高级Hadoop 2.x（二）

谭唐华

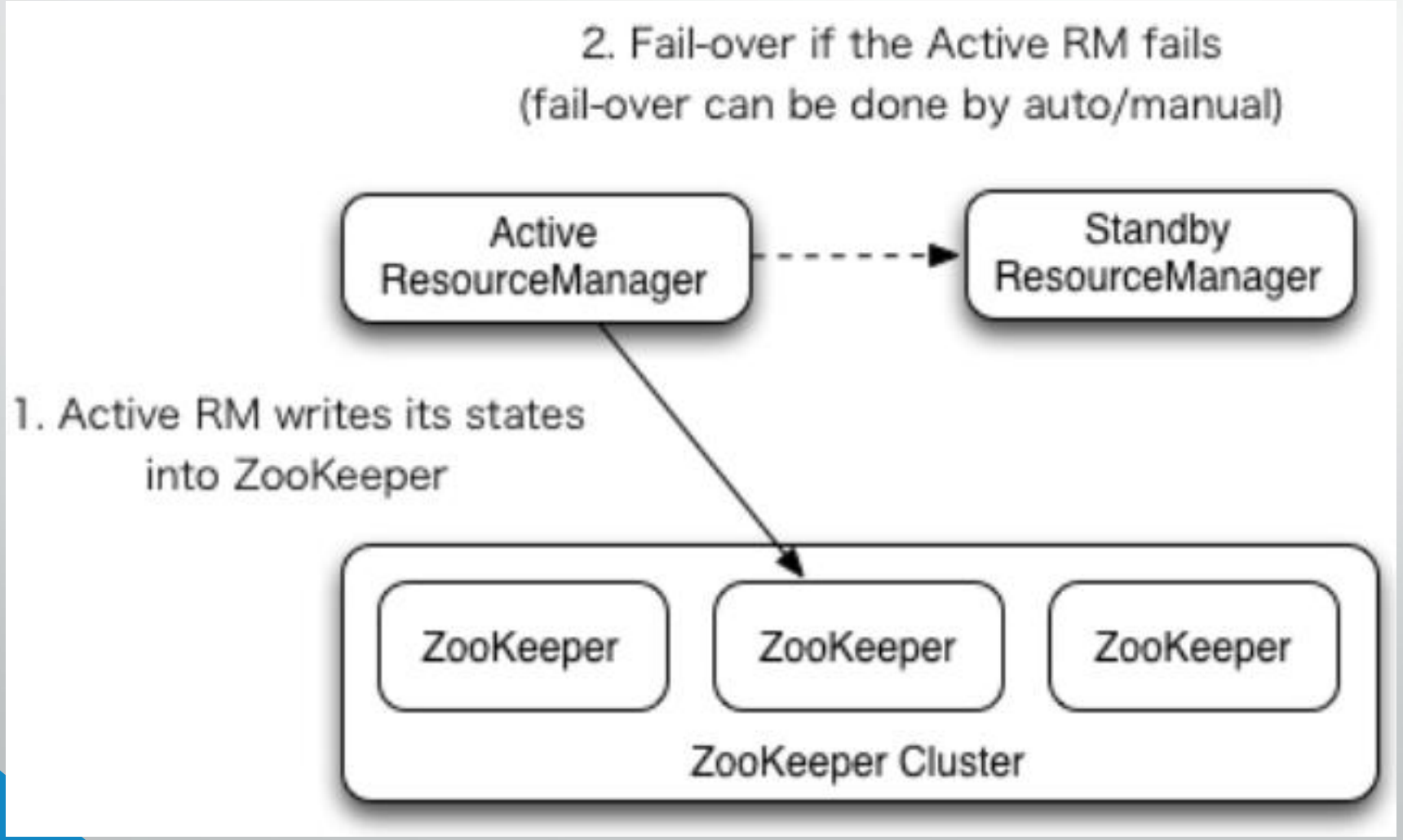
课程大纲

- ResourceManager 介绍
- HDFS Federation 介绍

ResourceManager High Availability

- The ResourceManager (RM) is responsible for tracking the resources in a cluster, and scheduling applications (e.g., MapReduce jobs).
- Prior to Hadoop 2.4, the ResourceManager is the single point of failure in a YARN cluster.
- The High Availability feature adds redundancy in the form of an Active/Standby ResourceManager pair to remove this otherwise single point of failure.

