HA: Quorum Jornal Manager and shared storage

Quorum Jornal Manager(QJM)

核心概念

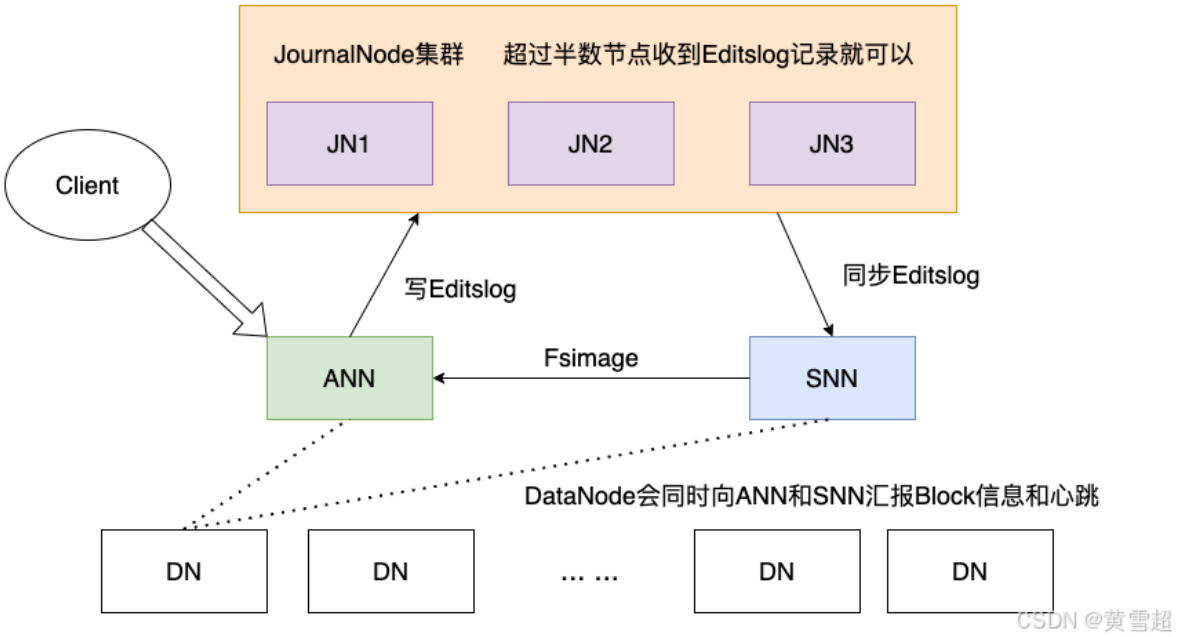
- **作用**：实现 NameNode 高可用（HA），防止单点故障。
- **组成**：由多个 **JournalNode（JN）** 组成的小集群（通常为3或5个节点）。
- **数据同步**：通过“多数确认”机制（Quorum）确保元数据变更的安全存储。

关键重点

- 基 于 QJM 的共享存储系统主要**用于保存 EditLog，并不保存 FSImage 文件。**FSImage 文件还是在 NameNode 的本地磁盘上。
- QJM 共享存储的基本思想来自于 `Paxos 算法`，采用多个称为 `JournalNode` 的节点组成的 `JournalNode` 集群来存储 EditLog。
- 每个 JournalNode 保存同样的 EditLog 副本。
- 每次 NameNode 写 EditLog 的时候，除了向本地磁盘写入 EditLog 之外，也会并行地向 **JournalNode 集群之中的每一个 JournalNode 发送写请求**，只要大多数 (majority) 的 JournalNode 节点返回成功就认为向 JournalNode 集群写入 EditLog 成功。
- 如果有 **2N+1** 台 JournalNode，那么根据大多数的原则，最多可以容忍有 **N** 台 JournalNode 节点挂掉。

