尚硅谷大数据技术之 Hadoop HA

第 1 章 Hadoop HA 高可用

1.1 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（Single Points Of Failure）。

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 Active/Standby 两个 NameNodes 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。

1.2 HDFS-HA 工作机制

通过多 NameNode 消除单点故障

1.3 HDFS-HA 手动故障转移

1.3.1 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；

两个 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.必须保证多 NameNode 之间能够 ssh 无密码登录

3.隔离（Fence），即同一时刻仅仅有一个 NameNode 对外提供服务

1.3.2 配置 HDFS-HA 集群

1.在 opt 目录下创建一个 ha 文件夹

```
[atguigu@hadoop202 opt]$ mkdir ha
```

2.将/opt/module/下的 hadoop-3.1.3 拷贝到/opt/ha 目录下

```
[atguigu@hadoop202 opt] cp -r hadoop-3.1.3 ha
```

3.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/module/ha/hadoop-
3.1.3/data/tmp</value>
      </property>
   <!-- 声明 journalnode 服务器存储目录-->
  <property>
      <name>dfs.journalnode.edits.dir</name>
      <value>file://${hadoop.tmp.dir}/jn</value>
  </property>
</configuration>
```

4.配置 hdfs-site.xml

```
<configuration>
  <!-- 完全分布式集群名称 -->
  <property>
      <name>dfs.nameservices</name>
      <value>mycluster</value>
  </property>
  <!-- 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>

  <!-- 集群中 NameNode 节点都有哪些 -->
  <property>
      <name>dfs.ha.namenodes.mycluster</name>
      <value>nn1,nn2,nn3</value>
  </property>

  <!-- nn1 的 RPC 通信地址 -->
  <property>
      <name>dfs.namenode.rpc-address.mycluster.nn1</name>
      <value>hadoop102:9000</value>
  </property>

  <!-- nn2 的 RPC 通信地址 -->
  <property>
      <name>dfs.namenode.rpc-address.mycluster.nn2</name>
      <value>hadoop103:9000</value>
  </property>
```

```
  <!-- nn3 的 RPC 通信地址 -->
  <property>
      <name>dfs.namenode.rpc-address.mycluster.nn3</name>
      <value>hadoop104:9000</value>
  </property>


  <!-- nn1 的 http 通信地址 -->
  <property>
      <name>dfs.namenode.http-address.mycluster.nn1</name>
      <value>hadoop102:9870</value>
  </property>

  <!-- nn2 的 http 通信地址 -->
  <property>
      <name>dfs.namenode.http-address.mycluster.nn2</name>
      <value>hadoop103:9870</value>
  </property>
  <!-- nn3 的 http 通信地址 -->
  <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;hadoop10
4:8485/mycluster</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 无秘钥登录-->
  <property>
      <name>dfs.ha.fencing.ssh.private-key-files</name>
      <value>/home/atguigu/.ssh/id_rsa</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.Configu
redFailoverProxyProvider</value>
  </property>
</configuration>
```

5.拷贝配置好的 hadoop 环境到其他节点

1.3.3 启动 HDFS-HA 集群

0. 修改 HADOOP 的环境变量 ， 指向 HA 的集群

```
sudo vim /etc/profile.d/my_env.sh
HADOOP_HOME=/opt/module/ha/hadoop-3.1.3
```

1. 在各个 JournalNode 节点上，输入以下命令启动 journalnode 服务

   先删除 hadoop102　hadoop103 hadoop104 机器中的/tmp 目录下的内容

```
hdfs --daemon start journalnode
```

2. 在[nn1]上，对其进行格式化，并启动

```
hdfs namenode -format
hdfs --daemon start namenode
```

3. 在[nn2]和[nn3]上，同步 nn1 的元数据信息

```
hdfs namenode -bootstrapStandby
```

4. 启动[nn2] 和 [nn3]

```
hdfs --daemon start namenode
```

5. 查看 web 页面显示，102 103 104 都为 standby 状态
6. 启动所有 datanode