ResourceManager High Availability



ResourceManager High Availability

Configuration Property	Description
yarn.resourcemanager.zk-address	Address of the ZK-quorum. Used both for the state-store and embedded leader-election.
yarn.resourcemanager.ha.enabled	Enable RM HA
yarn.resourcemanager.ha.rm-ids	List of logical IDs for the RMs. e.g., "rm1,rm2"
yarn.resourcemanager.hostname.*rm-id*	For each *rm-id*, specify the hostname the RM corresponds to. Alternately, one could set each of the RM's service addresses.
yarn.resourcemanager.ha.id	Identifies the RM in the ensemble. This is optional; however, if set, admins have to ensure that all the RMs have their own IDs in the config
yarn.resourcemanager.ha.automatic-failover.enabled	Enable automatic failover; By default, it is enabled only when HA is enabled.
yarn.resourcemanager.ha.automatic-failover.embedded	Use embedded leader-elector to pick the Active RM, when automatic failover is enabled. By default, it is enabled only when HA is enabled.
yarn.resourcemanager.cluster-id	Identifies the cluster. Used by the elector to ensure an RM doesn't take over as Active for another cluster.
yarn.client.failover-proxy-provider	The class to be used by Clients, AMs and NMs to failover to the Active RM.
yarn.client.failover-max-attempts	The max number of times FailoverProxyProvider should attempt failover.
yarn.client.failover-sleep-base-ms	The sleep base (in milliseconds) to be used for calculating the exponential delay between failovers.
yarn.client.failover-sleep-max-ms	The maximum sleep time (in milliseconds) between failovers
yarn.client.failover-retries	The number of retries per attempt to connect to a ResourceManager.
yarn.client.failover-retries-on-socket-timeouts	The number of retries per attempt to connect to a ResourceManager on socket timeouts.

ResourceManager High Availability

Sample configurations

Here is the sample of minimal setup for RM failover.

```
<property>
  <name>yarn.resourcemanager.ha.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.cluster-id</name>
  <value>cluster1</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.rm-ids</name>
  <value>rm1,rm2</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname.rm1</name>
  <value>master1</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname.rm2</name>
  <value>master2</value>
</property>
<property>
  <name>yarn.resourcemanager.zk-address</name>
  <value>zk1:2181,zk2:2181,zk3:2181</value>
</property>
```

ResourceManager High Availability

Admin commands

yarn rmadmin has a few HA-specific command options to check the health/state of an RM, and transition to Active/Standby. Commands for HA take service id of RM set by yarn.resourcemanager.ha.rm-ids as argument.

```
$ yarn rmadmin -getServiceState rm1
active

$ yarn rmadmin -getServiceState rm2
standby
```

If automatic failover is enabled, you can not use manual transition command.

```
$ yarn rmadmin -transitionToStandby rm1
Automatic failover is enabled for org.apache.hadoop.yarn.client.RMHAServiceTarget@1d8299fd
Refusing to manually manage HA state, since it may cause
a split-brain scenario or other incorrect state.
If you are very sure you know what you are doing, please
specify the forcemanual flag.
```

ResourceManger Restart

- ResourceManager是管理资源和调度运行在YARN上的应用程序的中央机构，因此在一个YARN集群中ResourceManager可能是单点故障的，即只存在一个ResourceManager，这样在该节点出现故障时，就需要尽快重启ResourceManager，以尽可能地减少损失。本文将学习ResourceManager重启的特性，该特性使ResourceManager在重启时可以继续运行，并且在ResourceManager处于故障时对最终用户不可见。
- ResourceManager重启可以划分为两个阶段。第一阶段，增强的ResourceManager（RM）将应用程序的状态和其它认证信息保存到一个插入式的状态存储中。RM重启时将从状态存储中重新加载这些信息，然后重新开始之前正在运行的应用程序，用户不需要重新提交应用程序。第二阶段，重启时通过从NodeManagers读取容器的状态和从ApplicationMasters读取容器的请求，集中重构RM的运行状态。与第一阶段不同的是，在第二阶段中，之前正在运行的应用程序将不会在RM重启后被杀死，所以应用程序不会因为RM中断而丢失工作。

