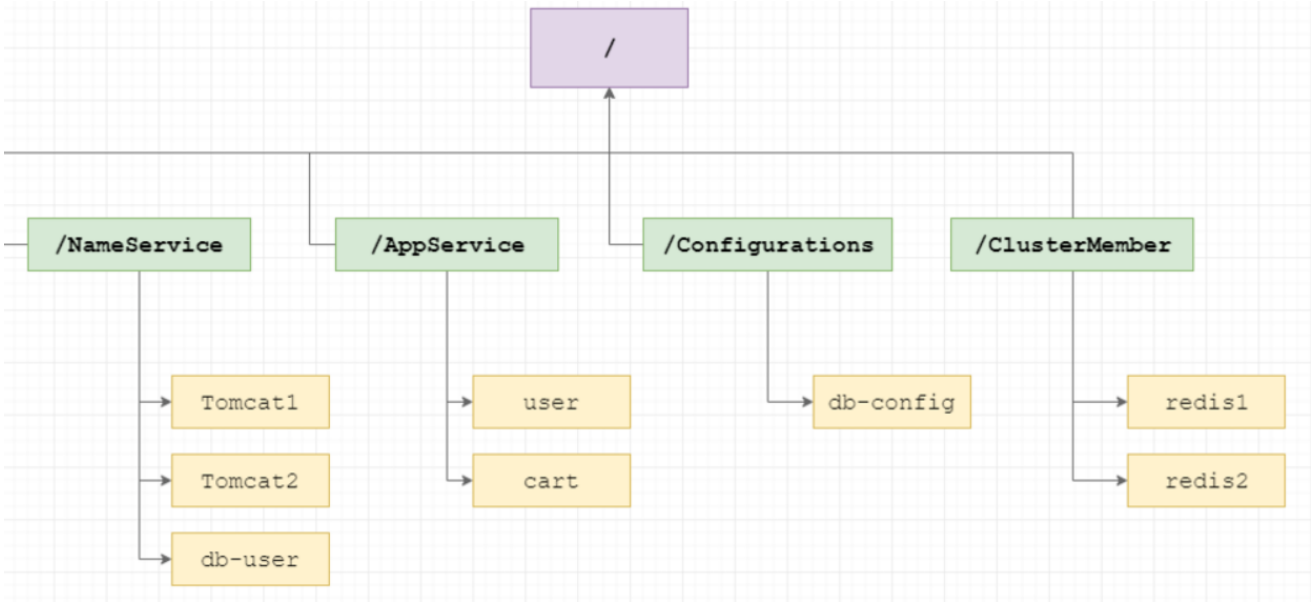
> **日志** zk启动异常，查看日志文件：zookeeper.out
>
> 默认位置： 启动zkServer的命令所在的目录，
>
> 可以通过conf/zkEnv.sh修改 ZOO_LOG_DIR = "/opt/installs/zookeeper3.4.6/logs"



3.zk客户端命令

ZNode节点

```
1. zk中的节点包含name-value。
2. zk中的节点可以有子节点。
3. zk中节点的结构是树状结构。
```



1. 客户端操作命令

```
# 1.客户端使用基本命令
  1. 进入客户端
    zkCli.sh
  2. 查看帮助命令
    [zk: localhost:2181(CONNECTED) 1] help
  3. 退出客户端
    [zk: localhost:2181(CONNECTED) 1] quit
```

2. znode管理命令

命令	含义
ls /	浏览某个节点下的子节点(的名字)
create /节点名 节点值	创建节点，并指定他的值。
get /节点名	查看节点的值
set /节点名 新值	修改节点的值
delete /节点名	删除某个节点
rmr /节点名	删除该节点，并递归删除内部所有节点。

```
# 1. 浏览某个节点下的子节点(的名字)
ls /
# 2. 创建节点，并指定他的值。
[zk: localhost:2181(CONNECTED) 8] create /baizhi baizhiinfo
Created /baizhi
# 3. 查看节点的值
```

```
[zk: localhost:2181(CONNECTED) 10] get /baizhi
baizhiinfo # 数据
cZxid = 0x200000005
ctime = Fri Apr 10 17:55:04 CST 2020 # 创建时间
mZxid = 0x200000005
mtime = Fri Apr 10 17:55:04 CST 2020
pZxid = 0x200000005
cversion = 0
dataVersion = 0 # 节点数据的更新次数【只要执行set就更新】
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 10 # 节点数据的字节,最大值1M
numChildren = 0 # 直接子节点的个数

# 4. 修改节点的值
[zk: localhost:2181(CONNECTED) 45] set /baizhi 新值
# 5. 删除节点
[zk: localhost:2181(CONNECTED) 53] delete /baizhi
# 6. 删除节点及其子节点
[zk: localhost:2181(CONNECTED) 53] rmr /baizhi
```

3.节点类型(面试)

 # 节点类型

zookeeper可以将节点设置不同的类型

1. 持久化节点
   节点只要不删除，会一直存在。
2. 顺序编号节点
   每次对相同节点，重复创建，会自动对znode名称进行编号
3. 临时节点
   客户端断开，则节点消失。

