HDFS 分布式文件系统简介

November 18, 2020

HDFS 分布式文件系统的架构

HDFS 是 GFS [1] 的开源实现，HDFS [2] 适合一次写入，多次读出的场景。比如，文件的多路合并、生产者-消费者队列等。

文件被切分为固定大小的数据块集合 (blocks)，进行存储。在数据块创建时，命名节点 (NameNode) 服务器会给这个数据块分配一个不可变，全局唯一的 64 位的数据块标识 (block handle)。

HDFS 分布式文件系统，主要由单个命名节点 (NameNode) 服务器和多个数据节点 (DataNodes) 服务器组成。

HDFS 分布式文件系统的架构

HDFS 是 GFS [1] 的开源实现，HDFS [2] 适合一次写入，多次读出的场景。比如，文件的多路合并、生产者-消费者队列等。

文件被切分为固定大小的数据块集合 (blocks)，进行存储。在数据块创建时，命名节点 (NameNode) 服务器会给这个数据块分配一个不可变，全局唯一的 64 位的数据块标识 (block handle)。

HDFS 分布式文件系统，主要由单个命名节点 (NameNode) 服务器和多个数据节点 (DataNodes) 服务器组成。

- 命名节点 (NameNode) 服务器,

HDFS 分布式文件系统的架构

HDFS 是 GFS [1] 的开源实现，HDFS [2] 适合一次写入，多次读出的场景。比如，文件的多路合并、生产者-消费者队列等。

文件被切分为固定大小的数据块集合 (blocks)，进行存储。在数据块创建时，命名节点 (NameNode) 服务器会给这个数据块分配一个不可变，全局唯一的 64 位的数据块标识 (block handle)。

HDFS 分布式文件系统，主要由单个命名节点 (NameNode) 服务器和多个数据节点 (DataNodes) 服务器组成。

- 命名节点 (NameNode) 服务器,
  1. 管理维护命名空间树 (namespace tree)。

HDFS 分布式文件系统的架构

HDFS 是 GFS [1] 的开源实现，HDFS [2] 适合一次写入，多次读出的场景。比如，文件的多路合并、生产者-消费者队列等。

文件被切分为固定大小的数据块集合 (blocks)，进行存储。在数据块创建时，命名节点 (NameNode) 服务器会给这个数据块分配一个不可变，全局唯一的 64 位的数据块标识 (block handle)。

HDFS 分布式文件系统，主要由单个命名节点 (NameNode) 服务器和多个数据节点 (DataNodes) 服务器组成。

- 命名节点 (NameNode) 服务器,
  1. 管理维护命名空间树 (namespace tree)。
  2. 管理维护文件 (file) 到数据块集合 (blocks) 的映射关系。

HDFS 分布式文件系统的架构

HDFS 是 GFS [1] 的开源实现，HDFS [2] 适合一次写入，多次读出的场景。比如，文件的多路合并、生产者-消费者队列等。

文件被切分为固定大小的数据块集合 (blocks)，进行存储。在数据块创建时，命名节点 (NameNode) 服务器会给这个数据块分配一个不可变，全局唯一的 64 位的数据块标识 (block handle)。

HDFS 分布式文件系统，主要由单个命名节点 (NameNode) 服务器和多个数据节点 (DataNodes) 服务器组成。

- 命名节点 (NameNode) 服务器,
  1. 管理维护命名空间树 (namespace tree)。
  2. 管理维护文件 (file) 到数据块集合 (blocks) 的映射关系。
  3. 管理维护数据块 (blocks) 到数据节点集合 (DataNodes) 的映射关系。

HDFS 分布式文件系统的架构

HDFS 是 GFS [1] 的开源实现，HDFS [2] 适合一次写入，多次读出的场景。比如，文件的多路合并、生产者-消费者队列等。

文件被切分为固定大小的数据块集合 (blocks)，进行存储。在数据块创建时，命名节点 (NameNode) 服务器会给这个数据块分配一个不可变，全局唯一的 64 位的数据块标识 (block handle)。

HDFS 分布式文件系统，主要由单个命名节点 (NameNode) 服务器和多个数据节点 (DataNodes) 服务器组成。

- 命名节点 (NameNode) 服务器,
  1. 管理维护命名空间树 (namespace tree)。
  2. 管理维护文件 (file) 到数据块集合 (blocks) 的映射关系。
  3. 管理维护数据块 (blocks) 到数据节点集合 (DataNodes) 的映射关系。
  4. 管理数据块的租约，数据块的垃圾回收，不同 DataNode 节点的块集合 (blocks) 迁移。