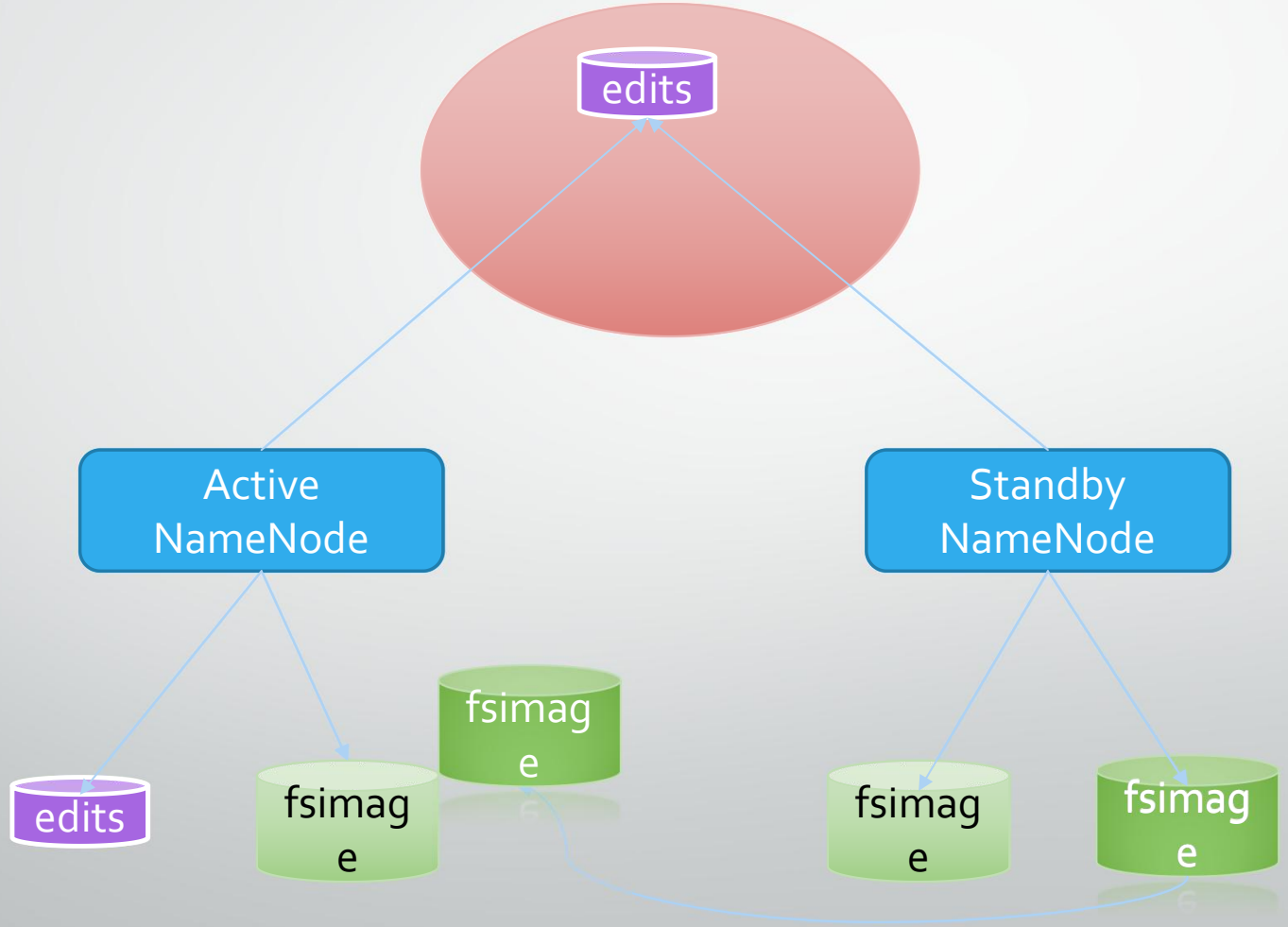
ResourceManger Restart

- RM在客户端提交应用时，将应用程序的元数据（如ApplicationSubmissionContext）保存到插入式的状态存储中，RM还保存应用程序的最终状态，如完成状态（失败, 被杀死, 执行成功），以及应用完成时的诊断。除此之外，RM还将在安全的环境中保存认证信息如安全密钥，令牌等。RM 任何时候关闭后，只要要求的信息（比如应用程序的元数据和运行在安全环境中的认证信息等）在状态存储中可用，在RM重启时，就可以从状态存储中获取应用程序的元数据然后重新提交应用。如果在RM关闭之前应用程序已经完成，不论是失败、被杀死还是执行成功，在RM重启后都不会再重新提交。

ResourceManger Restart

- NodeManagers和客户端在RM关闭期间将保持对RM的轮询，直到RM启动。当启动后，RM将通过心跳机制向正在与其会话的NodeManager和ApplicationMasters发送同步指令。目前NodeManager和ApplicationMaster处理该指令的方式为：NodeManager将杀死它管理的所有容器然后向RM重新注册，对于RM来说，这些重新注册的NodeManager与新加入的NodeManager相似。ApplicationMasters在接收到RM的同步指令后，将会关闭。在RM重启后，从状态存储中加载应用元数据和认证信息并放入内存后，RM将为每个还未完成的应用创建新的尝试。正如之前描述的，此种方式下之前正在运行的应用程序的工作将会丢失，因为它们已经被RM在重启后使用同步指令杀死了。

