Tedu.cn
达内教育

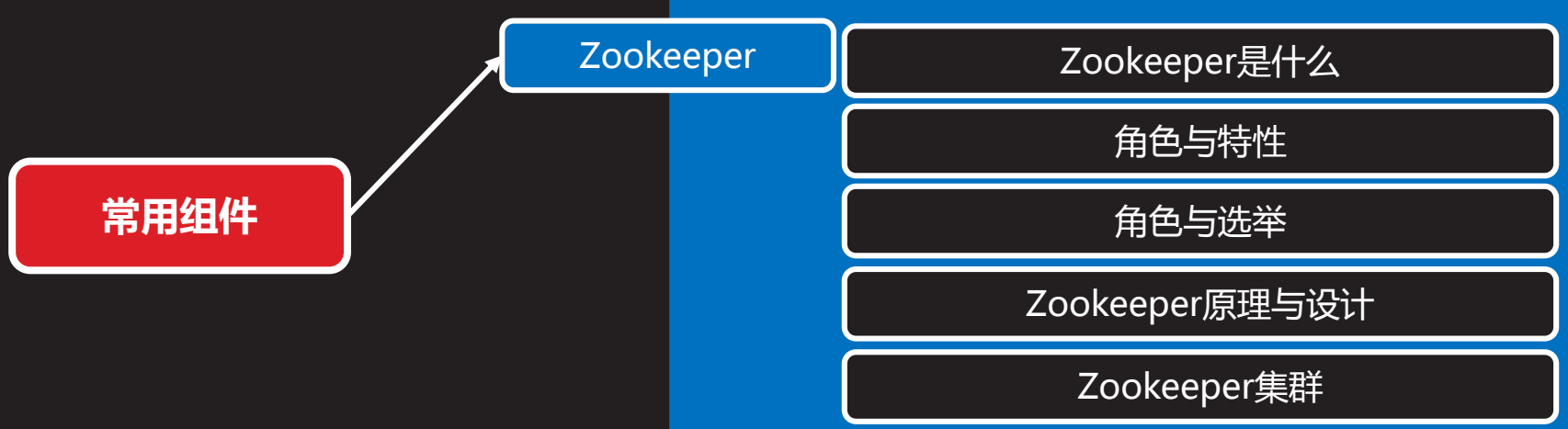
大型架构及配置技术

NSD ARCHITECTURE DAY07

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	常用组件
	10:30 ~ 11:20	Kafka集群
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	Hadoop高可用
	15:00 ~ 15:50	
	16:10 ~ 17:10	
	17:20 ~ 18:00	总结和答疑

常用组件


常用组件
Zookeeper
Zookeeper是什么
角色与特性
角色与选举
Zookeeper原理与设计
Zookeeper集群

Zookeeper

Zookeeper是什么

- Zookeeper是什么
  - Zookeeper是一个开源的分布式应用程序协调服务
- Zookeeper能做什么
  - Zookeeper是用来保证数据在集群间的事务一致性

知识讲解

Zookeeper是什么（续1）

- Zookeeper应用场景
  - 集群分布式锁
  - 集群统一命名服务
  - 分布式协调服务

知识讲解

角色与特性

- Zookeeper角色与特性
  - Leader：接受所有Follower的提案请求并统一协调发起提案的投票，负责与所有的Follower进行内部数据交换
  - Follower：直接为客户端服务并参与提案的投票，同时与Leader进行数据交换
  - Observer：直接为客户端服务但并不参与提案的投票，同时也与Leader进行数据交换

知识讲解

角色与选举

- Zookeeper角色与选举
  - 服务在启动的时候是没有角色的（LOOKING）
  - 角色是通过选举产生的
  - 选举产生一个Leader，剩下的是Follower
- 选举Leader原则
  - 集群中超过半数机器投票选择Leader
  - 假如集群中拥有n台服务器，那么Leader必须得到n/2+1台服务器的投票

知识讲解

角色与选举（续1）

- Zookeeper角色与选举
  - 如果Leader死亡，重新选举Leader
  - 如果死亡的机器数量达到一半，则集群挂掉
  - 如果无法得到足够的投票数量，就重新发起投票，如果参与投票的机器不足n/2+1，则集群停止工作
  - 如果Follower死亡过多，剩余机器不足n/2+1，则集群也会停止工作
  - Observer不计算在投票总设备数量里面

知识讲解

Zookeeper原理与设计

- Zookeeper可伸缩扩展性原理与设计
  - Leader所有写相关操作
  - Follower读操作与响应Leader提议
  - 在Observer出现以前，Zookeeper的伸缩性由Follower来实现，我们可以通过添加Follower节点的数量来保证Zookeeper服务的读性能，但是随着Follower节点数量的增加，Zookeeper服务的写性能受到了影响

