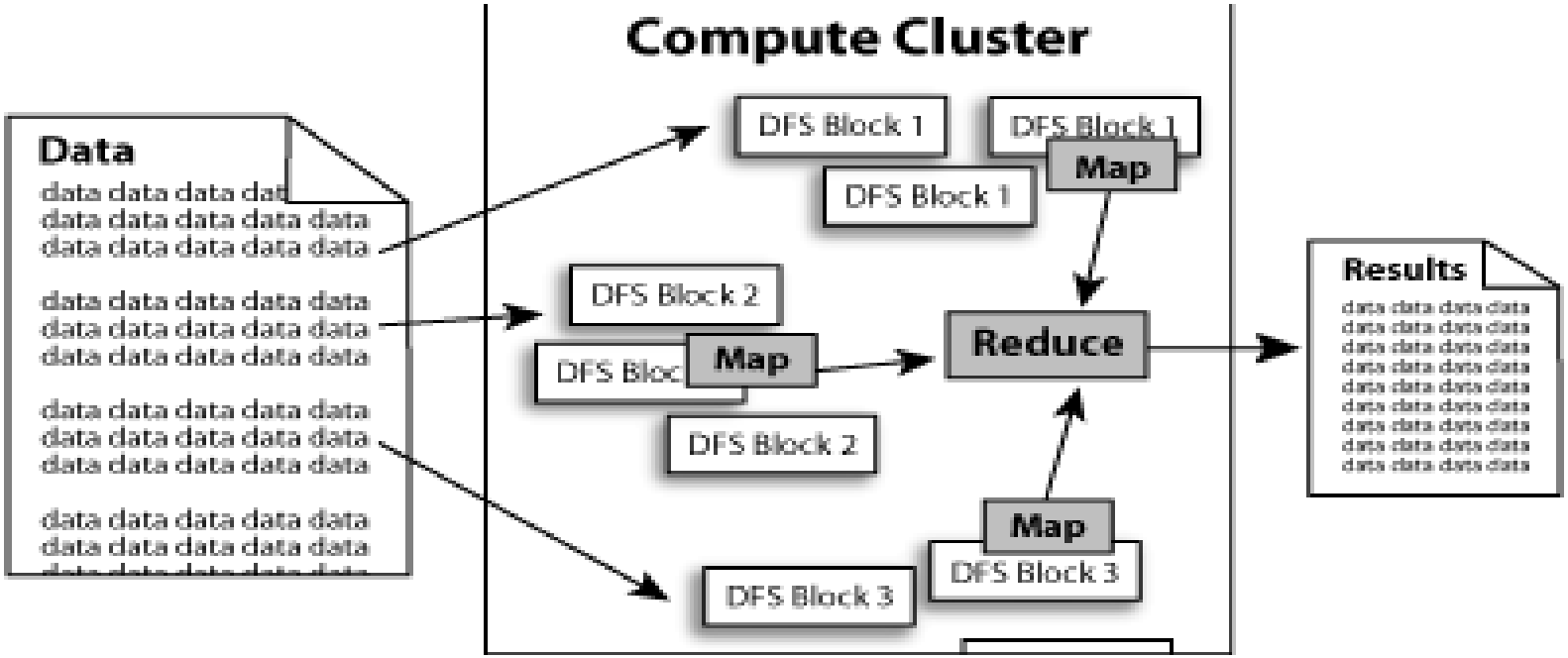
HDFS和HA

HDFS

HDFS简介

HDFS为了做到可靠性（reliability）创建了多份数据块（data blocks）的复制（replicas），并将它们放置在服务器群的计算节点中（compute nodes），MapReduce就可以在它们所在的节点上处理这些数据了。