NameNode HA 元数据管理



单 NameNode架构的局限性

- **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（200 million）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DateNode。存储的需求从12PB增长到大于100PB。

- **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。

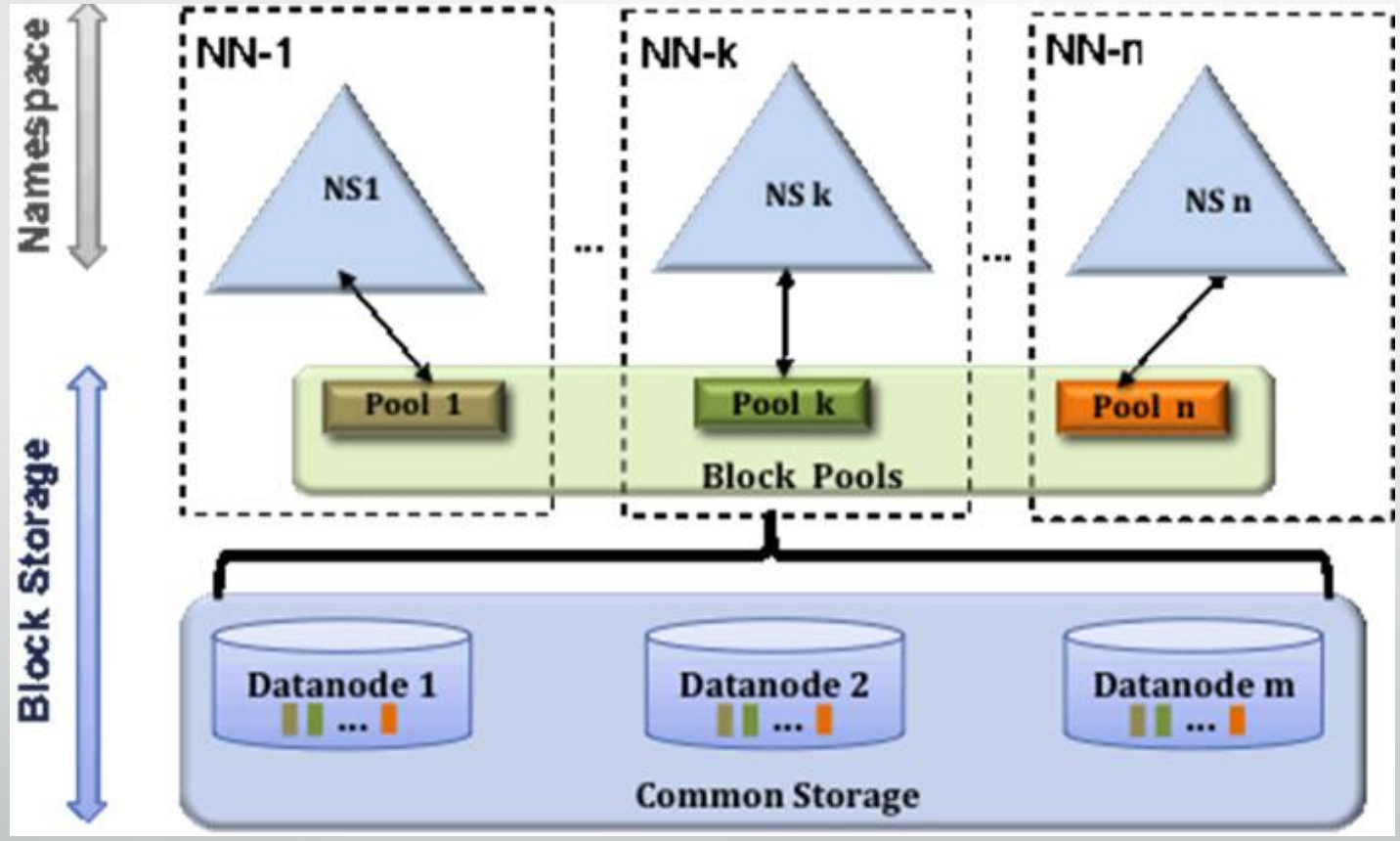
- **性能的瓶颈**

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。

课程大纲

- ResourceManager 介绍
- HDFS Federation 介绍

HDFS Federation 架构设计



HDFS Federation 配置

- Step 1:Add the following parameters to your configuration: **dfs.nameservices**: Configure with list of comma separated NameServiceIDs. This will be used by Datanodes to determine all the Namenodes in the cluster.
- Step 2:For each Namenode and Secondary Namenode/BackupNode/Checkpointer **add the following configuration suffixed with the corresponding NameServiceID** into the common configuration file.