HDFS 分布式文件系统的架构

- 数据节点 (DataNode) 服务器，

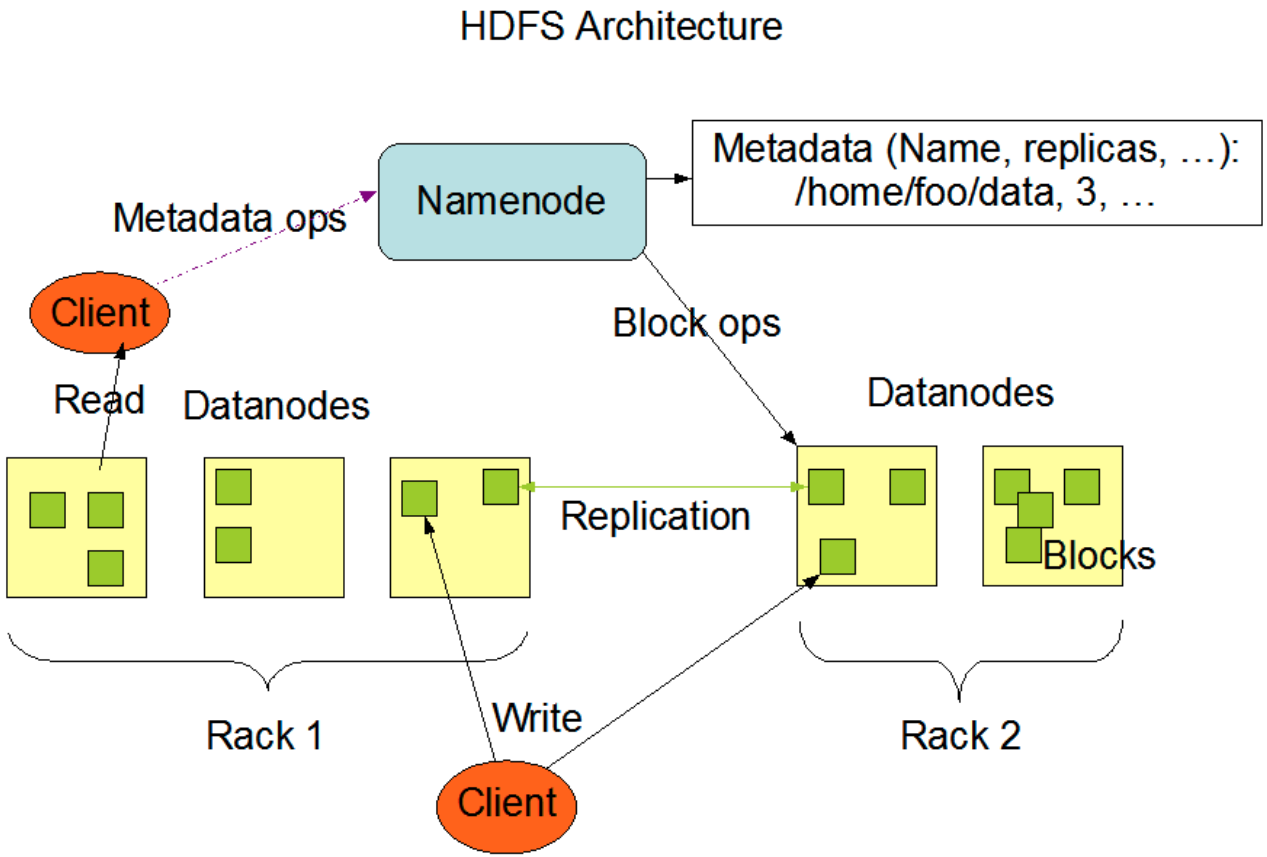
HDFS 分布式文件系统的架构

- 数据节点 (DataNode) 服务器，
  1. 管理维护实际存储数据块集合 (blocks) 的本地文件，定时向命名节点汇报数据块集合 (blocks) 的信息。

HDFS 分布式文件系统的架构

- 数据节点 (DataNode) 服务器，
  1. 管理维护实际存储数据块集合 (blocks) 的本地文件，定时向命名节点汇报数据块集合 (blocks) 的信息。
  2. 通过数据块标识 (block handle) 来，实现真实的文件读写。

HDFS 分布式文件系统的架构



读取操作

读的流程

- 当需要读一个文件时，HDFS 客户端首先向命名节点服务器 (NameNode) 发送读请求。
- 命名节点服务器 (NameNode) 返回所有这个文件数据块 (blocks) 的数据节点服务器集合 (DataNodes)。
- HDFS 客户端直接向对应的数据节点服务器 (DataNode) 发送传输相应数据块 (block) 的请求。
- 数据节点服务器 (DataNode) 响应客户请求，将对应的数据块 (block) 返回给客户端。

租约和修改顺序

每个修改操作必须一致性在所有的副本集合上进行操作。HDFS 使用租约机制来保证所有的副本上，实现一致的修改顺序 (不会同时出现两个主 DataNode)。

租约初始化通常是 60 秒，并且在 NameNode 和 DataNodes 间心跳中延续租约。

NameNode 会从副本服务器集合中挑选一个 DataNode，并赋予租约，被赋予租约的服务器定义为主 DataNode。因此，全局的修改顺序首先由 NameNode 选择租约顺序定义，在租约中由主 DataNode 分配的序列号定义。

租约和修改顺序

写入的步骤：

- 1. 客户端向 NameNode 发出请求，询问那个 DataNode 拥有当前请求数据块 (block) 的租约 (less)，以及其副本 (replicas) 所在的 DataNodes 集合。如果，NameNode 没有赋予把请求数据块 (block) 的租约赋予任何 DataNode，那么，选择一个主 DataNode (它的数据块版本必须大于等于 NameNode 上的数据块版本号)，并赋予租约。并且将这个数据块版本号 (block version number) 增加并持久化，同时通知主 DataNode 和其他 DataNodes，将新的数据块版本号 (block version number) 持久化。

租约和修改顺序

写入的步骤：

- 1. 客户端向 NameNode 发出请求，询问那个 DataNode 拥有当前请求数据块 (block) 的租约 (less)，以及其副本 (replicas) 所在的 DataNodes 集合。如果，NameNode 没有赋予把请求数据块 (block) 的租约赋予任何 DataNode，那么，选择一个主 DataNode (它的数据块版本必须大于等于 NameNode 上的数据块版本号)，并赋予租约。并且将这个数据块版本号 (block version number) 增加并持久化，同时通知主 DataNode 和其他 DataNodes，将新的数据块版本号 (block version number) 持久化。
- 2. NameNode 向客户端回应消息，指明主 DataNode 和其他副本的 DataNode 集合的位置。客户端将这些信息保存起来，用于未来的操作。客户端只有主 DataNode 不可以访问或者租约超时，才会再次请求 NameNode。

租约和修改顺序

写入的步骤：

- 3. 客户端将数据推送到所有副本 DataNodes 中，使用任意的顺序发送。每个 DataNode 将会把这个传输过来的文件块数据缓存在内部的 LRU 缓存中。

租约和修改顺序

写入的步骤：

- 3. 客户端将数据推送到所有副本 DataNodes 中，使用任意的顺序发送。每个 DataNode 将会把这个传输过来的文件块数据缓存在内部的 LRU 缓存中。
- 4. 一旦客户端收到所有 DataNodes 的数据写入缓存完成应答时，客户端向主 DataNode 发送写请求，这个请求会标识之前推送到所有 DataNodes 的数据。主 DataNode 会对它收到的所有数据修改操作，特别是来自不同的客户端的数据修改操作，赋予一个连续序列号，这样就提供了必要的串行化 (serialization)。主 DataNode 根据这个全局顺序将缓存写入本地磁盘。