知识讲解

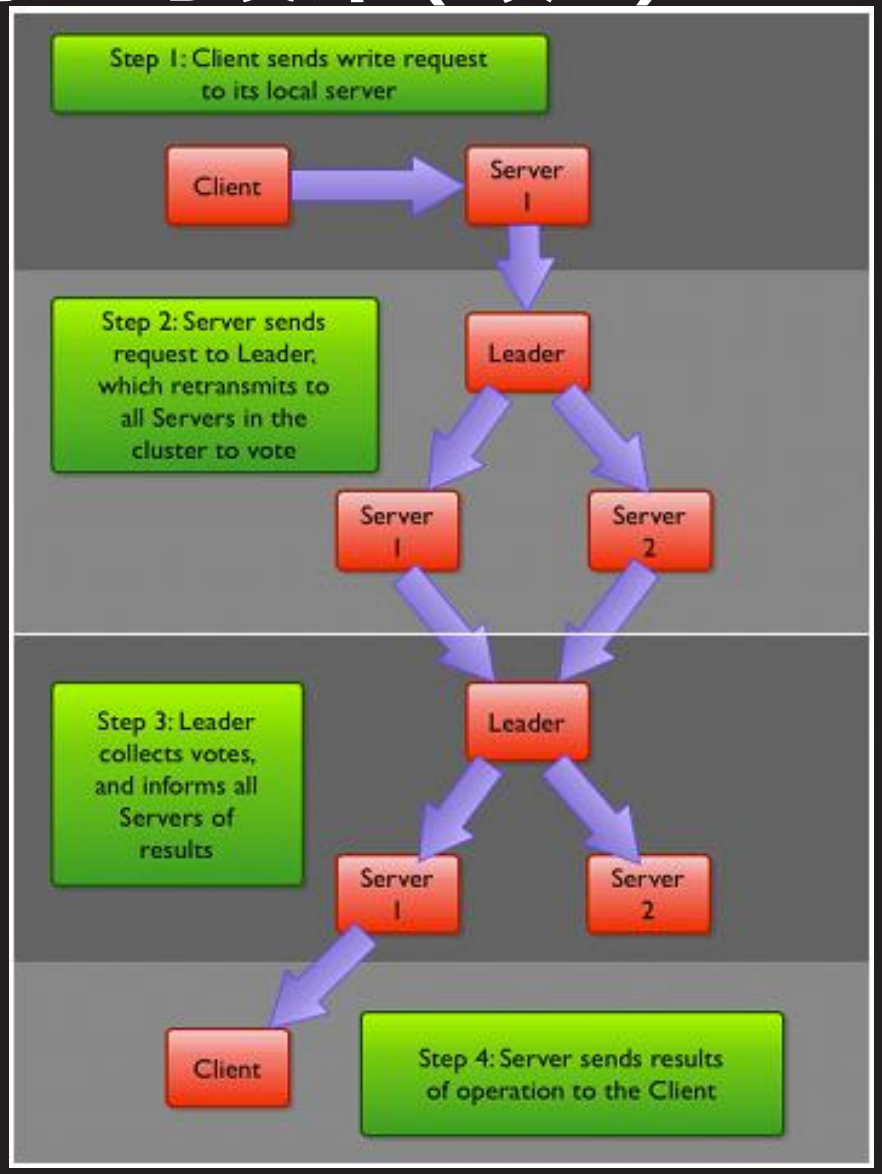
Zookeeper原理与设计（续1）

- Zookeeper可伸缩扩展性原理与设计
  - 客户端提交一个请求，若是读请求，则由每台Server的本地副本数据库直接响应。若是写请求，需要通过一致性协议（Zab）来处理
  - Zab协议规定：来自Client的所有写请求都要转发给ZK服务中唯一的Leader，由Leader根据该请求发起一个Proposal。然后其他的Server对该Proposal进行Vote。之后Leader对Vote进行收集，当Vote数量过半时Leader会向所有的Server发送一个通知消息。最后当Client所连接的Server收到该消息时，会把该操作更新到内存中并对Client的写请求做出回应

知识讲解

Zookeeper原理与设计（续2）

知识讲解



Zookeeper原理与设计（续3）

- 续上页
  - ZooKeeper在上述协议中实际扮演了两个职能。一方面从客户端接受连接与操作请求，另一方面对操作结果进行投票。这两个职能在Zookeeper集群扩展的时候彼此制约
  - 从Zab协议对写请求的处理过程中可以发现，增加Follower的数量，则增加了协议投票过程的压力。因为Leader节点必须等待集群中过半Server响应投票，是节点的增加使得部分计算机运行较慢，从而拖慢整个投票过程的可能性也随之提高，随着集群变大，写操作也会随之下降