```
hdfs --daemon start datanode
```

7. 将[nn1]切换为 Active

```
hdfs haadmin -transitionToActive nn1
```

8. 是否 Active

```
hdfs haadmin -getServiceState nn1
```

9. kill 掉 Active 的 NameNode，进行手动故障转移

1.4 配置 HDFS-HA 自动故障转移

1.4.1 工作要点

前面学习了使用命令 hdfs haadmin 手动进行故障转移，在该模式下，即使现役 NameNode 已经失效，系统也不会自动从现役 NameNode 转移到待机 NameNode，下面学习如何配置部署 HA 自动进行故障转移。自动故障转移为 HDFS 部署增加了两个新组件：ZooKeeper 和 ZKFailoverController（ZKFC）进程。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 中维护了一个持久会话，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。故障转移进程与前面描述的手动故障转移相似，首先如果必要保护之前的现役 NameNode，然后本地 NameNode 转换为 Active 状态。

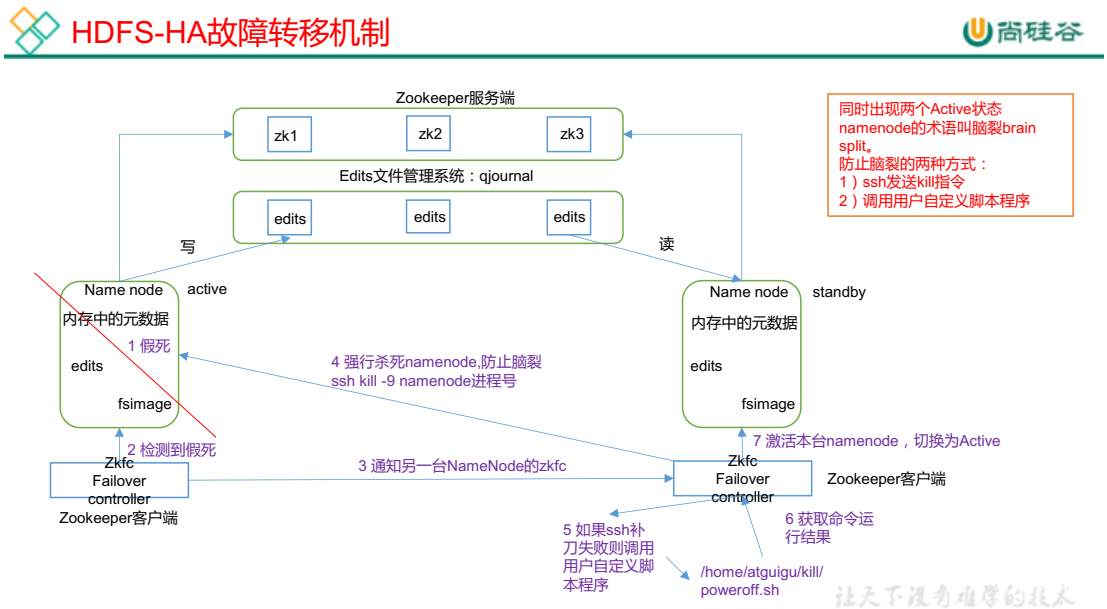

图 1-1 HDFS-HA 故障转移机制

1.4.2 规划集群

表 1-1

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

1.4.3 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

```
[atguigu@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

```
mkdir -p zkData
```

（3）重命名/opt/module/zookeeper-3.5.7/conf 这个目录下的 zoo_sample.cfg 为 zoo.cfg

```
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

```
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

```
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 hadoop103　hadoop104

```
[atguigu@hadoop102 module] xsync  zookeeper-3.5.7
```

并分别修改 myid 文件中内容为 3、4

（4）分别启动 zookeeper

```
[atguigu @hadoop102 zookeeper-3.5.7]# bin/zkServer.sh start
[atguigu @hadoop103 zookeeper-3.5.7]# bin/zkServer.sh start
[atguigu @hadoop104 zookeeper-3.5.7]# bin/zkServer.sh start
```

（5）查看状态