租约和修改顺序

写入的步骤：

- 3. 客户端将数据推送到所有副本 DataNodes 中，使用任意的顺序发送。每个 DataNode 将会把这个传输过来的文件块数据缓存在内部的 LRU 缓存中。
- 4. 一旦客户端收到所有 DataNodes 的数据写入缓存完成应答时，客户端向主 DataNode 发送写请求，这个请求会标识之前推送到所有 DataNodes 的数据。主 DataNode 会对它收到的所有数据修改操作，特别是来自不同的客户端的数据修改操作，赋予一个连续序列号，这样就提供了必要的串行化 (serialization)。主 DataNode 根据这个全局顺序将缓存写入本地磁盘。
- 5. 主 DataNode 将客户端的写请求转发到所有 DataNodes，其他 DataNodes 按照主 DataNode 赋予的顺序，将缓存写入本地磁盘。

租约和修改顺序

写入的步骤：

- 6. 其他 DataNodes 完成写入磁盘操作，向主 DataNode 应答，表明它完成了操作。

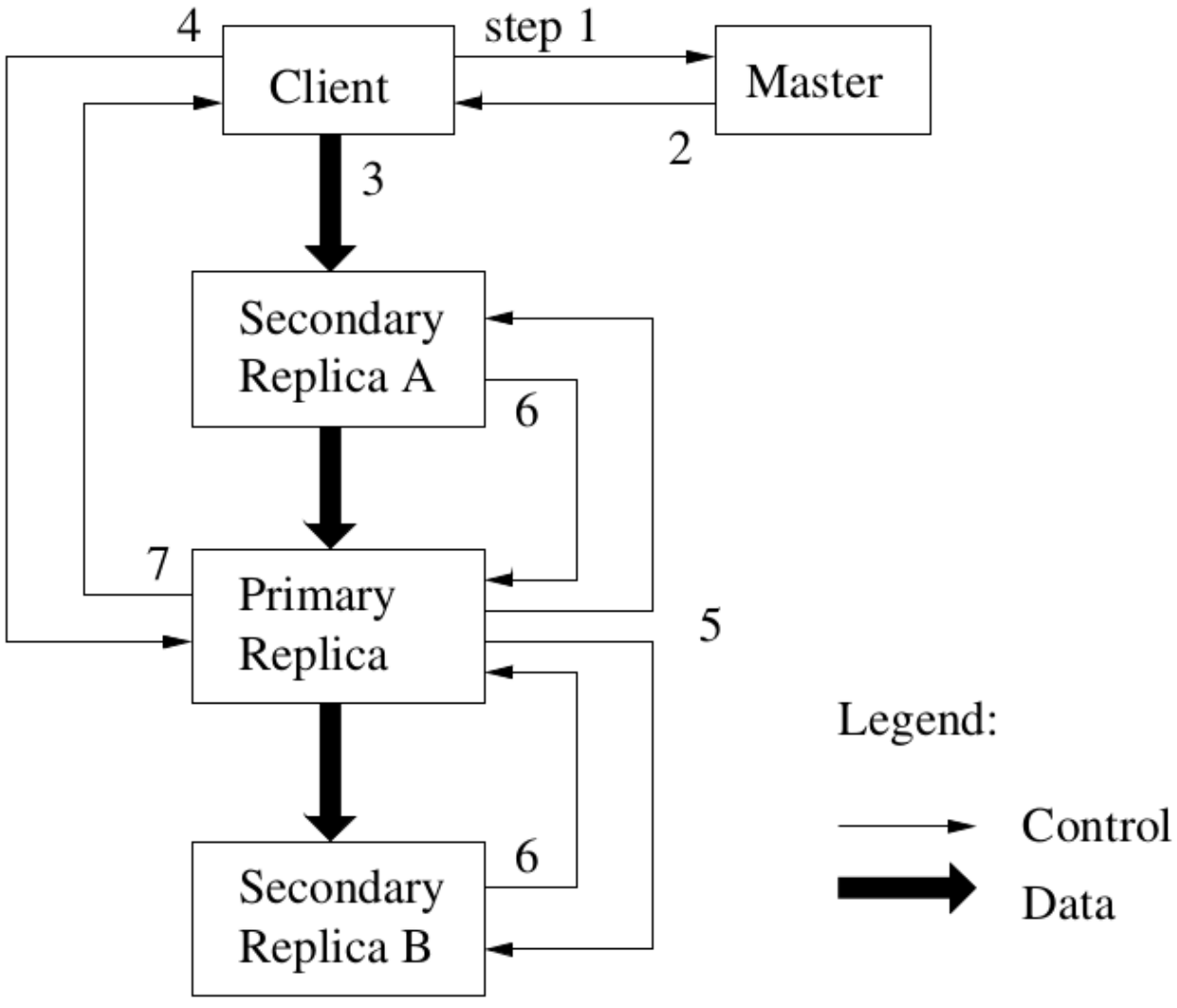
租约和修改顺序

写入的步骤：

- 6. 其他 DataNodes 完成写入磁盘操作，向主 DataNode 应答，表明它完成了操作。
- 7. 主 DataNode 收到所有其他 DataNodes 的应答，主 DataNode 应答客户端，表明数据块 (block) 已经写入完成。在每个阶段，每个 DataNode 出现错误时，都会把错误反馈给客户端。可能出现写操作在主 DataNode 和部分副本 DataNodes 成功的情况，但是请求依然是被认为失败的。一旦失败，客户端尝试几次步骤 3 到步骤 7 恢复，如果还是不成功。就从步骤 1 开始新的一轮写入操作。

租约和修改顺序

Master 就是 NameNode，Primary 就是主 DataNode。



原子记录追加 (Atomic Record Appends)

HDFS 提供一种原子追加的操作，被叫做记录追加 (record append)，HDFS 至少以原子方式将数据追加到文件一次 (append atomically at least once)。