HDFS适合做什么？

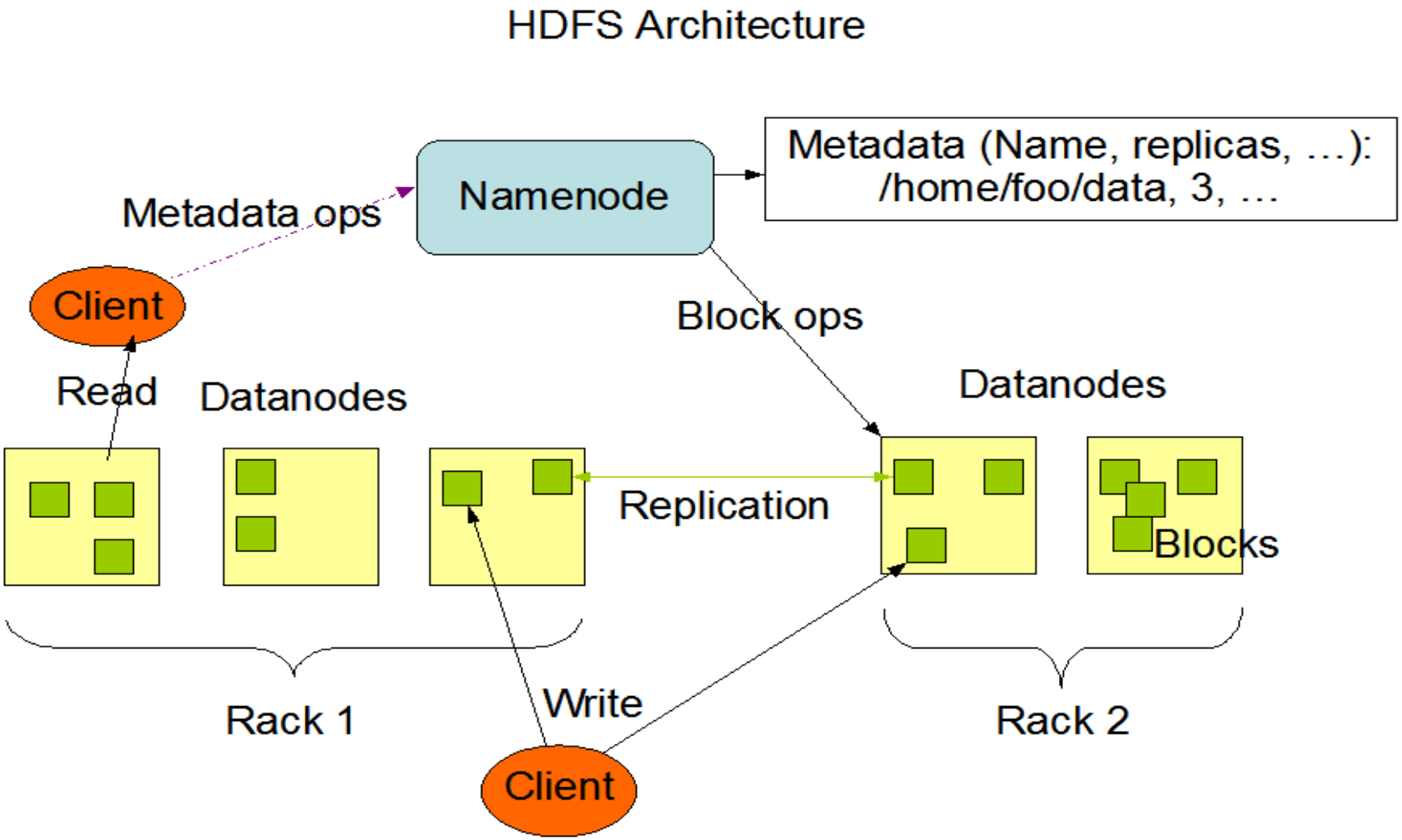
1. 存储并管理PB级数据
2. 处理非结构化数据
3. 注重数据处理的吞吐量（延迟不敏感）
4. 应用模式为：write-once-read-many存取模式

HDFS不适合做什么？

1. 存储小文件 (不建议使用)
2. 大量的随机读 (不建议使用)
3. 需要对文件的修改 (不支持)