节点名称	中文	含义
`PERSISTENT`	持久化节点	客户端与zookeeper断开连接后，该节点依旧存在
`PERSISTENT_SEQUENTIAL`	持久化顺序编号节点	客户端与zookeeper断开连接后，该节点依旧存在，只是Zookeeper给该节点名称进行顺序编号
`EPHEMERAL`	临时节点	客户端与zookeeper断开连接后，该节点被删除
`EPHEMERAL_SEQUENTIAL`	临时顺序编号节点	客户端与zookeeper断开连接后，该节点被删除，只是Zookeeper给该节点名称进行顺序编号

命令	含义
create /节点 节点值	持久化节点
create -s /节点 节点值	持久化节点+顺序编号节点
create -e /节点 节点值	临时节点，客户端断开连接则失效。
create -s -e /节点 节点值	顺序编号节点+临时节点

4.Watcher监听器命令

> 1. 监听**节点值**得修改(set)和删除(delete)变化
> 2. 监听某个**节点及其子节点**的增加、删除。

命令	含义
get /节点1/节点2 watch	查看节点内容，并监听该值的变化(修改、失效等)
ls /节点 watch	查看某个节点下的所有节点信息，并监听下节点的变化(添加删除子节点)

4. java访问zk

> zookeeper官方提供java api,并不是特别好用，所以curator框架

1. Znode操作API

- 导入依赖和log4j配置文件

```
<!--zk的客户端( curator )-->
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>2.7.1</version>
</dependency>

<!--junit测试-->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
```

- 获得zk的客户端

```
// 1. 创建一个连接(自动重连)
RetryNTimes retry = new RetryNTimes(10,1000);// 重连10次，每次间隔1000秒

// 2. 创建一个客户端对象。
CuratorFramework curator = CuratorFrameworkFactory.newClient("hadoop11:2181", retry);
// 3. 启动客户端
curator.start();
```

- 创建节点

```
String s = curator.create().withMode(CreateMode.PERSISTENT)
                           .forPath("/a1", "测试".getBytes());
System.out.println(s);
```

- 读取节点值

```
byte[] bytes = curator.getData().forPath("/a1");
System.out.println(new String(bytes));
```

- 修改节点值

```
curator.setData().forPath("/a1","测试2".getBytes());
```

- 判断节点是否存在

```
Stat stat = client.checkExists().forPath("/a1");
如果节点存在，stat包含节点描述信息
如果节点不存在，stat是一个null
```

- 删除节点

```
curator.delete().forPath("/a1");
```

- 获得子节点

```
List<String> strings = curator.getChildren().forPath("/a1");
//遍历子节点
for (String node : strings) {
    byte[] bytes = curator.getData().forPath("/a1/" + node);
    System.out.println(new String(bytes));
}
```

2. Watcher操作API

- 监听节点变化

命令：get /a1 watch

对应代码：

```
// 1 创建节点监听客户端
final NodeCache nodeCache = new NodeCache(client, "/a1");
// 2 启动客户端监听器
nodeCache.start();
// 3 为客户端监听器，绑定监听事件函数
nodeCache.getListenable().addListener(new NodeCacheListener() {
    /**
       * 一旦节点值变化，调用函数
       * @throws Exception
       */
    public void nodeChanged() throws Exception {
        byte[] data = nodeCache.getCurrentData().getData();
        System.out.println(new String(data));
    }
});
// 5 程序停止，客户端消失，监听也就失效
Thread.sleep(Long.MAX_VALUE);
```

- 监听子节点变化

```
// 1 创建子节点监听器客户端
PathChildrenCache childCache = new PathChildrenCache(curator, "/a1", true);
// 2 启动监听器
childCache.start();
// 3 为监听器绑定注册事件函数
childCache.getListenable().addListener(new PathChildrenCacheListener() {
    public void childEvent(CuratorFramework client, PathChildrenCacheEvent event)
                                                                    throws Exception {
        System.out.println(event.getType());
        switch(event.getType()){
            case CHILD_ADDED:
                System.out.println("节点添加");
                System.out.println(event.getData().getPath()+":"+new
String(event.getData().getData()));
                break;
            case CHILD_UPDATED:
                System.out.println("节点更新");
                System.out.println(event.getData().getPath()+":"+new
String(event.getData().getData()));
                break;
            case CHILD_REMOVED:
                System.out.println("节点删除");
                System.out.println(event.getData().getPath()+":"+new
String(event.getData().getData()));
                break;
            default:
                System.out.println("其他");
```

```
                break;
            }
        }
    });
    // 5 客户端程序永不停止。
    Thread.sleep(Long.MAX_VALUE);
```

5. HadoopHA(高可用)

> HAHadoop(High Available Hadoop) 高可用Hadoop集群。

1.HDFS分布式集群的问题

```
1. NameNode单点故障
2. NameNode备机空闲
3. NameNode仍然存在少量数据丢失的问题
4. NameNode假死和双主问题
5. NameNode主备自动切换，客户端无法知晓入口地址
```