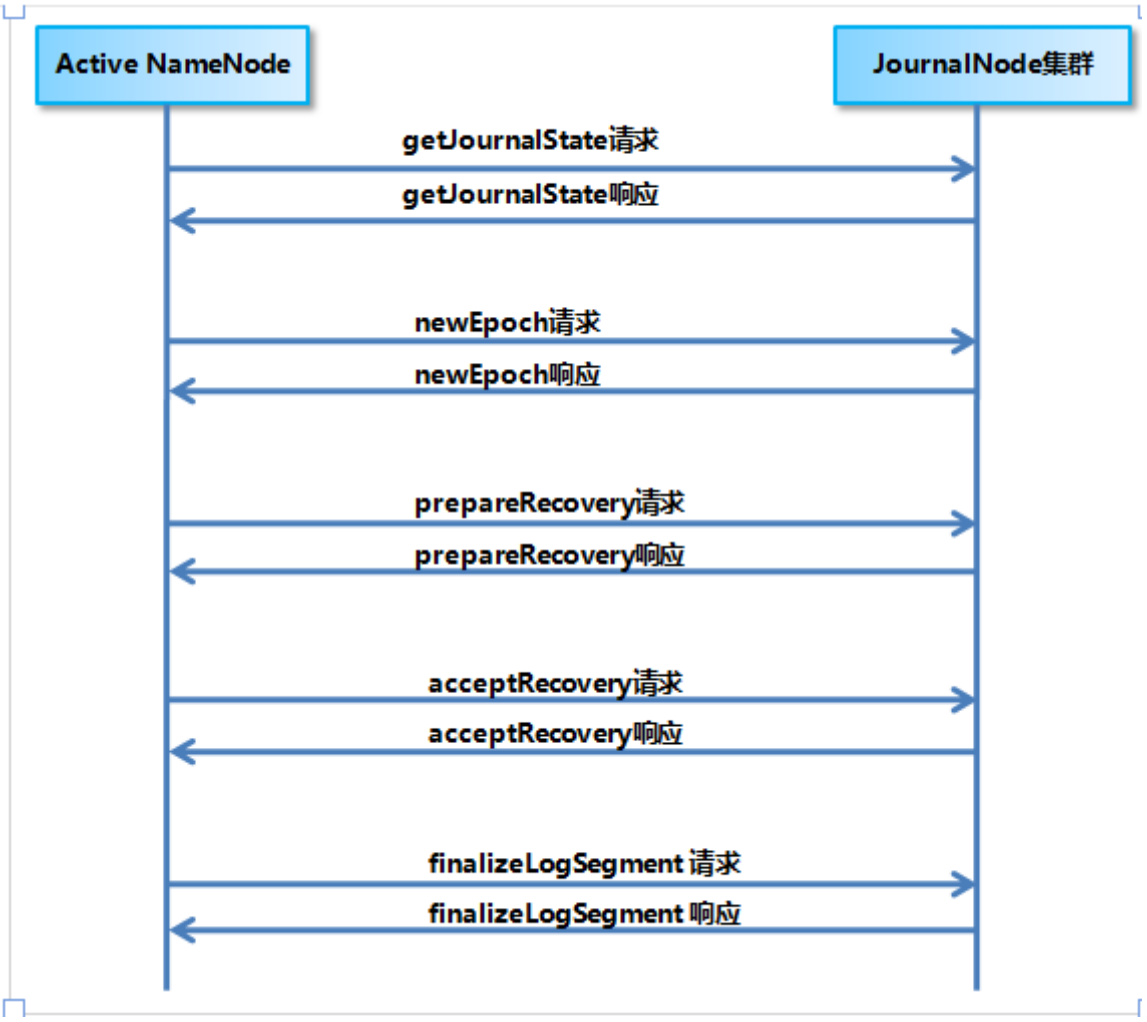
QJM 的共享存储系统的数据同步机制

Active NameNode 和 StandbyNameNode 使用 JouranlNode 集群来进行数据同步的过程如下图 所示，Active NameNode 首先把 EditLog 提交到 JournalNode 集群，然后 Standby NameNode 再从 JournalNode 集群定时同步 EditLog：