知识讲解

Zookeeper原理与设计（续4）

- 续上页
  - 所以，我们不得不在增加Client数量的期望和我们希望保持较好吞吐性能的期望间进行权衡。要打破这一耦合关系，我们引入了不参与投票的服务器Observer。Observer可以接受客户端的连接，并将写请求转发给Leader节点。但Leader节点不会要求Observer参加投票，仅仅在上述第3步那样，和其他服务节点一起得到投票结果

Zookeeper原理与设计（续5）

- 续上页
  - Observer的扩展，给Zookeeper的可伸缩性带来了全新的景象。加入很多Observer节点，无须担心严重影响写吞吐量。但并非是无懈可击，因为协议中的通知阶段，仍然与服务器的数量呈线性关系。但是这里的串行开销非常低。因此，我们可以认为在通知服务器阶段的开销不会成为瓶颈
  - Observer提升读性能的可伸缩性
  - Observer提供了广域网能力

知识讲解

Zookeeper集群

知识讲解

- Zookeeper集群的安装配置
  - 配置文件改名zoo.cfg

```
# mv   zoo_sample.cfg    zoo.cfg
```

  - zoo.cfg文件最后添加如下内容

```
server.1=node1:2888:3888
server.2=node2:2888:3888
server.3=node3:2888:3888
server.4=nn01:2888:3888:observer
```

Zookeeper集群（续1）

知识讲解

- zoo.cfg集群的安装配置
  - 创建datadir指定的目录

    ```
    # mkdir  /tmp/zookeeper
    ```

  - 在目录下创建id对应主机名的myid文件
- 关于myid文件
  - myid文件中只有一个数字
  - 注意：请确保每个server的myid文件中id数字不同
  - server.id中的id与myid中的id必须一致
  - id的范围是1~255

Zookeeper集群（续2）

- Zookeeper集群的安装配置
  - 启动集群，查看验证（在所有集群节点执行）

    # /usr/local/zookeeper/bin/zkServer.sh start
  - 查看角色

    # /usr/local/zookeeper/bin/zkServer.sh status
  - Zookeeper管理文档

    http://zookeeper.apache.org/doc/r3.4.10/zookeeperAdmin.html

知识讲解

案例1：Zookeeper安装

1. 搭建Zookeeper集群并查看各服务器的角色
2. 停止Leader并查看各服务器的角色

课堂练习

Kafka集群

Kafka

什么是Kafka

- Kafka是什么
  - Kafka是由LinkedIn开发的一个分布式的消息系统
  - Kafka是使用Scala编写
  - Kafka是一种消息中间件
- 为什么要使用Kafka
  - 解耦、冗余、提高扩展性、缓冲
  - 保证顺序，灵活，削峰填谷
  - 异步通信