原子追加和普通的写操作是一样的，除了个别的地方。

- 1. 客户端将数据推送到每个 DataNode 的缓存中，然后先主 DataNode 发送写命令。

原子记录追加 (Atomic Record Appends)

HDFS 提供一种原子追加的操作，被叫做记录追加 (record append)，HDFS 至少以原子方式将数据追加到文件一次 (append atomically at least once)。

原子追加和普通的写操作是一样的，除了个别的地方。

- 1. 客户端将数据推送到每个 DataNode 的缓存中，然后先主 DataNode 发送写命令。
- 2. 主 DataNode 检查是否对当前文件追加会让文件超过最大长度 (64M)，如果是真，主 DataNode 将该文件填充到最大长度。然后，通知客户端在下一个数据块 (block) 处重试。

原子记录追加 (Atomic Record Appends)

HDFS 提供一种原子追加的操作，被叫做记录追加 (record append)，HDFS 至少以原子方式将数据追加到文件一次 (append atomically at least once)。

原子追加和普通的写操作是一样的，除了个别的地方。

- 1. 客户端将数据推送到每个 DataNode 的缓存中，然后先主 DataNode 发送写命令。
- 2. 主 DataNode 检查是否对当前文件追加会让文件超过最大长度 (64M)，如果是真，主 DataNode 将该文件填充到最大长度。然后，通知客户端在下一个数据块 (block) 处重试。
- 3. 如果追加不超过文件的最大长度约束，主 DataNode 将缓存追加到本地文件末尾，并且告诉其他的 DataNode 文件写入的准确偏移量。最终，回应客户端写入成功，告诉写入的偏移量。

原子记录追加 (Atomic Record Appends)

如果记录追加操作在任意的 DataNode 失败的化，客户端重试上述操作。因此，同一数据块 (block) 的不同副本可能包含不同的数据，其中可能包含同一记录的重复项。HDFS 不保证所有的副本都是字节完全一样的。它只保证记录至少作为原子单元写入一次。这个来源与一个简单的观察，如果客户端操作是成功的，那么记录必定已经在所有副本的相同偏移量处写入成功。

在此之后，所有的副本的长度至少和写入的记录长度一样，因此，以后的记录会被分配更高的偏移量或者新的数据块 (block)(不和旧的记录重叠)，即使一个不同的 DataNode 成为新的主 DataNode。

HDFS 文件系统的一致性保证

文本区域状态定义：

- 如果无论从哪个副本 (replicas) 读取一个文件区域 (file region)，所有的客户端总是看到一样的数据，那么，这个文件区域 (file region) 就是一致的 (consistent)。
- 一个文件区域 (file region) 是一致的，并且客户端能够看到它修改的完整内容。那么，这个文件区域 (file region) 就是有定义的 (defined)。

HDFS 文件系统的一致性保证

HDFS 文件操作 (mutations) 的两种方法：

- 客户端指定文件写入的位置，然后进行文件数据写入记录操作 (record write)。
- 客户端不指定文件写入的位置，只对文件末尾进行原子追加记录操作 (atomic record append)。成功后，HDFS 文件系统给客户端返回一个开始区域的位置。

HDFS 文件系统的一致性保证

文件操作 (mutations) 的几种状态：

- 如果一个串行写入成功 (没有并发的写入)，那么这个文件区域 (file region) 是有定义的 (defined)，同样也是一致的 (consistent)。
- 一个并发的写入成功，那么这个文件区域 (file region) 不是有定义的 (defined)，可能其修改的一部分内容被覆盖了，但是一致的 (consistent)。
- 一个并发的原子记录追加写入成功，那么包含该记录的文件区域 (file region) 是有定义的 (defined)，也是一致的 (consistent)。
- 一个失败的写入，这个文件区域 (file region) 是不一致的 (inconsistent)，客户端可能在不同的时间和不同的地方，看到不同的数据。

HDFS 文件系统的 Shell 操作

put

使用方法：bin/hdfs dfs -put <localsrc> ... <dst>
从本地文件系统中复制单个或多个源路径到 HDFS 文件系统。
也支持从标准输入中读取输入写入目标文件系统。实例：

```
bin/hdfs dfs -put tmp.txt  /
#将本地的tmp.txt文件拷贝到HDFS文件系统的根目录下。
```

HDFS 文件系统的 Shell 操作

get

使用方法：bin/hdfs dfs -get [-ignorecrc] [-crc] <src> <localdst>
从 HDFS 文件系统复制文件到本地文件系统。可用-ignorecrc 选项复制 CRC 校验失败的文件。使用-crc 选项复制文件以及 CRC 信息。实例：

```
bin/hdfs dfs -get /tmp.txt .
#将HDFS文件系统上根目录下的tmp.txt拷贝到本地文件系统上。
```

HDFS 文件系统的 Shell 操作

mkdir

使用方法：bin/hdfs dfs -mkdir -p <paths> 接受路径指定的 uri 作为参数，创建这些目录。其行为类似于 Unix 的 mkdir -p，它会创建路径中的各级父目录。实例：

```
bin/hdfs dfs -mkdir /data1/note1 /data1/note2
#创建两个目录，note1和note2
```

HDFS 文件系统的 Shell 操作

ls

使用方法：bin/hdfs dfs -ls <paths> 列举指定的 HDFS 路径的信息。实例：

```
bin/hdfs dfs -ls /
#列举根目录的文件的信息。
```

HDFS 文件系统的 Shell 操作

cat

使用方法：bin/hdfs dfs -cat URI 将路径指定文件的内容输出到 stdout。实例：

```
bin/hdfs dfs -cat /tmp.txt
#输出HDFS文件系统上的文件的内容。
```

HDFS 文件系统的 Shell 操作

rm

使用方法：bin/hdfs dfs -rm -r URI 将路径指定文件删除。实例：

```
bin/hdfs dfs -rm -r data1/note1/tmp.txt
```