思考 10PB 级别数据如何存储 ？

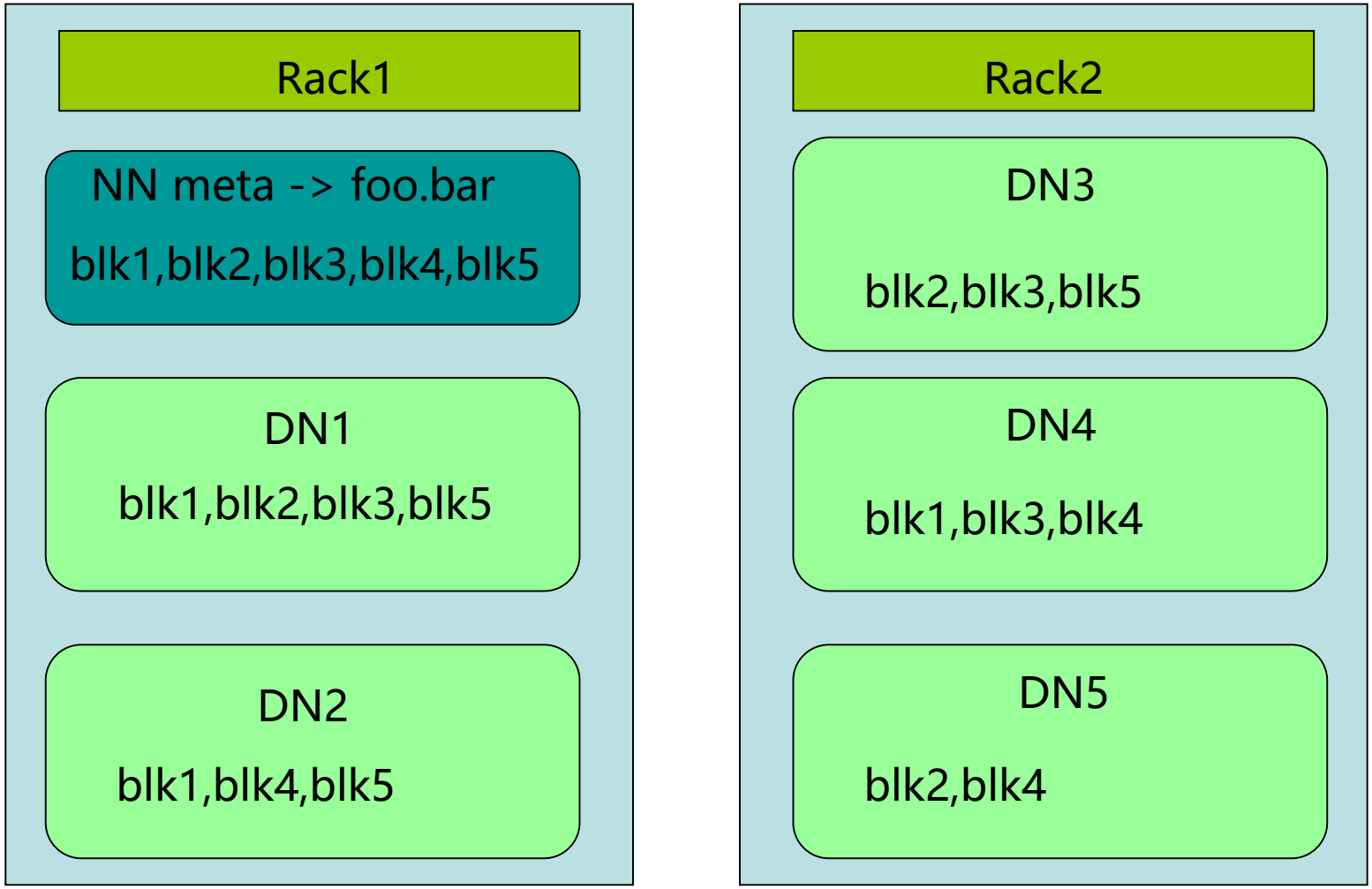
HDFS 基本结构


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Client
Rack 2

Block的副本放置策略

- ☆第一个副本：放置在上传文件的DN；如果是集群外提交，则随机挑选一台磁盘不太满，CPU不太忙的节点
- ☆第二个副本：放置在于第一个副本不同的机架的节点上
- ☆第三个副本：与第二个副本相同集群的节点
- ☆更多副本：随机节点

HDFS基础知识


Rack1
NN meta -> foo.bar
blk1,blk2,blk3,blk4,blk5
DN1
blk1,blk2,blk3,blk5
DN2
blk1,blk4,blk5
Rack2
DN3
blk2,blk3,blk5
DN4
blk1,blk3,blk4
DN5
blk2,blk4