知识讲解

Kafka角色

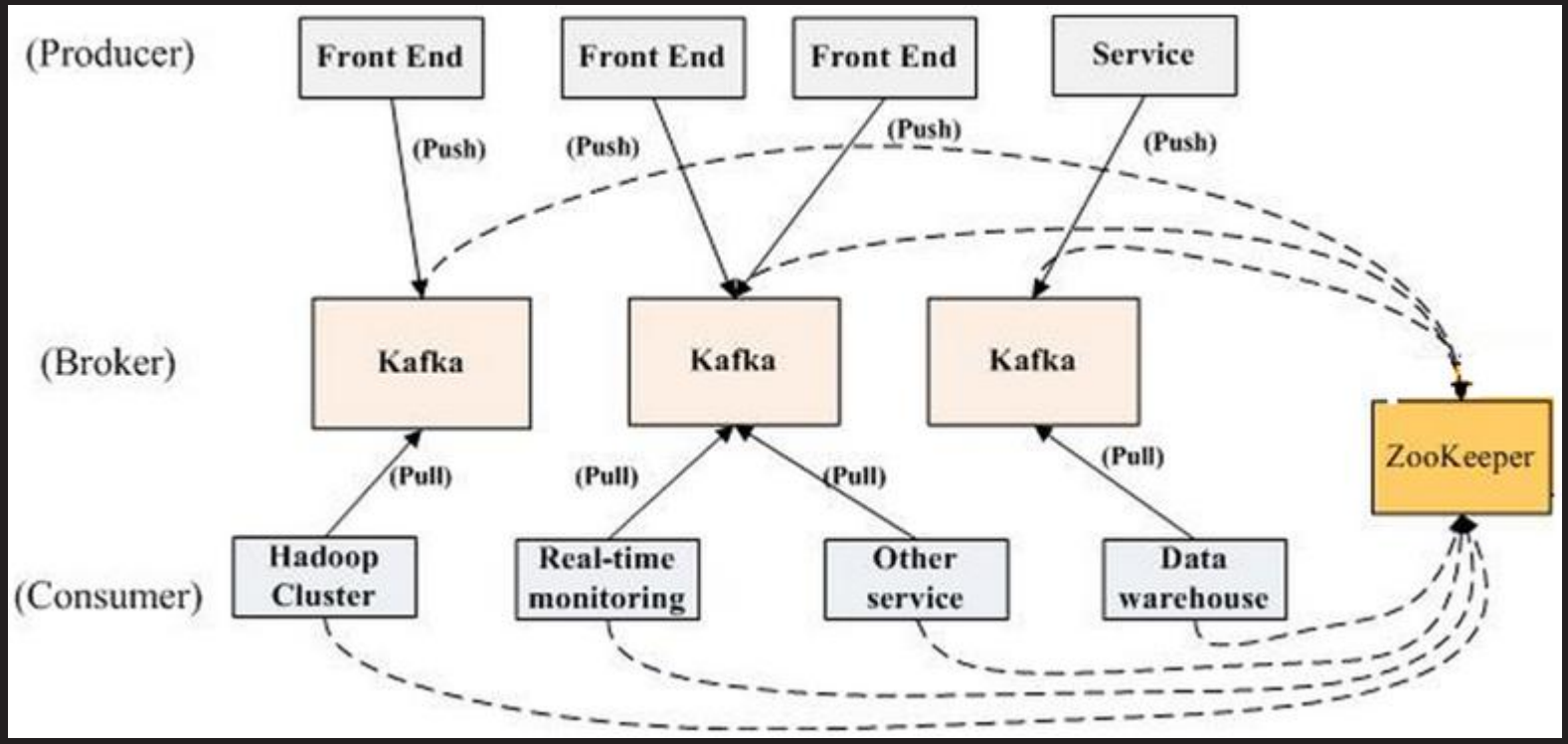
- Kafka角色与集群结构
  - producer：生产者，负责发布消息
  - consumer：消费者，负责读取处理消息
  - topic：消息的类别
  - Parition：每个Topic包含一个或多个Partition
  - Broker：Kafka集群包含一个或多个服务器

- Kafka通过Zookeeper管理集群配置，选举Leader

知识讲解

Kafka集群安装与配置

- Kafka角色与集群结构



Kafka集群安装与配置（续1）

- Kafka集群的安装配置
  - Kafka集群的安装配置依赖Zookeeper，搭建Kafka集群之前，请先创建好一个可用的Zookeeper集群
  - 安装OpenJDK运行环境
  - 同步Kafka拷贝到所有集群主机
  - 修改配置文件
  - 启动与验证

知识讲解

Kafka集群安装与配置（续2）

知识讲解

- Kafka集群的安装配置
- server.properties
  - broker.id
  - 每台服务器的broker.id都不能相同
- zookeeper.connect
  - zookeeper集群地址，不用都列出，写一部分即可

Kafka集群安装与配置（续3）

- Kafka集群的安装配置
  - 在所有主机启动服务

```
# ./bin/kafka-server-start.sh -daemon config/server.properties
```

- 验证
  - jps命令应该能看到Kafka模块
  - netstat应该能看到9092在监听

知识讲解

Kafka集群安装与配置（续4）

知识讲解

- 集群验证与消息发布
  - 创建一个 topic

```
# ./bin/kafka-topics.sh --create --partitions 2 --replication-factor 2 \
--zookeeper localhost:2181 --topic mymsg
```

  - 生产者

```
# ./bin/kafka-console-producer.sh \
--broker-list  localhost:9092 --topic mymsg
```