2.问题解决思路分析

```
# 1. NameNode单点故障
思路：提供NameNode备机
在两个NameNode机器上，各自提供一个zkclient。负责注册节点，并监控节点变化
方案：ZKFC--基于zookeeper实现的故障转移程序(基于zk实现的客户端程序)
```

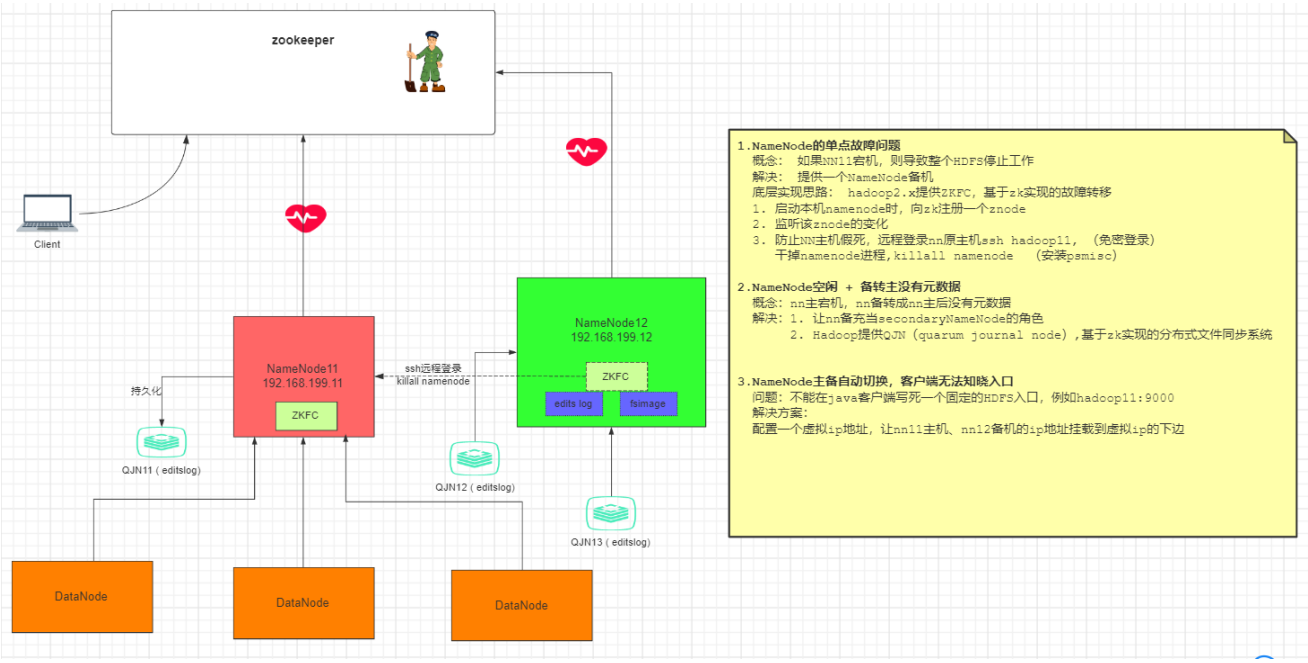
```
# 2. NameNode假死和双主问题
思路：
只要NN备机切换，无论NN主机是否死机，都强制杀死。
① 远程登录NN主机节点：ssh
② 执行杀死NameNode进程命令：killall namenode
方案： ZKFC自带该效果，当NN主机宕机，切换备机的同时，会远程登陆NN主机，使用psmisc的命令killall杀死NameNode。防止出现双主。
```

```
# 3. NameNode备机空闲
思路：承担SNN的职责
好处：
① NameNode备机资源不闲置，同时替换SNN

② NameNode备机转正，则可以直接从本地读取fsimage恢复数据，方便快捷。
```