HDFS 文件系统的 Shell 操作

一个完整的例子：

```
bash-4.1# bin/hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - root supergroup          0 2015-05-16 05:42 /user
bash-4.1# bin/hdfs dfs -mkdir -p /data1/note1 data2/note2
bash-4.1# bin/hdfs dfs -put tmp.txt /data1/note1
bash-4.1# bin/hdfs dfs -ls /data1/note1
Found 1 items
-rw-r--r--   1 root supergroup         11 2020-10-22 23:42 /data1/note1/tmp.txt
bash-4.1# bin/hdfs dfs -cat /data1/note1/tmp.txt
hello hdfs
bash-4.1# bin/hdfs dfs -get /data1/note1/tmp.txt tmp2.txt
20/10/22 23:43:24 WARN hdfs.DFSClient: DFSInputStream has been closed already
bash-4.1# ls
LICENSE.txt  README.txt  etc      input  libexec  sbin   tmp.txt
NOTICE.txt   bin         include  lib    logs     share  tmp2.txt
bash-4.1# cat tmp2.txt
hello hdfs
bash-4.1#
```

HDFS 文件系统 Java API

HDFS 文件系统文件访问的功能，主要由 LocalFileSystem 类和 DistributedFileSystem 类实现。它们的公共父类是 FileSystem 类，文件访问的 API 主要是通过对 FileSystem 类的抽象方法，进行操作。

- LocalFileSystem 类：主要进行本地文件系统的读写、删除等操作。
- DistributedFileSystem 类：主要是进行 HDFS 文件系统的读写、删除等操作。

HDFS 文件系统 Java API

FileSystem 类是一个工厂类，需要通过传进 Configuration 类，来选择不同的实现子类。Configuration 类可以看作是一个 key,value 对的表，主要是对fs.defaultFS属性进行配置，来选择不同的子类。

- hdfs://localhost:9000 选择 DistributedFileSystem 类，访问 HDFS 文件系统。
- file:/// 选择 LocalFileSystem 类，访问本地文件系统。

HDFS 文件系统 Java API

org/apache/hadoop/fs/FileSystem.java 类的主要操作方法有：

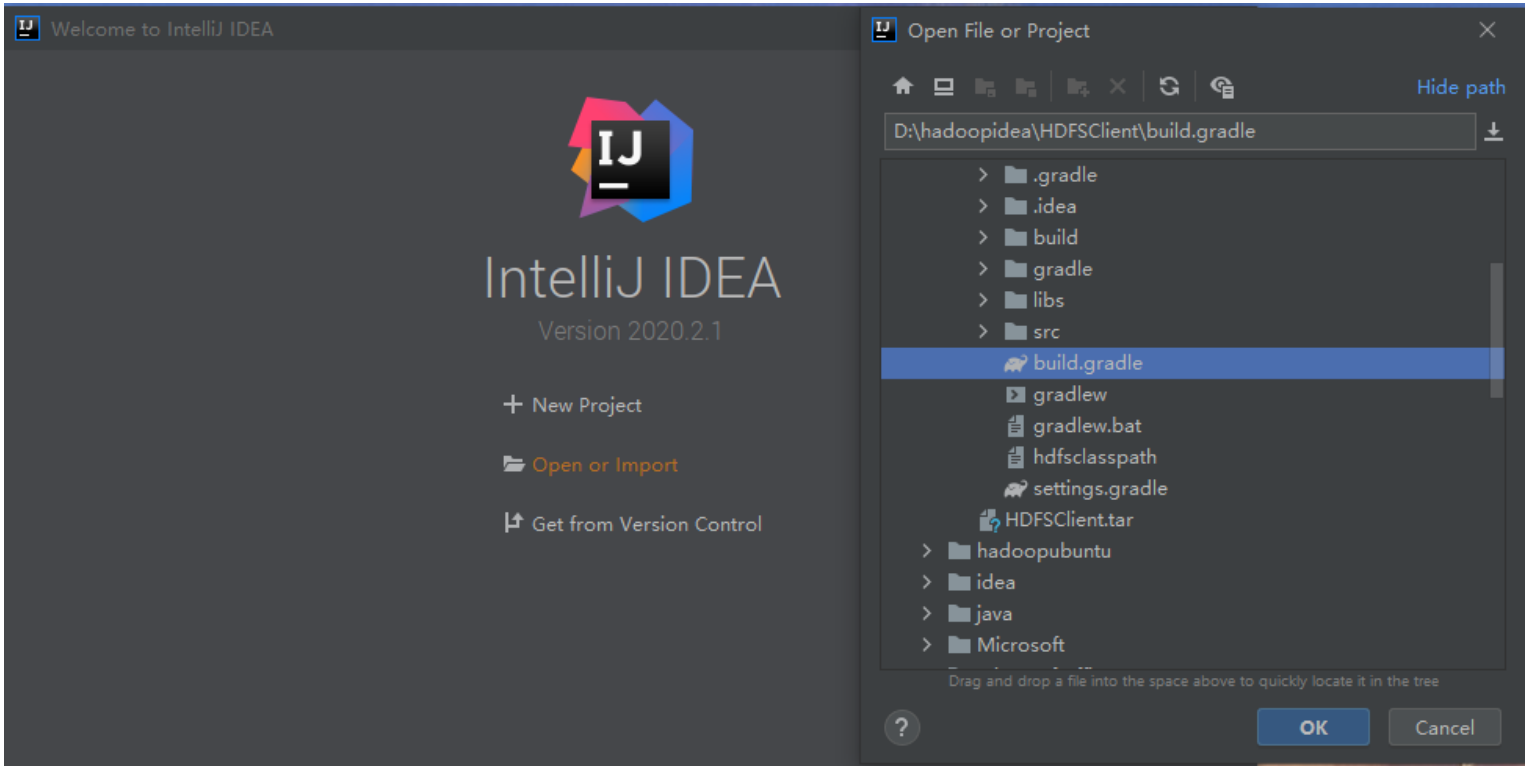
```
//得到文件的状态
public FileStatus getFileStatus(Path f) throws IOException
//打开一个文件，准备读取
public FSDataInputStream open(Path f) throws IOException
//创建一个文件，准备写入
public FSDataOutputStream create(Path f, boolean overwrite) throws IOException
//追加文件内容
public FSDataOutputStream append(Path f) throws IOException
//删除文件
public abstract boolean delete(Path f, boolean recursive) throws IOException
//创建目录
public boolean mkdirs(Path f) throws IOException
```