  - 消费者

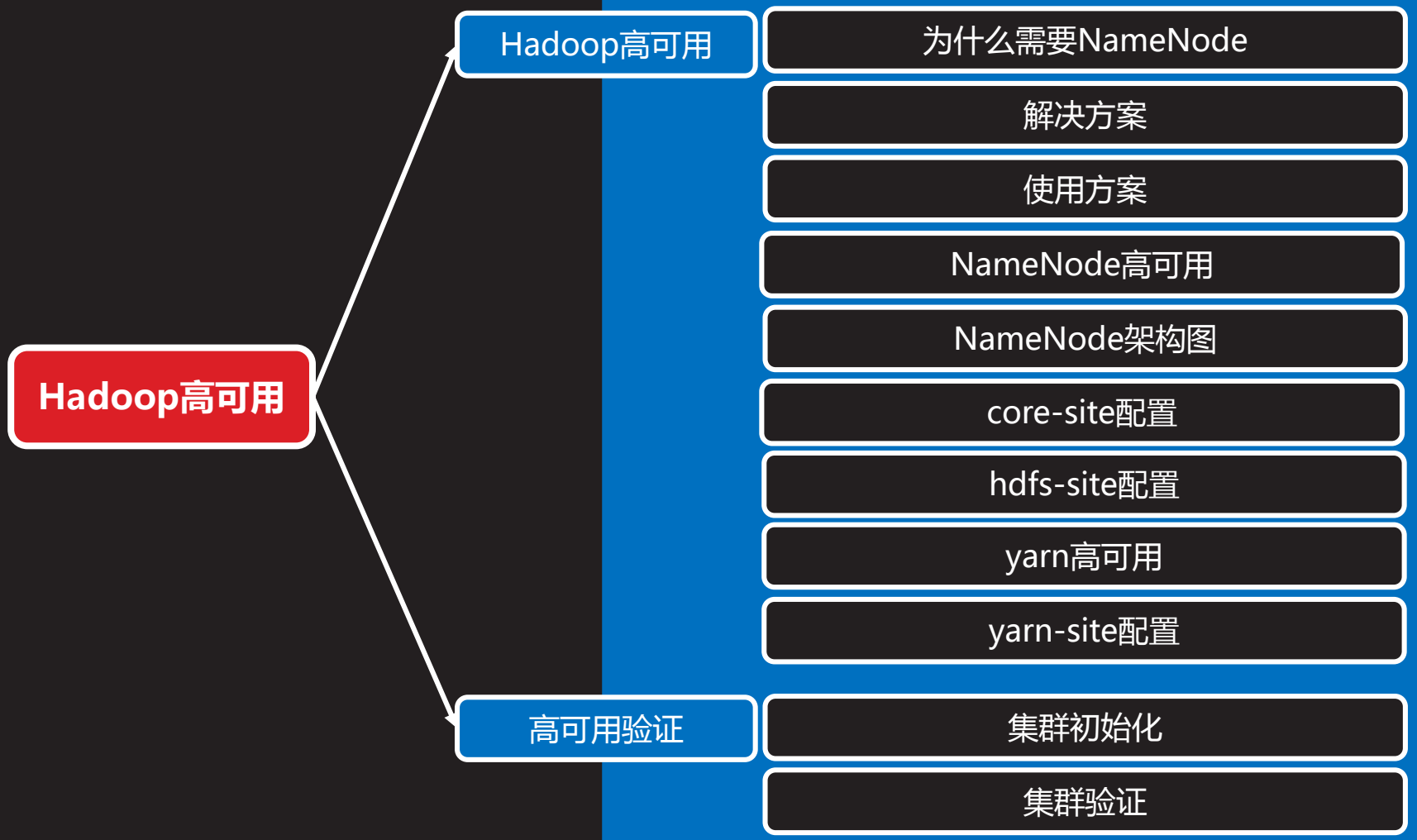
```
# ./bin/kafka-console-consumer.sh \
--bootstrap-server localhost:9092 --topic mymsg
```

案例2：Kafka集群实验

1. 利用Zookeeper搭建一个Kafka集群
2. 创建一个topic
3. 模拟生产者发布消息
4. 模拟消费者接收消息

课堂练习

Hadoop高可用


Hadoop高可用
Hadoop高可用
为什么需要NameNode
解决方案
使用方案
NameNode高可用
NameNode架构图
core-site配置
hdfs-site配置
yarn高可用
yarn-site配置
高可用验证
集群初始化
集群验证

Hadoop高可用

为什么需要NameNode

- 原因
  - NameNode是HDFS的核心配置，HDFS又是Hadoop核心组件，NameNode在Hadoop集群中至关重要
  - NameNode宕机，将导致集群不可用，如果NameNode数据丢失将导致整个集群的数据丢失，而NameNode的数据的更新又比较频繁，实现NameNode高可用势在必行

知识讲解

解决方案

- 官方提供了两种解决方案
  - HDFS with NFS
  - HDFS with QJM
- 两种方案异同

NFS	QJM
NN	NN
ZK	ZK
ZKFailoverController	ZKFailoverController
NFS	JournalNode

知识讲解

解决方案（续1）

- HA方案对比
  - 都能实现热备
  - 都是一个Active NN和一个Standby NN
  - 都使用Zookeeper和ZKFC来实现自动失效恢复
  - 失效切换都使用Fencin配置的方法来Active NN
  - NFS数据共享变更方案把数据存储在共享存储里，我们还需要考虑NFS的高可用设计
  - QJM不需要共享存储，但需要让每一个DN都知道两个NN的位置，并把块信息和心跳包发送给Active和Standby这两个NN