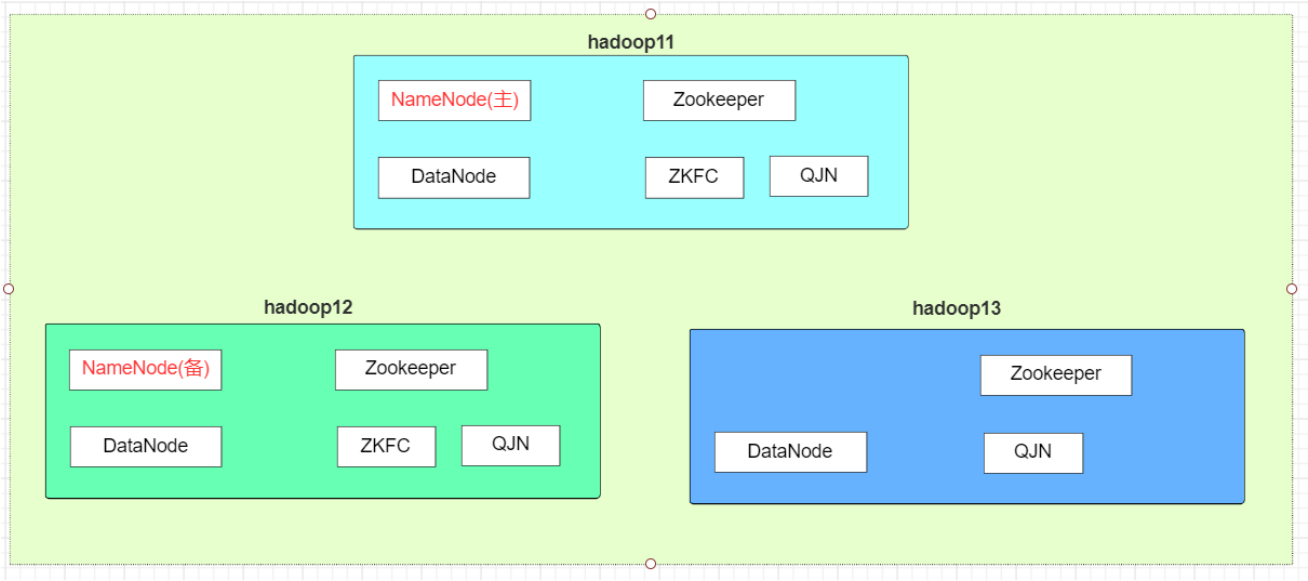
```
# 4. NameNode仍然存在少量数据丢失的问题
思路:
提供一个分布式存储空间，并可以数据同步共享，用来存储editslog文件。
方案:
  QJN：基于zookeeper实现的分布式存储系统，还有数据共享同步机制。
  QJN之间通过网络进行数据同步--RPC技术
好处:
NameNode备机节点，也存在一个QJN，存储了editslog，这边可以避免从NameNode主机远程拷贝editslog，提升ckeckpoint性能。
NameNode备机转正，除了恢复本地fsimage，还能恢复本地的editslog数据文件，恢复完整数据。
```

```
# 5. NameNode主备自动切换，客户端无法知晓入口地址
思路:
  ① 不能写死ip,将NN主机ip注册再Zk集群中，zk负责主备切换，并更新该节点的值。
  ② Java客户端通过读取zk节点中的节点之，动态读取HDFS的入口地址(NameNode现主机ip)
方案:
  HDFS的NameNode入口地址，配置虚拟ip。
```

3.架构图



4.HDFS-HA安装



1.配置NN主机和NN备机所在节点的免密登录

```
# 1. NameNode主节点执行
# 生成密钥(在namenode主和namenode备生成，将公钥发布其他节点)
[root@hadoop11 ~]# ssh-keygen
# 发送公钥(所有NameNode节点都要发送)
[root@hadoop11 ~]# ssh-copy-id hadoop11
[root@hadoop11 ~]# ssh-copy-id hadoop12
[root@hadoop11 ~]# ssh-copy-id hadoop13


# 2. NameNode备机节点执行
# 生成密钥(在namenode主和namenode备生成，将公钥发布其他节点)
[root@hadoop12 ~]# ssh-keygen
# 发送公钥(所有NameNode节点都要发送)
[root@hadoop12 ~]# ssh-copy-id hadoop11
[root@hadoop12 ~]# ssh-copy-id hadoop12
[root@hadoop12 ~]# ssh-copy-id hadoop13
```

2.安装psmisc

```
# ZKFC远程杀死假死NN使用的killall namenode命令属于该软件中的。
# 建议所有节点都安装psmisc
[root@hadoop11 ~]# yum install -y psmisc
[root@hadoop12 ~]# yum install -y psmisc
```

3.安装配置jdk

```
# 1. 解压jdk
  [root@hadoop11 modules]# tar zxvf jdk-8u221-linux-x64.tar.gz -C /opt/installs/

# 2. 改jdk的目录名

# 3. 配置profile环境变量
  export JAVA_HOME=/opt/installs/jdk1.8
  export PATH=$PATH:$JAVA_HOME/bin

# 4. 同步集群环境:
  1. 其他节点同步JDK的安装
  2. 同步profile配置文件,
  3. 并重新加载其他节点的JDK环境变量
```

4.安装zookeeper

```
# 1. 解压zk
  [root@hadoop11 modules]# tar zxvf zookeeper-3.4.6.tar.gz -C /opt/installs/

# 2. 改名

# 3. 配置zk的环境变量
  export PATH=$PATH:/opt/installs/zookeeper3.4.6/bin/

# 4. 新建data目录，并编写投票编号myid文件

# 5. 初始化zoo.cfg文件
  tickTime=2000
  initLimit=10
  syncLimit=5
  dataDir=/opt/installs/zookeeper3.4.6/data #data文件目录
  clientPort=2181
  server.11=192.168.199.11:2888:3888 # zk主机信息
  server.12=192.168.199.12:2888:3888 # zk主机信息
  server.13=192.168.199.13:2888:3888 # zk主机信息

# 6. 同步集群环境(需要安装zk的节点)
  1. 同步zk的软件包
  2. 同步zk的profile环境
  3. 其他节点重新加载profile
  4. 修改其他节点的myid

# 7. 验证启动效果，并关闭zk服务器
  zkServer.sh start
  jps
  zkServer.sh stop
```