Java 实验环境

本实验，主要使用 IntelliJ IDEA Community IDE 开发，Gradle 集成编译工具等来做实验。gradle 集成编译工具，完成代码的编译，打包成 jar，将 jar 上传到 Ubuntu 服务器等一系列自动化操作。

1. 初始化工程时，在 IDE 上，选择 build.gradle 文件进行导入工程。
2. 选择 gradle 任务中的 remoteUpload 任务，进行执行。
3. remoteUpload 任务执行成功后，会将 hdfsClient-1.1.0.jar 上传到服务器/root/目录下。
4. ssh 登录服务器，在/root/目录下，执行java -jar hdfsClient-1.1.0.jar命令，开始运行实验。

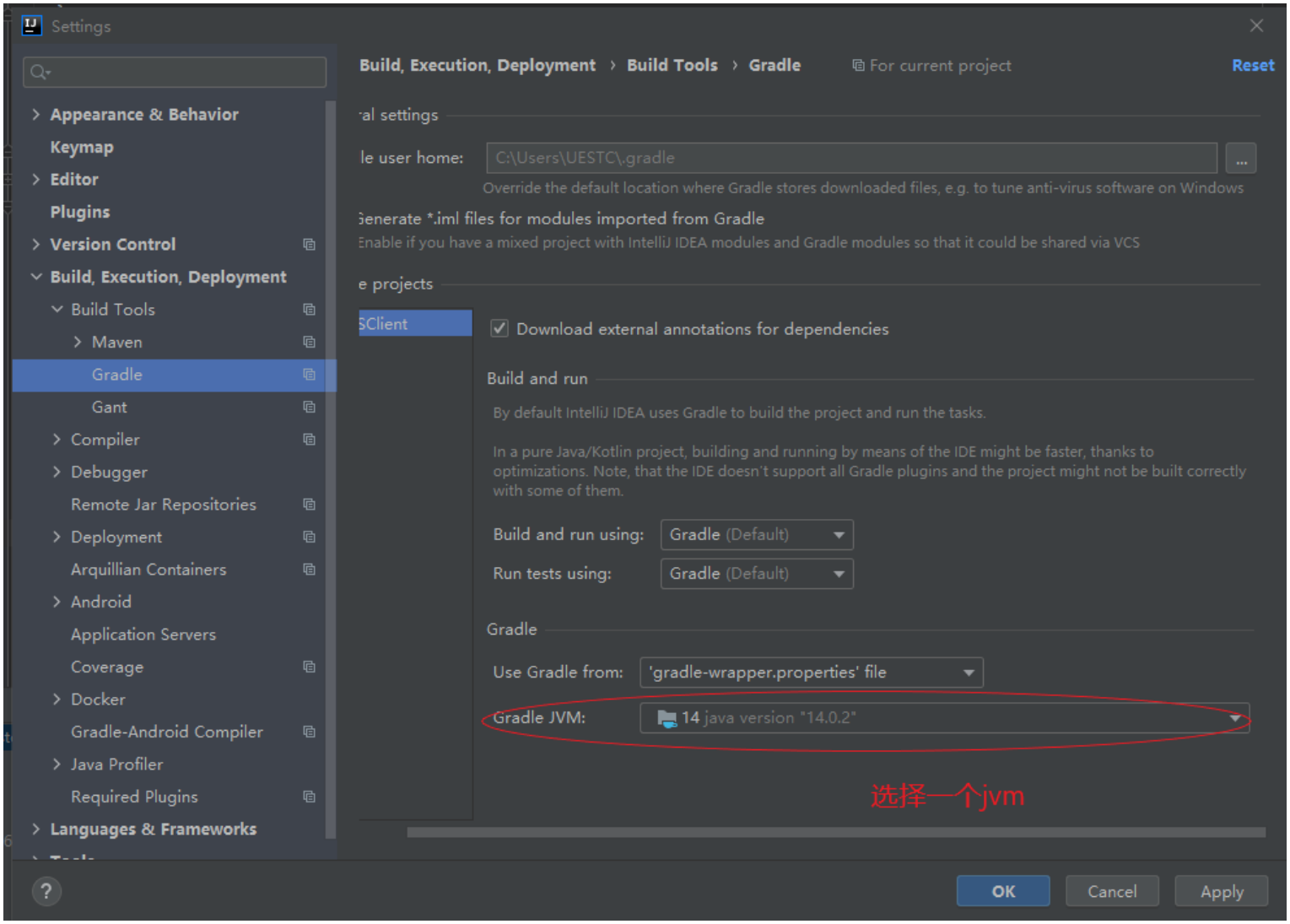
Java 实验环境


Welcome to IntelliJ IDEA
IntelliJ IDEA
Version 2020.2.1
New Project
Open or Import
Get from Version Control
Open File or Project
Hide path
D:\hadoopidea\HDFSClient\build.gradle
.gradle
.idea
build
gradle
libs
src
build.gradle
gradlew
gradlew.bat
hdfsclasspath
settings.gradle
HDFSClient.tar
hadoopubuntu
idea
java
Microsoft
Drag and drop a file into the space above to quickly locate it in the tree
OK
Cancel