使用方案

- 使用原因（QJM）
  - 解决NameNode单点故障问题
  - Hadoop给出了HDFS的高可用HA方案：HDFS通常由两个NameNode组成，一个处于Active状态，另一个处于Standby状态。Active NameNode对外提供服务，比如处理来自客户端的RPC请求，而Standby NameNode则不对外提供服务，仅同步Active NameNode的状态，以便能够在它失败时进行切换

知识讲解

使用方案（续1）

- 典型的HA集群
  - NameNode会被配置在两台独立的机器上，在任何时候，一个NameNode处于活动状态，而另一个NameNode则处于备份状态
  - 活动状态的NameNode会响应集群中所有的客户端，备份状态的NameNode只是作为一个副本，保证在必要的时候提供一个快速的转移

知识讲解

NameNode高可用

- NameNode高可用架构
  - 为了让Standby Node与Active Node保持同步，这两个Node都与一组称为JNS的互相独立的进程保持通信（Journal Nodes）。当Active Node更新了namespace，它将记录修改日志发送给JNS的多数派。Standby Node将会从JNS中读取这些edits，并持续关注它们对日志的变更
  - Standby Node将日志变更应用在自己的namespace中，当Failover发生时，Standby将会在提升自己为Active之前，确保能够从JNS中读取所有的edits，即在Failover发生之前Standy持有的namespace与Active保持完全同步

知识讲解

NameNode高可用（续1）

- NameNode高可用架构 续……
  - NameNode更新很频繁，为了保持主备数据的一致性，为了支持快速Failover，Standby Node持有集群中blocks的最新位置是非常必要的。为了达到这一目的，DataNodes上需要同时配置这两个Namenode的地址，同时和它们都建立心跳连接，并把block位置发送给它们