5.HA-HDFS配置文件初始化

```
# 省略hadoop软件安装过程
1. 解压hadoop
2. 配置hadoop的环境变量
3. 重新加载profile文件。

全新的Hadoop集群，要清空data目录。
# 0. 清空data目录，全部节点都要做
[root@hadoop11 data]# rm -rf /opt/installs/hadoop2.9.2/data/*
[root@hadoop11 installs]# rm -rf /opt/installs/hadoop2.9.2/logs/*

# 1. 配置hadoop-env.sh
export JAVA_HOME=/opt/installs/jdk1.8/

# 2. 配置core-site.xml
<configuration>
    <!--hdfs入口，设置虚拟地址，具体地址后面配置-->
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hdfs-cluster</value>
    </property>
    <!--hdfs集群的文件位置-->
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/installs/hadoop2.9.2/data</value>
    </property>
    <!--hdfs要访问zookeeper集群-->
    <property>
        <name>ha.zookeeper.quorum</name>
        <value>hadoop11:2181,hadoop12:2181,hadoop13:2181</value>
    </property>
</configuration>

# 3. 配置hdfs-site.xml
<configuration>
    <!-- 副本数 -->
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
    <!-- 定义dhfs入口的命名服务 -->
    <property>
        <name>dfs.nameservices</name>
        <value>hdfs-cluster</value>
    </property>
    <!-- 定义hdfs入口的命名服务下虚拟ip-->
    <property>
        <name>dfs.ha.namenodes.hdfs-cluster</name>
        <value>nn1,nn2</value>
    </property>
    <!-- 虚拟ip地址1 RPC入口 -->
    <property>
        <name>dfs.namenode.rpc-address.hdfs-cluster.nn1</name>
        <value>hadoop11:9000</value>
```

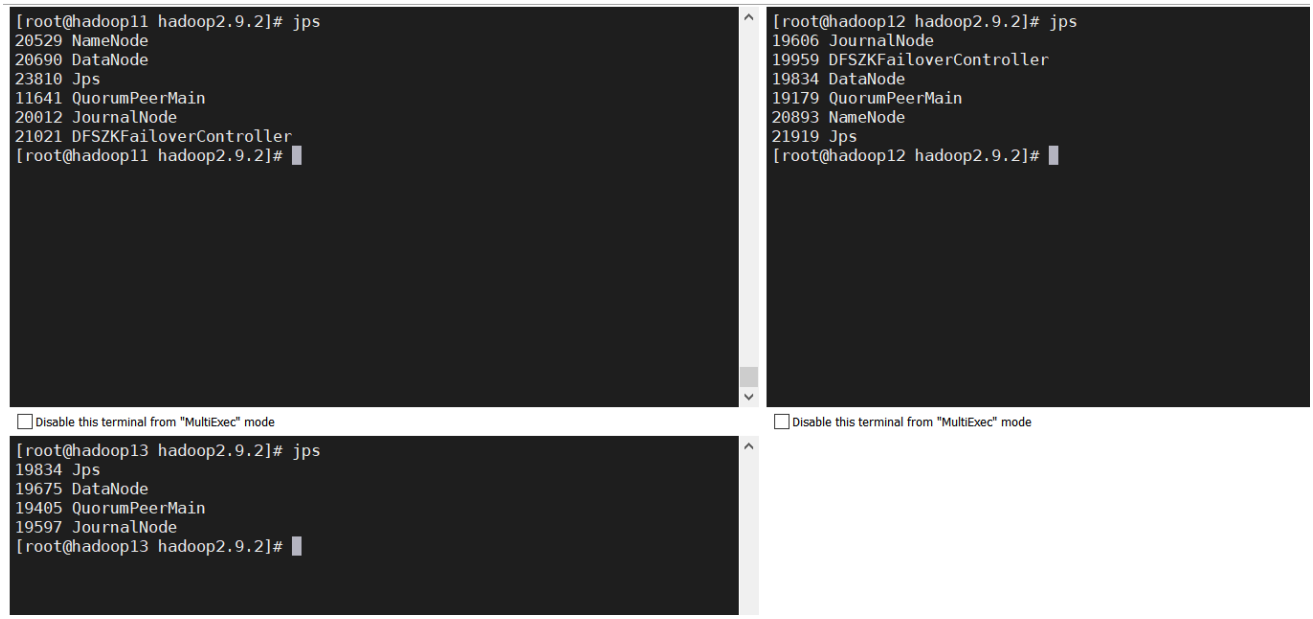
```
    </property>
    <!-- 虚拟ip地址1 HTTP入口 -->
    <property>
        <name>dfs.namenode.http-address.hdfs-cluster.nn1</name>
        <value>hadoop11:50070</value>
    </property>
    <!-- 虚拟ip地址2 PRC入口 -->
    <property>
        <name>dfs.namenode.rpc-address.hdfs-cluster.nn2</name>
        <value>hadoop12:9000</value>
    </property>
    <!-- 虚拟ip地址1 HTTP入口 -->
    <property>
        <name>dfs.namenode.http-address.hdfs-cluster.nn2</name>
        <value>hadoop12:50070</value>
    </property>

    <!-- 定义QJN在linux中保存文件磁盘目录 -->
    <property>
        <!-- Journal Edit Files 的存储目录:() -->
        <name>dfs.journalnode.edits.dir</name>
        <value>/opt/installs/journalnode/data/</value>
    </property>
    <!-- namenode要向zk的QJN写入editslog，所以要明确入口地址 -->
    <property>
        <name>dfs.namenode.shared.edits.dir</name>
        <value>qjournal://hadoop11:8485;hadoop12:8485;hadoop13:8485/hdfs-cluster</value>
    </property>


    <!-- 是否开启故障切换 -->
    <property>
        <name>dfs.ha.automatic-failover.enabled</name>
        <value>true</value>
    </property>


    <!-- 基于zookeeper的故障切换的代码类 -->
    <property>
        <name>dfs.client.failover.proxy.provider.hdfs-cluster</name>
       <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
    </property>

    <!-- 远程杀死namenode方式(防止namenode假死，导致双主出现)  -->
    <property>
        <name>dfs.ha.fencing.methods</name>
        <value>sshfence</value>
    </property>

    <!-- 指定私钥的文件目录，使用免密登录杀死NN进程 -->
    <property>
        <name>dfs.ha.fencing.ssh.private-key-files</name>

        <value>/root/.ssh/id_rsa</value>
```

```
    </property>

</configuration>

# 4. 配置slaves
  hadoop11
  hadoop12
  hadoop13

# 5. 同步以上配置到其他节点
  [root@hadoop11 etc]# scp -r hadoop/ root@hadoop12:/opt/installs/hadoop2.9.2/etc/
  [root@hadoop11 etc]# scp -r hadoop/ root@hadoop13:/opt/installs/hadoop2.9.2/etc/
```