QJM 的共享存储系统的数据恢复机制

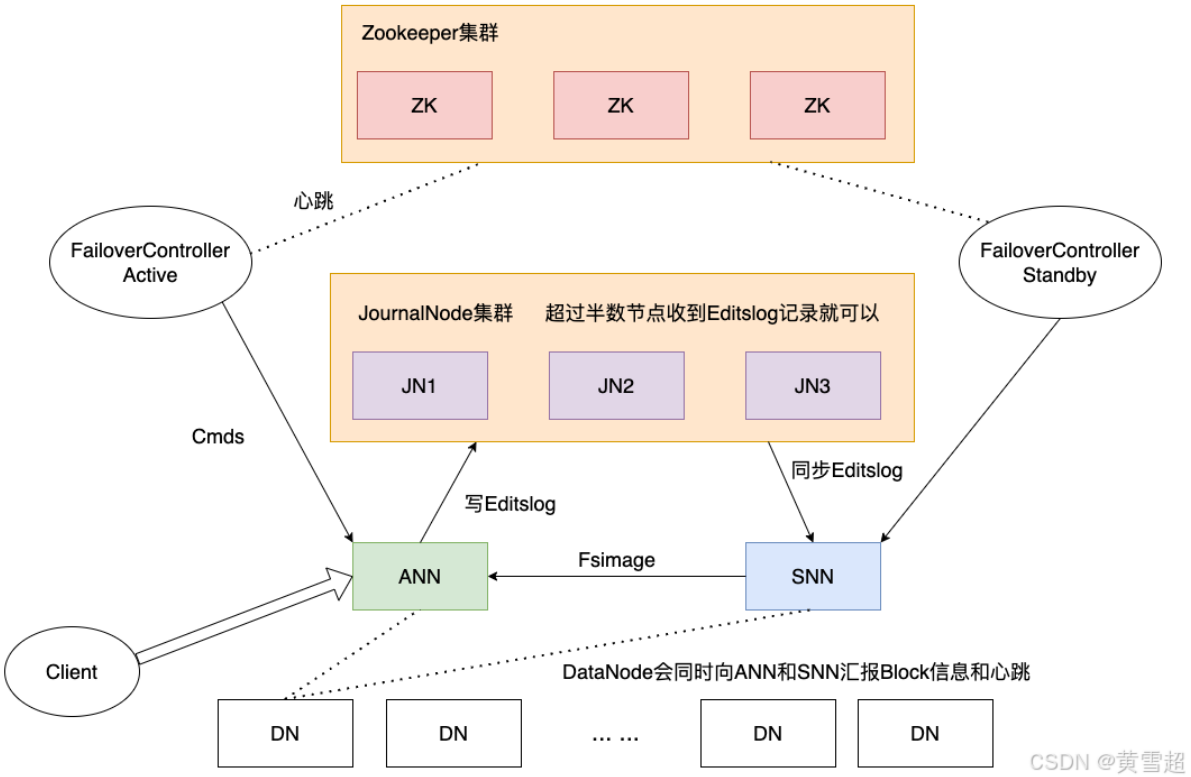
> 处于 Standby 状态的 NameNode 转换为 Active 状态的时候，有可能上一个 Active NameNode 发生了异常退出，那么 JournalNode 集群中各个 JournalNode 上的 EditLog 就可能会处于不一致的状态，所以首先要做的事情就是让 JournalNode 集群中各个节点上的 EditLog 恢复为一致。另外如前所述，当前处于 Standby 状态的 NameNode 的内存中的文件系统镜像有很大的可能是落后于旧的 Active NameNode 的，所以在 JournalNode 集群中各个节点上的 EditLog 达成一致之后，接下来要做的事情就是从 JournalNode 集群上补齐落后的 EditLog。只有在这两步完成之后，当前新的 Active NameNode 才能安全地对外提供服务。



快速正确切换

通过Journal Node实现NameNode HA时，可以手动将Standby NameNode切换成Active NameNode，也可以通过自动方式实现NameNode切换。但是通过手动进行切换StandbyNamenode为Active NameNode，效率太低了，不满足我们快速的需求。所以引入了Zookeeper进行协调。通过Zookeeper来检测Activate NameNode节点是否挂掉，如果挂掉立即将Standby NameNode切换成Active NameNode，自动实现NameNode HA。