知识讲解

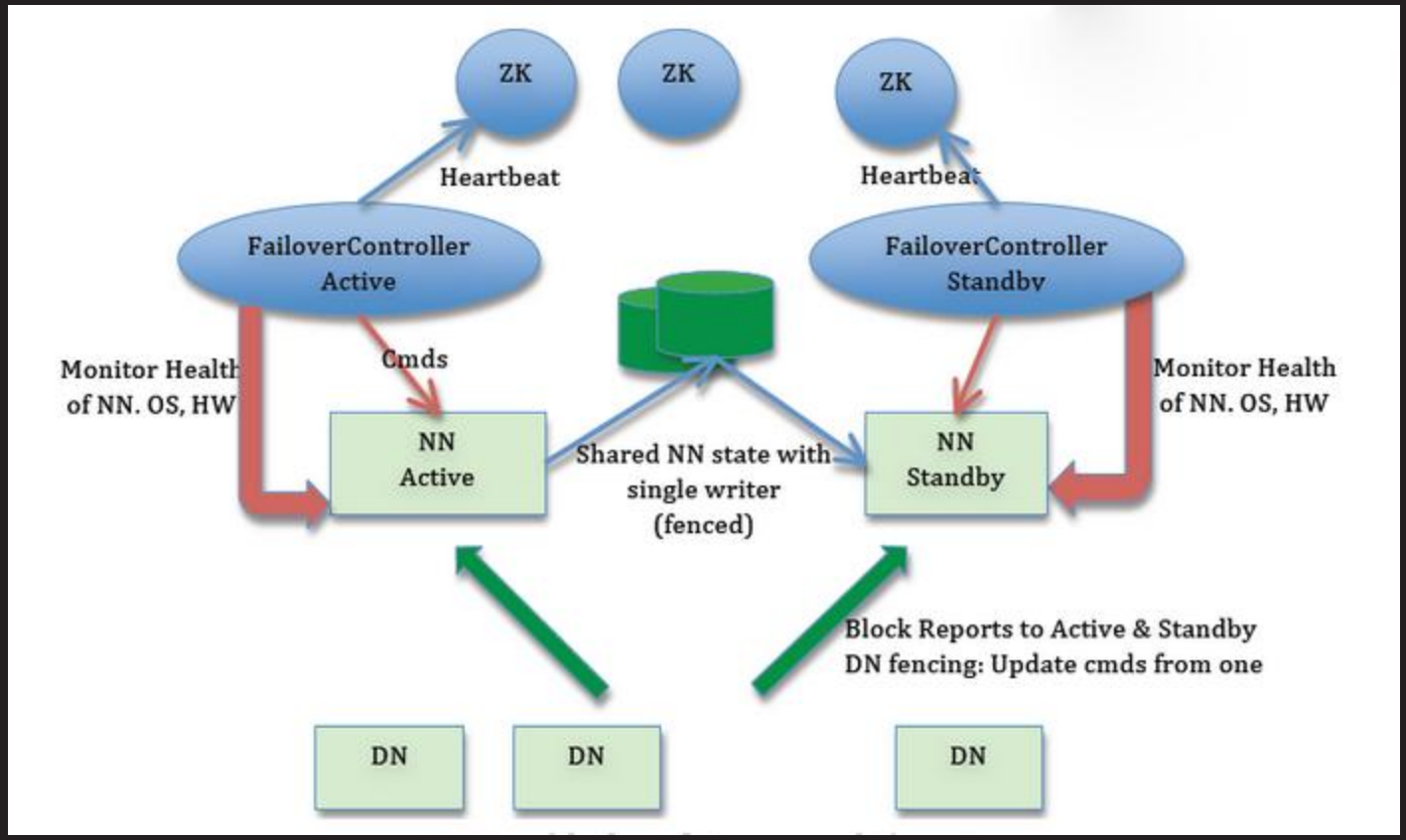
NameNode高可用（续2）

- NameNode高可用架构 续.....
  - 任何时刻，只能有一个Active NameNode，否则会导致集群操作混乱，两个NameNode将会有两种不同的数据状态，可能会导致数据丢失或状态异常，这种情况通常称为"split-brain"（脑裂，三节点通讯阻断，即集群中不同的DataNode看到了不同的Active NameNodes）
  - 对于JNS而言，任何时候只允许一个NameNode作为writer；在Failover期间，原来的Standby Node将会接管Active的所有职能，并负责向JNS写入日志记录，这种机制阻止了其他NameNode处于Active状态的问题

知识讲解

NameNode架构图

- NameNode高可用架构图



知识讲解

NameNode架构图（续1）

- 系统规划

主机	角色	软件
192.168.1.61	NameNode1	Hadoop
192.168.1.65	NameNode2	Hadoop
192.168.1.62 node1	DataNode journalNode Zookeeper	HDFS Zookeeper
192.168.1.63 node2	DataNode journalNode Zookeeper	HDFS Zookeeper
192.168.1.64 node3	DataNode journalNode Zookeeper	HDFS Zookeeper

知识讲解

core-site配置

- core-site.xml文件

```
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://mycluster</value>
</property>
<property>
    <name>hadoop.tmp.dir</name>
    <value>/var/hadoop</value>
</property>
<property>
    <name>ha.zookeeper.quorum</name>
    <value>node1:2181,node2:2181,node3:2181</value>
</property>
```

知识讲解

hdfs-site配置

- hdfs-site.xml文件

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

  - SecondaryNameNode在高可用里没有用，这里把它关闭
  - NameNode在后面定义

知识讲解

hdfs-site配置（续1）

- hdfs-site.xml文件
  - <!-- 指定hdfs的nameservices名称为mycluster -->

```
<property>
    <name>dfs.nameservices</name>
    <value>mycluster</value>
</property>
```

  - 指定集群的两个NaneNode的名称分别为nn1,nn2

```
<property>
    <name>dfs.ha.namenodes.mycluster</name>
    <value>nn1,nn2</value>
</property>
```

知识讲解

hdfs-site配置（续2）

- hdfs-site.xml文件
  - 配置nn1,nn2的rpc通信端口

```
<property>
    <name>dfs.namenode.rpc-
address.mycluster.nn1</name>
    <value>nn01:8020</value>
</property>
<property>
    <name>dfs.namenode.rpc-
address.mycluster.nn2</name>
    <value>nn02:8020</value>
</property>
```

知识讲解

hdfs-site配置（续3）

- hdfs-site.xml文件
  - 配置nn1,nn2的http通信端口

```
<property>
    <name>dfs.namenode.http-address.mycluster.nn1</name>
    <value>nn01:50070</value>
</property>
<property>
    <name>dfs.namenode.http-address.mycluster.nn2</name>
    <value>nn02:50070</value>
</property>
```

知识讲解

hdfs-site配置（续4）

- hdfs-site.xml文件
  - 指定NameNode元数据存储在journalnode中的路径

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

<value>qjournal://node1:8485;node2:8485;node3:8485/my
cluster</value>
</property>
```

  - 指定journalnode日志文件存储的路径

```
<property>
    <name>dfs.journalnode.edits.dir</name>
    <value>/var/hadoop/journal</value>
</property>
```

知识讲解

hdfs-site配置（续5）

- hdfs-site.xml文件
  - 指定HDFS客户端连接Active NameNode的java类