6.HA-HDFS初次启动

```
启动顺序:
1. zk
2. 格式化zkfc
3. 启动QJN
4. 启动hdfs(zkfc qjn namenode datanode)
5. 声明hadoop12的namenode为备机
6. 启动nn备机

# 1. 启动zkserver集群
[root@hadoop11 etc]# zkServer.sh start
[root@hadoop12 etc]# zkServer.sh start
[root@hadoop13 etc]# zkServer.sh start

# 2. 初始化ZKFC在zk中的Znode信息【第一次启动需要做】
[root@hadoop11 etc]# hdfs zkfc -formatZK

# 3. 格式化hdfs的namenode主机(在namenode主节点)【第一次启动需要做】
1. 先启动journalnode(3台)(QJN将作为NameNode存储持久化文件的空间，要先启动才能格式化)
[root@hadoop11 etc]# hadoop-daemon.sh start journalnode
[root@hadoop12 etc]# hadoop-daemon.sh start journalnode
[root@hadoop13 etc]# hadoop-daemon.sh start journalnode

2. 格式化namenode主机(在NN主机节点)
[root@hadoop11 etc]# hdfs namenode -format

# 4. 格式化hdfs的namenode备机(namenode standby备节点)【第一次启动需要做】
1. 先启动主namenode主机(启动整个namenode集群)
  [root@hadoop11 etc]# start-dfs.sh
2. 在格式化namenode备机(第一次启动)
  [root@hadoop12 ~]# hdfs namenode -bootstrapStandby
3. 启动namenode备机
  [root@hadoop12 ~]# hadoop-daemon.sh start namenode

# 5. 以后HAHadoop启动和关闭，在NN主机节点执行命令。
     zk集群还是要单独启动和关闭的
[root@hadoop11 etc]# start-dfs.sh
```

```
    该命令会自动依次启动:
      NameNode主机
      NameNode备机
      DataNode有节点
      启动所有journal node
      启动所有zkfc

# 同理，在namenode主机上执行stop-dfs.sh
  会将上述所有进程都停止。
```




```
# 搭建失败如何修复
1. 看日志，改配置，同步修改其他节点配置。
2. 清空所有节点的/opt/installs/hadoop/data目录
3. 清空所有QJN所在节点的目录：/opt/installs/journalnode/
4. 按照首次启动HAHadoop步骤操作
```

7.java访问hadoopHA的编码

```
# 1. 导入HDFS操作以来
# 2. 拷贝HDFS的core-site.xml和hdfs-site.xml到resources目录下
# 3. 拷贝log4j.properties到 resources目录下
# 4. 下面是Java代码
```

```
# 配置文件失效，可能是Maven项目没有clean
public static void main(String[] args) throws IOException {
    //1. 初始化配置文件
    Configuration conf = new Configuration();
```

```
    conf.addResource("/core-site.xml");
    conf.addResource("/hdfs-site.xml");
    //2. 获得HDFS的客户端
    FileSystem fs = FileSystem.get(conf);
    //3. 操作HDFS的文件信息
    FileStatus[] files = fs.listStatus(new Path("/hdfs"));
    for (FileStatus file : files) {
        System.out.println(file);
    }
}
```