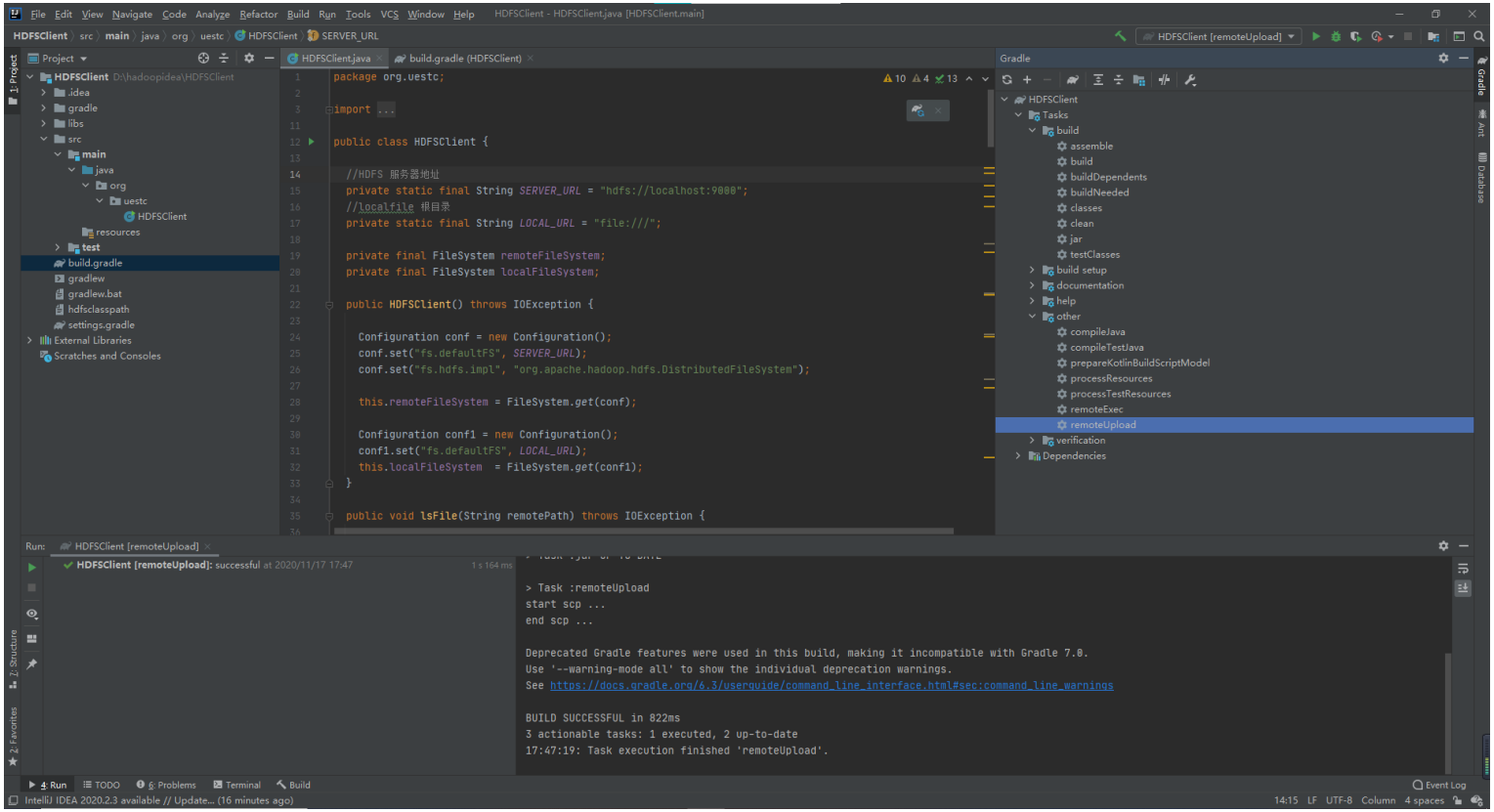
Java 实验环境


Welcome to IntelliJ IDEA
IntelliJ IDEA
Version 2020.2.1
New Project
Open or Import
Get from Version Control
Open Project
build.gradle is a project file.
Would you like to open this project?
Open as Project
Open as File
Cancel
Configure
Get Help