HDFS Federation 配置

Deamon	Configuration Parameter	Port
NameNode	dfs.namenode.rpc-address	8020
	dfs.namenode.servicerpc-address	8022
	dfs.namenode.http-address	50070
	dfs.namenode.name.dir	
	dfs.namenode.edits.dir	
	dfs.namenode.checkpoint.dir	
	dfs.namenode.checkpoint.edits.dir	
	dfs.namenode.https-address	50470
Secondary Namenode	dfs.namenode.secondary.http-address	50090
BackupNode	dfs.namenode.backup.address	50100
	dfs.namenode.backup.http-address	50105

HDFS Federation 配置

- 机器与服务规划

bigdata-senior01.ibeifeng.com	bigdata-senior02.ibeifeng.com	
NameNode	**NameNode**	
DataNode	**DataNode**	
SecondaryNameNode	SecondaryNameNode	可选

- 修改hdfs-site.xml

```
<property>
        <name>dfs.nameservices</name>
        <value>ns1,ns2</value>
</property>
<property>……</property>
<property>……</property>
```

HDFS Federation 配置

- 修改 core-site.xml

bigdata-senior01.ibeifeng.com:

```
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://bigdata-senior01.ibeifeng.com:8020</value>
</property>
```

bigdata-senior02.ibeifeng.com:

```
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://bigdata-senior02.ibeifeng.com:8020</value>
</property>
```

Formatting NameNodes

Step 1: Format a namenode using the following command:

```
> $HADOOP_PREFIX_HOME/bin/hdfs namenode -format [-clusterId <cluster_id>]
```

Choose a unique cluster_id, which will not conflict other clusters in your environment. If it is not provided, then a unique ClusterID is auto generated.

Step 2: Format additional namenode using the following command:

```
> $HADOOP_PREFIX_HOME/bin/hdfs namenode -format -clusterId <cluster_id>
```

Note that the cluster_id in step 2 must be same as that of the cluster_id in step 1. If they are different, the additional Namenodes will not be part of the federated cluster.

每个NameNode 节点格式化NameNode，命令

$ bin/hdfs namenode -format -clusterId hdfs-cluster

每个NameNode 节点启动NameNode，命令

$ sbin/hadoop-daemon.sh start namenode

数据节点上启动DataNode，命令 **$ sbin/hadoop-daemon.sh start datanode**

启动NN节点对应的SNN，命令 **$ sbin/hadoop-daemon.sh start secondarynamenode**

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）

Distributed Copy

- **DistCp Version 2 (distributed copy) is a tool used for large inter/intra-cluster copying.** It uses MapReduce to effect its distribution, error handling and recovery, and reporting. It expands a list of files and directories into input to map tasks, each of which will copy a partition of the files specified in the source list.
- The erstwhile implementation of DistCp has its share of quirks and drawbacks, both in its usage, as well as its extensibility and performance. **The purpose of the DistCp refactor was to fix these shortcomings, enabling it to be used and extended programmatically.** New paradigms have been introduced to improve runtime and setup performance, while simultaneously retaining the legacy behaviour as default.

HFTP Introduction

- HFTP is a Hadoop filesystem implementation that lets you read data from a remote Hadoop HDFS cluster. The reads are done via HTTP, and data is sourced from DataNodes.
- HFTP is a read-only filesystem, and will throw exceptions if you try to use it to write data or modify the filesystem state.
- HFTP is primarily useful if you have **multiple HDFS clusters with different versions** and you need to move data from one to another. **HFTP is wire-compatible even between different versions of HDFS.**

数据类型

- 数据类型都实现Writable接口，以便用这些类型定义的数据可以被序列化进行网络传输和文件存储。
- 基本数据类型

BooleanWritable：标准布尔型数值　　ByteWritable：单字节数值

DoubleWritable：双字节数值　　FloatWritable：浮点数

IntWritable：整型数　　**LongWritable：长整型数**

Text：使用UTF8格式存储的文本

NullWritable：当<key,value>中的key或value为空时使用

数据类型

- Writable - value
  - write() 是把每个对象序列化到输出流。
  - readFields()是把输入流字节反序列化。
- WritableComparable - key必须要实现
- Java值对象的比较：

  重写 toString()、hashCode()、equals()方法