```
<property>

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

<value>org.apache.hadoop.hdfs.server.namenode.ha.Confi
guredFailoverProxyProvider</value>
</property>
```

知识讲解

hdfs-site配置（续6）

- hdfs-site.xml文件
  - 配置隔离机制为SSH

```
<property>
    <name>dfs.ha.fencing.methods</name>
    <value>sshfence</value>
</property>
```

  - 指定密钥的位置

```
<property>
    <name>dfs.ha.fencing.ssh.private-key-files</name>
    <value>/root/.ssh/id_rsa</value>
</property>
```

知识讲解

hdfs-site配置（续7）

- hdfs-site.xml文件
  - 开启自动故障转移

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

知识讲解

yarn高可用

- ResourceManager高可用
  - RM的高可用原理与NN一样，需要依赖ZK来实现，这里配置文件的关键部分，感兴趣的同学可以自己学习和测试
  - yarn.resourcemanager.hostname
  - 同理因为使用集群模式，该选项应该关闭

知识讲解

yarn-site配置

- yarn-site.xml配置

```
<property>
        <name>yarn.resourcemanager.ha.enabled</name>
        <value>true</value>
</property>

<property>
        <name>yarn.resourcemanager.ha.rm-ids</name>
        <value>rm1,rm2</value>
</property>
```

知识讲解

yarn-site配置（续1）

- yarn-site.xml配置

```
<property>

<name>yarn.resourcemanager.recovery.enabled</name>
    <value>true</value>
</property>

<property>
    <name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>
</property>
```

知识讲解

yarn-site配置（续2）

- yarn-site.xml配置

```
<property>
        <name>yarn.resourcemanager.zk-address</name>
        <value>node1:2181,node2:2181,node3:2181</value>
</property>

<property>
         <name>yarn.resourcemanager.cluster-id</name>
         <value>yarn-ha</value>
</property>
```

知识讲解

yarn-site配置（续3）

- yarn-site.xml配置

```
<property>

<name>yarn.resourcemanager.hostname.rm1</name>
        <value>nn01</value>
</property>

<property>

<name>yarn.resourcemanager.hostname.rm2</name>
        <value>nn02</value>
</property>
```

知识讲解

案例3：Hadoop高可用

1. 配置Hadoop的高可用
2. 修改配置文件

课堂练习

Tedu.cn
达内教育

高可用验证

集群初始化

知识讲解

- 初始化
  - ALL: 所有机器
  - nodeX： node1　node2　node3

  - ALL: 同步配置到所有集群机器
  - NN1: 初始化ZK集群

    # ./bin/hdfs zkfc -formatZK
  - nodeX: 启动journalnode服务

    # ./sbin/hadoop-daemon.sh start journalnode

集群初始化（续1）

- 初始化
  - NN1: 格式化

    ```
    # ./bin/hdfs  namenode  -format
    ```
  - NN2: 数据同步到本地/var/hadoop/dfs

    ```
    # rsync  -aSH nn01:/var/hadoop/dfs  /var/hadoop/
    ```
  - NN1: 初始化JNS

    ```
    # ./bin/hdfs namenode -initializeSharedEdits
    ```
  - nodeX: 停止journalnode服务

    ```
    # ./sbin/hadoop-daemon.sh stop journalnode
    ```

知识讲解

集群初始化（续2）

- 启动集群
  - NN1: 启动hdfs

    # ./sbin/start-dfs.sh
  - NN1: 启动yarn

    # ./sbin/start-yarn.sh
  - NN2: 启动热备ResourceManager

    # ./sbin/yarn-daemon.sh start resourcemanager

知识讲解

集群验证

- 查看集群状态
  - 获取NameNode状态

```
# ./bin/hdfs haadmin -getServiceState nn1
# ./bin/hdfs haadmin -getServiceState nn2
```

  - 获取ResourceManager状态

```
# ./bin/yarn rmadmin -getServiceState rm1
# ./bin/yarn rmadmin -getServiceState rm2
```

知识讲解

集群验证（续1）

知识讲解

- 查看集群状态
  - 获取节点信息

```
# ./bin/hdfs dfsadmin -report
# ./bin/yarn node -list
```

  - 访问集群文件

```
# ./bin/hadoop fs -mkdir /input
# ./bin/hadoop fs -ls hdfs://mycluster/
```

  - 主从切换Activate

```
# ./sbin/hadoop-daemon.sh stop namenode
```

案例4：高可用验证

1. 初始化集群
2. 验证集群

课堂练习

Tedu.cn
达内教育

总结和答疑