结合Zookeeper后的架构设计图如下：



- ZKFailoverController：ZKFailoverController 作为独立的进程运行，对 NameNode 的主备切换进行总体控制。ZKFailoverController 能及时检测到 NameNode 的健康状况，在主 NameNode 故障时借助 Zookeeper 实现自动的主备选举和切换。
- Zookeeper集群：分布式协调器，NameNode选主使用。
- Journal集群：Journal集群作为共享存储系统保存HDFS运行过程中的元数据，ANN和SNN通过Journal集群实现元数据同步。

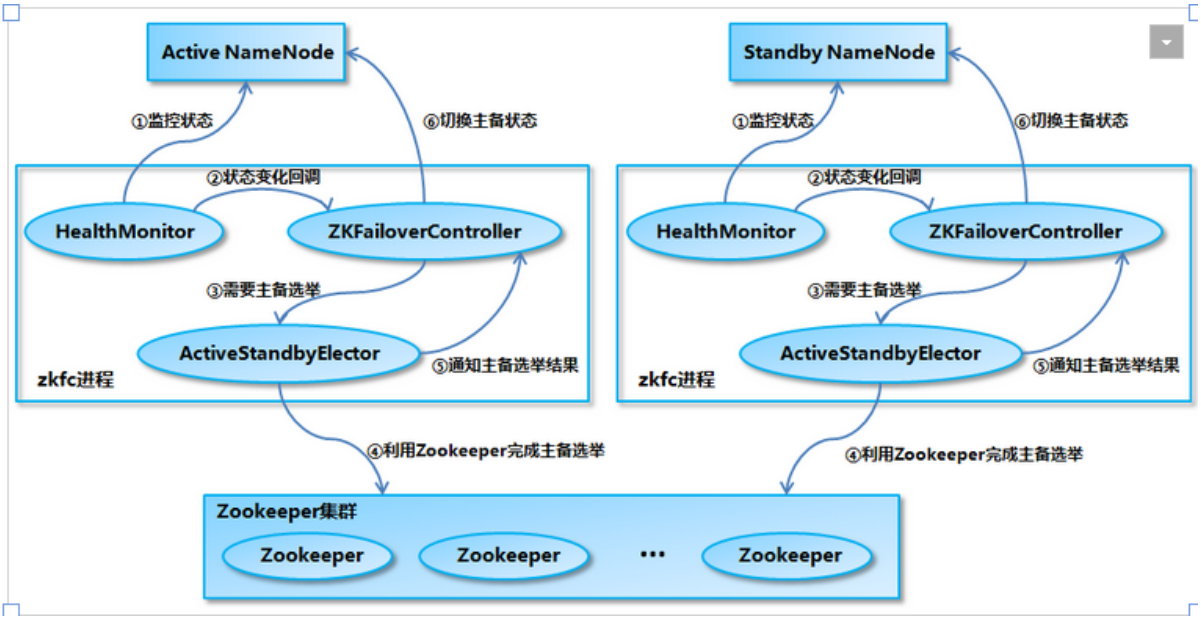
NameNode 的主备切换流程

核心组件

NameNode 主备切换主要由 `ZKFailoverController`、`HealthMonitor` 和 `ActiveStandbyElector` 这 3 个组件来协同实现：

- `ZKFailoverController` 作为 NameNode 机器上一个独立的进程启动 (在 hdfs 启动脚本之中的进程名为 zkfc)，启动的时候会创建 HealthMonitor 和 ActiveStandbyElector 这两个主要的内部组件，ZKFailoverController 在创建 HealthMonitor 和 ActiveStandbyElector 的同时，也会向 HealthMonitor 和 ActiveStandbyElector 注册相应的回调方法。
- `HealthMonitor` 主要负责检测 NameNode 的健康状态，如果检测到 NameNode 的状态发生变化，会回调 ZKFailoverController 的相应方法进行自动的主备选举。
- `ActiveStandbyElector` 主要负责完成自动的主备选举，内部封装了 Zookeeper 的处理逻辑，一旦 Zookeeper 主备选举完成，会回调 ZKFailoverController 的相应方法来进行 NameNode 的主备状态切换。