```
[atguigu @hadoop102 zookeeper-3.5.7]# bin/zkServer.sh status
JMX enabled by default
Using              config:              /opt/module/zookeeper-
3.4.10/bin/../conf/zoo.cfg
Mode: follower
[atguigu @hadoop103 zookeeper-3.5.7]# bin/zkServer.sh status
JMX enabled by default
Using              config:              /opt/module/zookeeper-
3.4.10/bin/../conf/zoo.cfg
Mode: leader
[atguigu @hadoop104 zookeeper-3.5.7]# bin/zkServer.sh status
JMX enabled by default
Using              config:              /opt/module/zookeeper-
3.4.10/bin/../conf/zoo.cfg
Mode: follower
```

1.4.4 配置 HDFS-HA 自动故障转移

1. 具体配置

（1）在 hdfs-site.xml 中增加

```
<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>
```

（2）在 core-site.xml 文件中增加

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

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

2. 启动

（1）关闭所有 HDFS 服务：

```
stop-dfs.sh
```

（2）启动 Zookeeper 集群，在每个 Zookeeper 节点执行

```
zkServer.sh start
```

（3）初始化 HA 在 Zookeeper 中状态：

```
hdfs zkfc -formatZK
```

（4）启动 HDFS 服务：

```
start-dfs.sh
```

3. web 端查看，一个 Active 的 NameNode， 两个 Standby 的 NameNode

4．验证

（1）将 Active NameNode 进程 kill,实现自动故障转移

```
kill -9 namenode 的进程 id
```

1.5 YARN-HA 配置

1.5.1 YARN-HA 工作机制

1. YARN-HA 工作机制，如图 1-2 所示

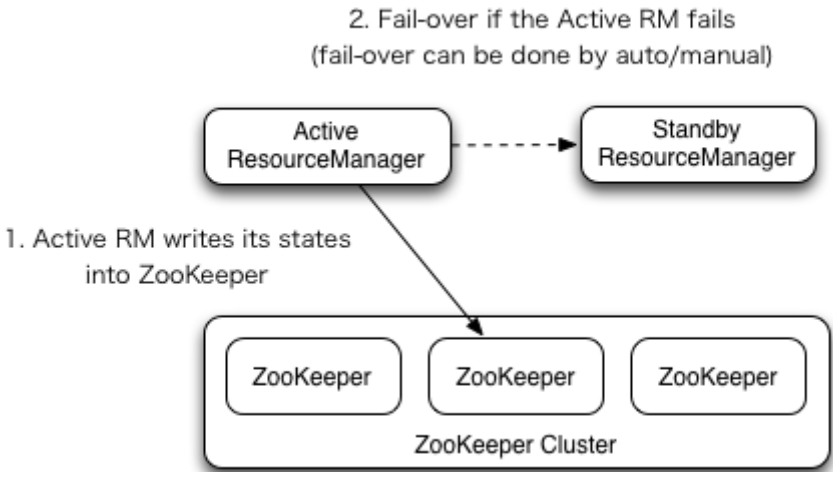

图 1-2 YARN-HA 工作机制

1.5.2 配置 YARN-HA 集群

1.规划集群

表 1-2

hadoop102	hadoop103	hadoop104
NameNode	NameNode	NameNode

JournalNode	JournalNode	JournalNode
ZKFC	ZKFC	ZKFC
DataNode	DataNode	DataNode
ZK	ZK	ZK
ResourceManager	ResourceManager	ResourceManager
NodeManager	NodeManager	NodeManager

2. 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 HA resourcemanager 的地址-->
    <property>
        <name>yarn.resourcemanager.cluster-id</name>
        <value>cluster-yarn1</value>
    </property>
     <!-- 指定 RM 的逻辑列表 -->
    <property>
        <name>yarn.resourcemanager.ha.rm-ids</name>
        <value>rm1,rm2,rm3</value>
    </property>

    <!-- 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>
      <!--  ========== rm1 配置==========  -->
      <!-- 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 配置============  -->

    <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>

    <!--  ========== rm3 配置============  -->

    <property>
       <name>yarn.resourcemanager.hostname.rm3</name>
       <value>hadoop104</value>
    </property>

    <property>

<name>yarn.resourcemanager.webapp.address.rm3</name>
          <value>hadoop104:8088</value>
    </property>
       <property>
          <name>yarn.resourcemanager.address.rm3</name>
          <value>hadoop104:8032</value>
       </property>
```

```
        <property>

<name>yarn.resourcemanager.scheduler.address.rm3</name>
            <value>hadoop104:8030</value>
        </property>

    <property>
            <name>yarn.resourcemanager.resource-
tracker.address.rm3</name>
            <value>hadoop104:8031</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.recov
ery.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,HADOO
P_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HA
DOOP_MAPRED_HOME</value>
    </property>

</configuration>
```