设计目标

假设：
节点失效是常态

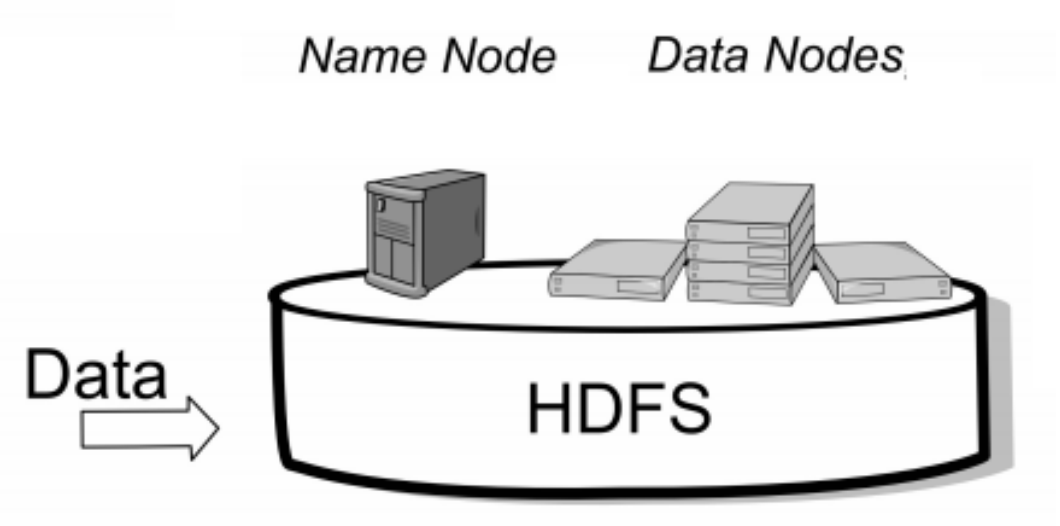
理想：
1. 任何一个节点失效，不影响HDFS服务
2. HDFS可以自动完成副本的复制

设计目标

目标：

- write-once-read-many存取模式
- 不支持文件并发写入
- 不支持文件修改

HDFS主要组件的功能



NameNode

- 存储元数据
- 元数据保存在内存中
- 保存文件,block ，datanode之间的映射关系

DataNode

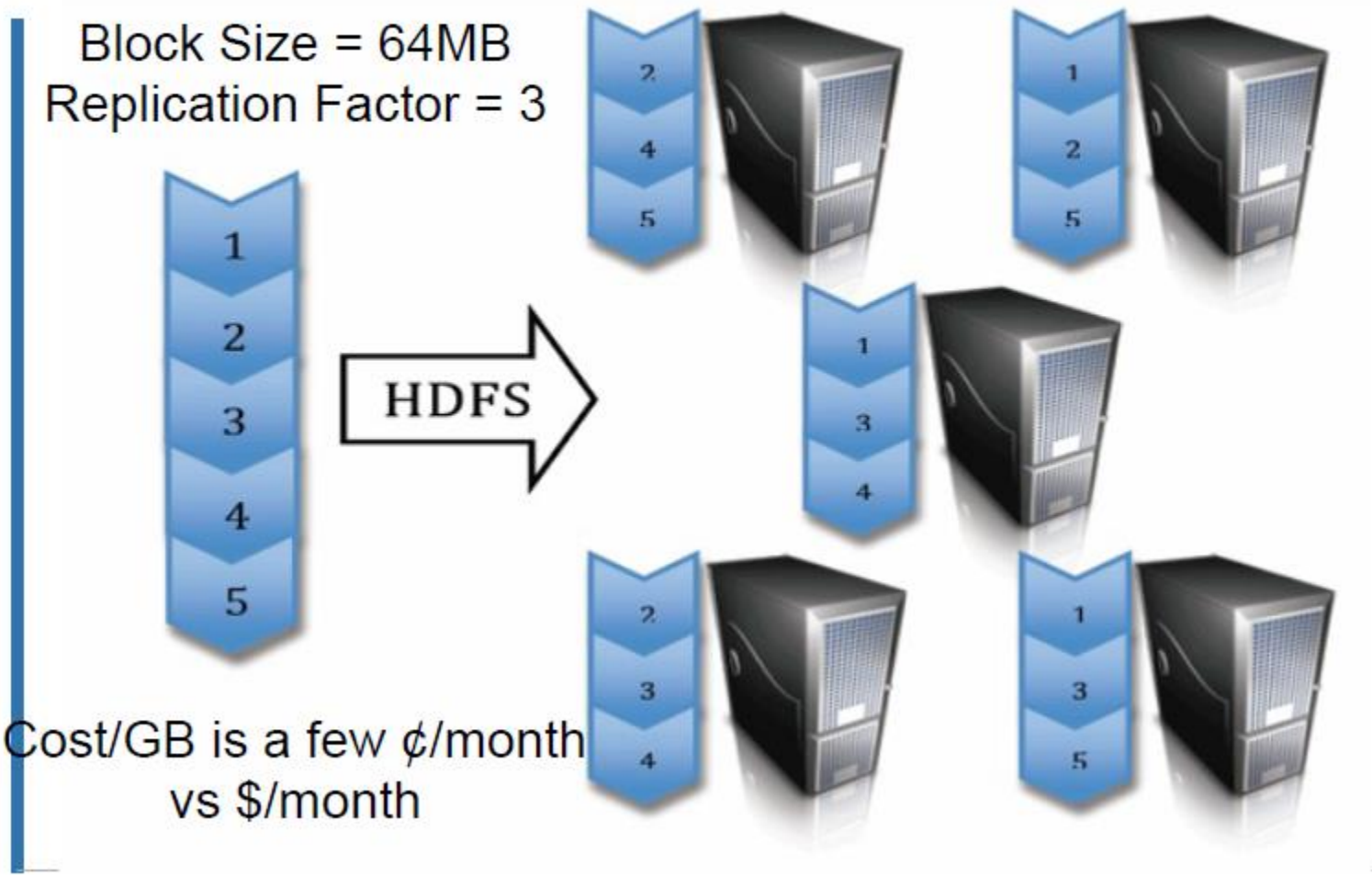
- 存储文件内容
- 文件内容保存在磁盘
- 维护了block id到datanode本地文件的映射关系