切换流程



- `HealthMonitor` 初始化完成之后会启动内部的线程来定时调用对应 NameNode 的 HAServiceProtocol RPC 接口的方法，对 NameNode 的健康状态进行检测。
- HealthMonitor 如果检测到 NameNode 的健康状态发生变化，会回调 `ZKFailoverController` 注册的相应方法进行处理。
- 如果 ZKFailoverController 判断需要进行主备切换，会首先使用 `ActiveStandbyElector` 来进行自动的主备选举。
- ActiveStandbyElector 与 Zookeeper 进行交互完成自动的主备选举。
- ActiveStandbyElector 在主备选举完成后，会回调 ZKFailoverController 的相应方法来通知当前的 NameNode 成为主 NameNode 或备 NameNode。
- `ZKFailoverController` 调用对应 NameNode 的 HAServiceProtocol RPC 接口的方法将 NameNode 转换为 Active 状态或 Standby 状态。

优缺点

优点	缺点
无单点故障（JN 是分布式集群）	需部署额外 JournalNode 服务
无需依赖外部存储	网络分区时可能影响仲裁决策
Hadoop 原生支持，社区推荐方案	

Shared Storage

在HDFS里面，会把当前正在提供服务的主节点称为Active NameNode(ANN)，而备用节点就叫做StandbyNameNode(SNN)。

我们知道，在NameNode中存储了HDFS中所有元数据信息，在HA设计中，当ANN挂掉后，期望的是SNN要及时顶上，这就需要将所有的元数据同步到SNN节点。

那我们来看看怎么实现好：

1. 在向HDFS中写入一个文件时，让元数据同步写入ANN和SNN。这样会导致当SNN挂掉时，影响到ANN，肯定是不行的。
2. 让元数据异步写入ANN和SNN。但这样有会有概率ANN挂掉的时候，元数据又没有及时异步写入到SNN，那切换后也会导致数据丢失。这个方案也不行。
3. 引入第三方存储。就是每次向HDFS中写入文件时，需要将Editslog同步写入这个第三方共享存储，这个步骤成功才能认定写文件成功，然后SNN定期从共享存储中同步Editslog，以便拥有完整元数据，确保ANN挂掉切换后数据也不丢失。

HDFS自然时采用的第三种方案，这个第三方存储，在HA方案中叫做“**共享存储**”。

---

1. 核心概念

- **作用**：通过共享存储（如NAS、云存储）实现 NameNode 的元数据同步。
- **典型方案**：NFS（网络文件系统）、AWS EBS、分布式存储（如Ceph）。

---

2. 工作原理

（1）写入流程

1. **Active NameNode** 将元数据变更直接写入共享存储的目录。
2. **Standby NameNode** 实时监听共享存储中的变更，并同步到本地内存。

（2）故障恢复

- 如果 Active NameNode 宕机：
  - Standby 检测到 Active 失联后，从共享存储加载最新的 EditLog。
  - 接管成为新的 Active NameNode。

---

3. 关键细节

- **存储要求**：共享存储必须自身高可用（如云存储的多副本机制）。
- **网络延迟**：共享存储的读写速度直接影响 NameNode 性能。
- **锁机制**：需通过 fencing（隔离）防止“脑裂”（即多个 Active 节点同时写入）。

---

4. 优缺点

优点	缺点
配置简单，依赖成熟存储系统	共享存储自身可能成为单点故障
适合已有高可用存储的环境	需要额外成本（如云存储费用）
无需管理 JournalNode 集群	网络不稳定时性能下降

QJM vs Shared Storage 对比

对比项	QJM	Shared Storage
数据存储位置	分布式 JournalNode 集群	外部共享存储（如NFS、云盘）
单点风险	无（依赖多数节点存活）	有（依赖存储自身的高可用性）
部署复杂度	需配置 JournalNode	需配置共享存储和网络挂载
适用场景	无现成共享存储的自建集群	云环境或已有高可用存储的企业
性能影响	依赖 JournalNode 网络延迟	依赖共享存储的I/O性能