（2）同步配置文件到其他节点

```
[atguigu@hadoop202 hadoop]$ xsync yarn-site.xml
```

3. 启动 HDFS

4. 启动 YARN

（1）在 hadoop102 中执行：

```
start-yarn.sh
```

（2）查看服务状态

```
yarn rmadmin -getServiceState rm1
```

（3）web 端查看 YARN 的状态

1.6 HDFS Federation 架构设计

1. NameNode 架构的局限性

（1）Namespace（命名空间）的限制

由于 NameNode 在内存中存储所有的元数据（metadata），因此单个 NameNode 所能存储的对象（文件+块）数目受到 NameNode 所在 JVM 的 heap size 的限制。50G 的 heap 能够存储 20 亿（200million）个对象，这 20 亿个对象支持 4000 个 DataNode，12PB 的存储（假设文件平均大小为 40MB）。随着数据的飞速增长，存储的需求也随之增长。单个 DataNode 从 4T 增长到 36T，集群的尺寸增长到 8000 个 DataNode。存储的需求从 12PB 增长到大于 100PB。

（2）隔离问题

由于 HDFS 仅有一个 NameNode，无法隔离各个程序，因此 HDFS 上的一个实验程序就很有可能影响整个 HDFS 上运行的程序。

（3）性能的瓶颈

由于是单个 NameNode 的 HDFS 架构，因此整个 HDFS 文件系统的吞吐量受限于单个 NameNode 的吞吐量。

2. HDFS Federation 架构设计，如图 1-3 所示

能不能有多个 NameNode

表 1-3

NameNode	NameNode	NameNode
元数据	元数据	元数据
Log	machine	电商数据/话单数据

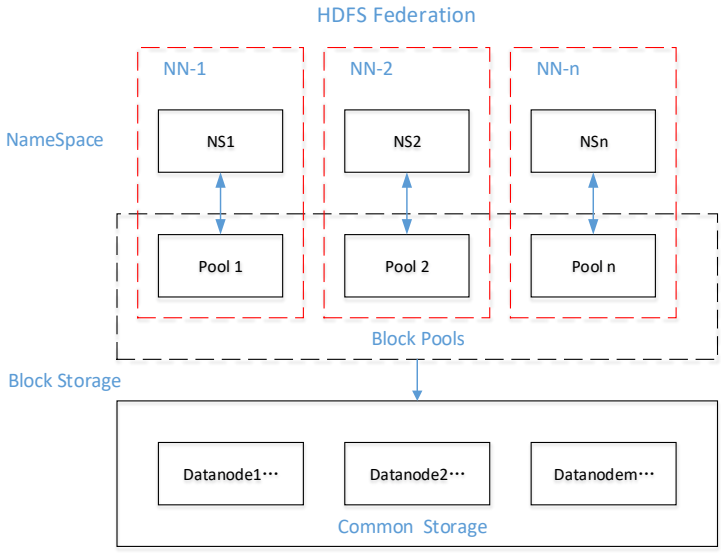

图 1-3　HDFS Federation 架构设计

3. HDFS Federation 应用思考

不同应用可以使用不同 NameNode 进行数据管理

图片业务、爬虫业务、日志审计业务

Hadoop 生态系统中，不同的框架使用不同的 NameNode 进行管理 NameSpace。（隔离性）