Yarn-HA安装

1.修该mapred-site.xml

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

<!-- 设置日志服务器的远程传输日志信息的端口和地址 -->
<property>
  <name>mapreduce.jobhistory.address</name>
  <value>hadoop11:10020</value>
</property>

<!-- 设置日志服务器的web访问的地址和端口 -->
<property>
  <name>mapreduce.jobhistory.webapp.address</name>
  <value>hadoop11:19888</value>
</property>
```

2.修改yarn-site.xml

```
<!--指定yarn上允许运行的分布式代码为mapreduce-->
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<!-- 开启日志聚合：将各个节点上的日志文件集中到HDFS中，便于管理 -->
<property>
  <name>yarn.log-aggregation-enable</name>
  <value>true</value>
</property>
<!-- 设置日志保存时间 -->
<property>
  <name>yarn.log-aggregation.retain-seconds</name>
```

```
  <value>106800</value>
</property>

<!--配置resourcemanager的HA-->
<property>
    <name>yarn.resourcemanager.ha.enabled</name>
    <value>true</value>
</property>
<!-- RM 集群标识 -->
<property>
    <name>yarn.resourcemanager.cluster-id</name>
    <value>yarn-cluster</value>
</property>
 <!-- RM 的逻辑 ID 列表 -->
<property>
    <name>yarn.resourcemanager.ha.rm-ids</name>
    <value>rm1,rm2</value>
</property>
<!-- RM1 的主机地址 -->
<property>
    <name>yarn.resourcemanager.hostname.rm1</name>
    <value>hadoop11</value>
</property>
<!-- RM1 的主机web管理界面地址 -->
<property>
    <name>yarn.resourcemanager.webapp.address.rm1</name>
    <value>hadoop11:8088</value>
</property>
<!-- RM2 的主机地址 -->
<property>
    <name>yarn.resourcemanager.hostname.rm2</name>
    <value>hadoop12</value>
</property>
<!-- RM2 的主机web管理界面地址 -->
<property>
    <name>yarn.resourcemanager.webapp.address.rm2</name>
    <value>hadoop12:8088</value>
</property>
<!-- ZooKeeper 集群的地址 -->
<property>
    <name>yarn.resourcemanager.zk-address</name>
    <value>hadoop11:2181,hadoop12:2181,hadoop13:2181</value>
</property>
<!-- 启用自动恢复 -->
<property>
    <name>yarn.resourcemanager.recovery.enabled</name>
    <value>true</value>
</property>
<!-- 用于yarn故障转移持久化zk的类 -->
<property>
    <name>yarn.resourcemanager.store.class</name>
    <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>
```

3.同步配置文件

```
# 同步mapred-site.xml
[root@hadoop11 hadoop]# scp  mapred-site.xml root@hadoop12:/opt/installs/hadoop2.9.2/etc/hadoop/
[root@hadoop11 hadoop]# scp  mapred-site.xml root@hadoop13:/opt/installs/hadoop2.9.2/etc/hadoop/

# 同步yarn-site.xml
[root@hadoop11 hadoop]# scp  yarn-site.xml root@hadoop12:/opt/installs/hadoop2.9.2/etc/hadoop/
[root@hadoop11 hadoop]# scp  yarn-site.xml root@hadoop13:/opt/installs/hadoop2.9.2/etc/hadoop/
```

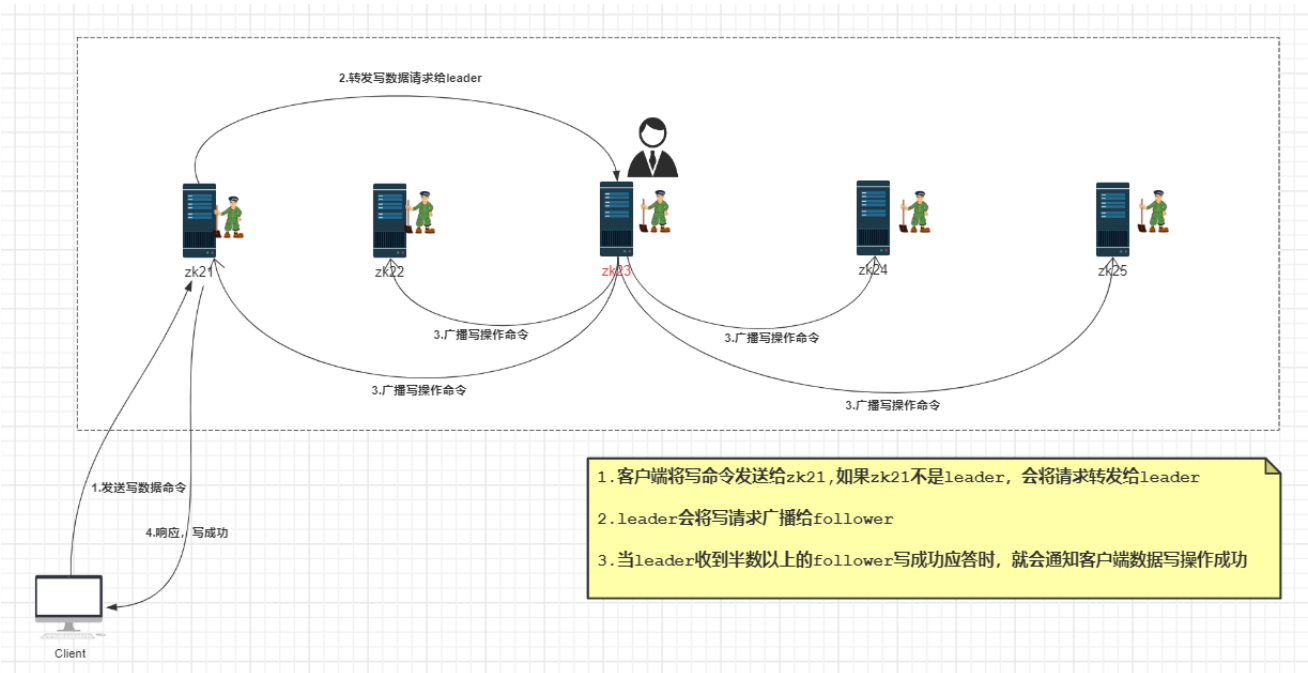
4.启动yarn集群