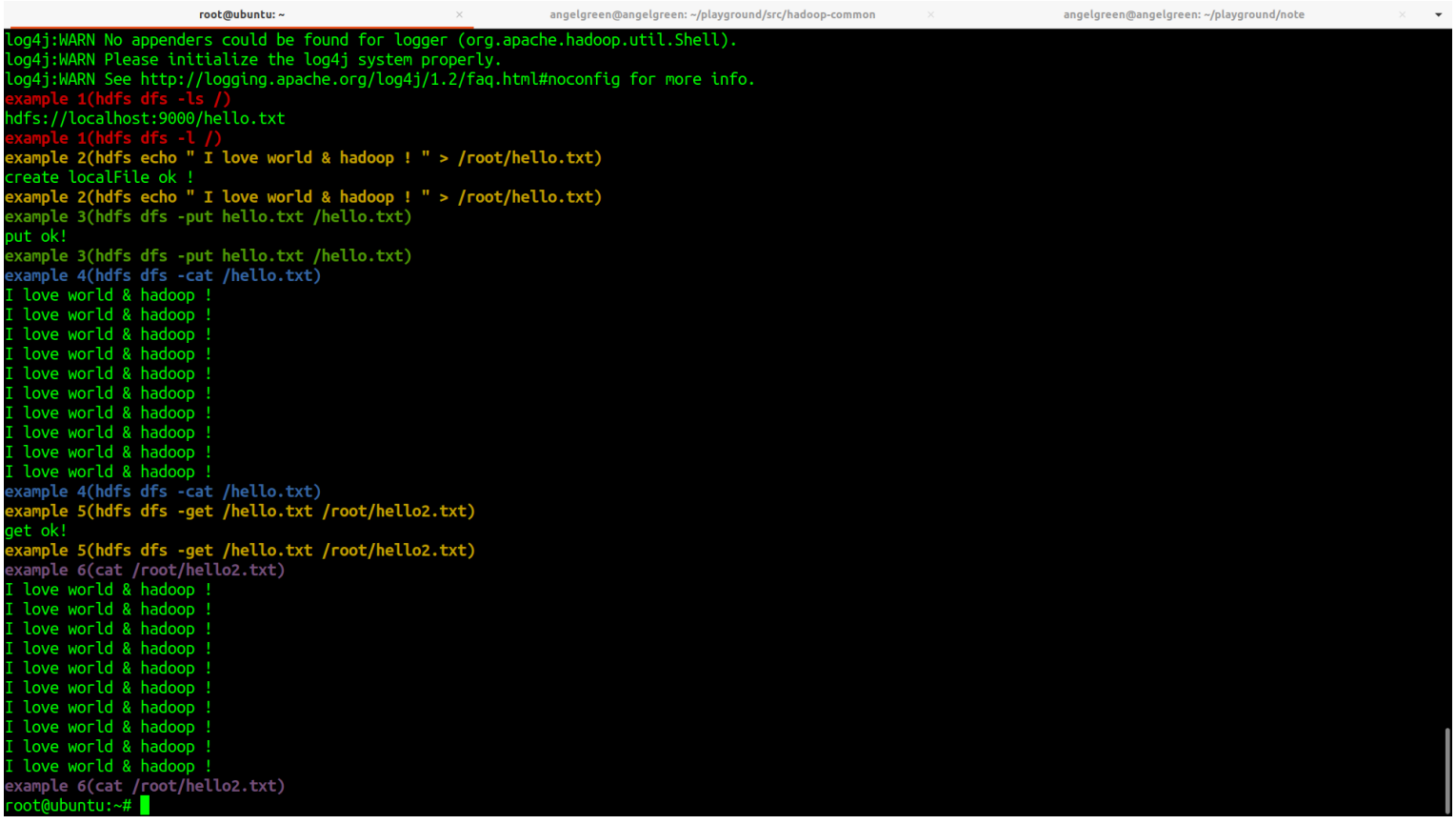
Java 实验环境


Settings
Build, Execution, Deployment › Build Tools › Gradle
For current project
Reset
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Build Tools
Maven
Gradle
Gant
Compiler
Debugger
Remote Jar Repositories
Deployment
Arquillian Containers
Android
Application Servers
Coverage
Docker
Gradle-Android Compiler
Java Profiler
Required Plugins
Languages & Frameworks
le user home:
C:\Users\UESTC\.gradle
Override the default location where Gradle stores downloaded files, e.g. to tune anti-virus software on Windows
Enable if you have a mixed project with IntelliJ IDEA modules and Gradle modules so that it could be shared via VCS
Download external annotations for dependencies
Build and run
By default IntelliJ IDEA uses Gradle to build the project and run the tasks.
In a pure Java/Kotlin project, building and running by means of the IDE might be faster, thanks to optimizations. Note, that the IDE doesn't support all Gradle plugins and the project might not be built correctly with some of them.
Build and run using:
Gradle (Default)
Run tests using:
Gradle (Default)
Gradle
Use Gradle from:
'gradle-wrapper.properties' file
Gradle JVM:
14 java version "14.0.2"
选择一个jvm
OK
Cancel
Apply

Java 实验环境


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HDFSClient - HDFSClient.java [HDFSClient.main]
HDFSClient › src › main › java › org › uestc › HDFSClient › SERVER_URL
HDFSClient [remoteUpload]
Project
HDFSClient D:\hadoopidea\HDFSClient
.idea
gradle
libs
src
main
java
org
uestc
HDFSClient
resources
test
build.gradle
gradlew
gradlew.bat
hdfsclasspath
settings.gradle
External Libraries
Scratches and Consoles
HDFSClient.java
build.gradle (HDFSClient)
package org.uestc;
import ...
public class HDFSClient {
//HDFS 服务器地址
private static final String SERVER_URL = "hdfs://localhost:9000";
//localfile 根目录
private static final String LOCAL_URL = "file:///";
private final FileSystem remoteFileSystem;
private final FileSystem localFileSystem;
public HDFSClient() throws IOException {
Configuration conf = new Configuration();
conf.set("fs.defaultFS", SERVER_URL);
conf.set("fs.hdfs.impl", "org.apache.hadoop.hdfs.DistributedFileSystem");
this.remoteFileSystem = FileSystem.get(conf);
Configuration conf1 = new Configuration();
conf1.set("fs.defaultFS", LOCAL_URL);
this.localFileSystem = FileSystem.get(conf1);
}
public void lsFile(String remotePath) throws IOException {
Gradle
HDFSClient
Tasks
build
assemble
build
buildDependents
buildNeeded
classes
clean
jar
testClasses
build setup
documentation
help
other
compileJava
compileTestJava
prepareKotlinBuildScriptModel
processResources
processTestResources
remoteExec
remoteUpload
verification
Dependencies
Run: HDFSClient [remoteUpload]
HDFSClient [remoteUpload]: successful at 2020/11/17 17:47
> Task :remoteUpload
start scp ...
end scp ...
Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.3/userguide/command_line_interface.html#sec:command_line_warnings
BUILD SUCCESSFUL in 822ms
3 actionable tasks: 1 executed, 2 up-to-date
17:47:19: Task execution finished 'remoteUpload'.
4: Run
TODO
6: Problems
Terminal
Build
Event Log
IntelliJ IDEA 2020.2.3 available // Update... (16 minutes ago)
14:15 LF UTF-8 Column 4 spaces

Java 实验环境


root@ubuntu: ~
angelgreen@angelgreen: ~/playground/src/hadoop-common
angelgreen@angelgreen: ~/playground/note
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
example 1(hdfs dfs -ls /)
hdfs://localhost:9000/hello.txt
example 1(hdfs dfs -l /)
example 2(hdfs echo " I love world & hadoop ! " > /root/hello.txt)
create localFile ok !
example 2(hdfs echo " I love world & hadoop ! " > /root/hello.txt)
example 3(hdfs dfs -put hello.txt /hello.txt)
put ok!
example 3(hdfs dfs -put hello.txt /hello.txt)
example 4(hdfs dfs -cat /hello.txt)
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
example 4(hdfs dfs -cat /hello.txt)
example 5(hdfs dfs -get /hello.txt /root/hello2.txt)
get ok!
example 5(hdfs dfs -get /hello.txt /root/hello2.txt)
example 6(cat /root/hello2.txt)
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
I love world & hadoop !
example 6(cat /root/hello2.txt)
root@ubuntu:~#

参考

[1] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.
The google file system.
In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[2] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler.
The hadoop distributed file system.
In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.