```
#1. rm主节点执行(启动后需要等一会NodeManager才能注册到RM中)
[root@hadoop11 hadoop]# start-yarn.sh

#2. rm备机节点执行启动备机
[root@hadoop12 hadoop]# yarn-daemon.sh start resourcemanager
```

6. Zookeeper相关概念和原理

1. zk数据的读写流程(数据一致性)

- 写数据流程



- 读数据流程

> 相比写数据流程，读数据流程就简单得多；因为每台server中数据一致性都一样，所以随便访问哪台server读数据就行；

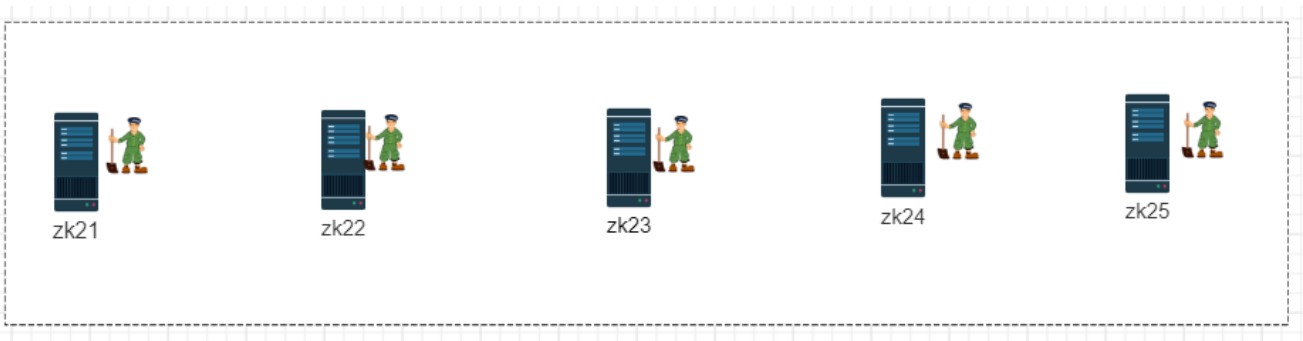
2. zk的启动投票选主流程 [面试]

（1）半数机制：集群中半数以上机器存活，集群可用。所以Zookeeper适合安装奇数台服务器。

（2）Zookeeper虽然在配置文件中并没有指定Master和Slave。但是，Zookeeper工作时，是有一个节点为Leader，其他则为Follower，Leader是通过内部的选举机制临时产生的。

（3）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21-25，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，发起一次选举。服务器1投自己一票。此时服务器1票数一票，不够半数以上（3票），选举无法完成，服务器1状态保持为LOOKING；
（2）服务器2启动，再发起一次选举。服务器1和2分别投自己一票并交换选票信息：此时服务器1发现服务器2的ID比自己目前投票推举的（服务器1）大，更改选票为推举服务器2。此时服务器1票数0票，服务器2票数2票，没有半数以上结果，选举无法完成，服务器1，2状态保持LOOKING
（3）服务器3启动，发起一次选举。此时服务器1和2都会更改选票为服务器3。此次投票结果：服务器1为0票，服务器2为0票，服务器3为3票。此时服务器3的票数已经超过半数，服务器3当选Leader。服务器1，2更改状态为FOLLOWING，服务器3更改状态为LEADING；
（4）服务器4启动，发起一次选举。此时服务器1，2，3已经不是LOOKING状态，不会更改选票信息。交换选票信息结果：服务器3为3票，服务器4为1票。此时服务器4服从多数，更改选票信息为服务器3，并更改状态为FOLLOWING；
（5）服务器5启动，同4一样当小弟。