文件

- 文件切分成块（默认大小64M），以块为单位，每个块有多个副本存储在不同的机器上，副本数可在文件生成时指定（默认3）
- NameNode是主节点，存储文件的元数据如文件名，文件目录结构，文件属性（生成时间,副本数,文件权限），以及每个文件的块列表以及块所在的DataNode等等
- DataNode在本地文件系统存储文件块数据，以及块数据的校验

文件

HDFS: Hadoop Distributed File System



NameNode

- Namenode是一个中心服务器，单一节点，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问
- 文件操作，NameNode负责文件元数据的操作，DataNode负责处理文件内容的读写请求，数据流不经过NameNode，只会询问它跟那个DataNode联系

NameNode

•副本存放在那些DataNode上由NameNode来控制，根据全局情况做出块放置决定，读取文件时NameNode尽量让用户先读取最近的副本，降低带块消耗和读取时延

NameNode

- Namenode全权管理数据块的复制，它周期性地从集群中的每个Datanode接收心跳信号和块状态报告(Blockreport)。接收到心跳信号意味着该Datanode节点工作正常。块状态报告包含了一个该Datanode上所有数据块的列表。

NameNode(NN)

metadata

File.txt=
Blk A:
DN1, DN5, DN6

Blk B:
DN7, DN1, DN2

Blk C:
DN5, DN8,DN9

块存储结构

```
/data/cache1/dfs/nn/
|-- current
|   |-- VERSION
|   |-- edits
|   |-- fsimage
|   `-- fstime
|-- image
|   `-- fsimage
|-- in_use.lock
`-- previous.checkpoint
    |-- VERSION
    |-- edits
    |-- fsimage
    `-- fstime
```

metadate物理

DataNode

- 一个数据块在DataNode以文件存储在磁盘上，包括两个文件，一个是数据本身，一个是元数据包括数据块的长度，块数据的校验和，以及时间戳
- DataNode启动后向NameNode注册，通过后，周期性（1小时）的向NameNode上报所有的块信息。

DataNode

- 心跳是每3秒一次，心跳返回结果带有NameNode给该DataNode的命令如复制块数据到另一台机器，或删除某个数据块。如果超过10分钟没有收到某个DataNode 的心跳，则认为该节点不可用。
- 集群运行中可以安全加入和退出一些机器

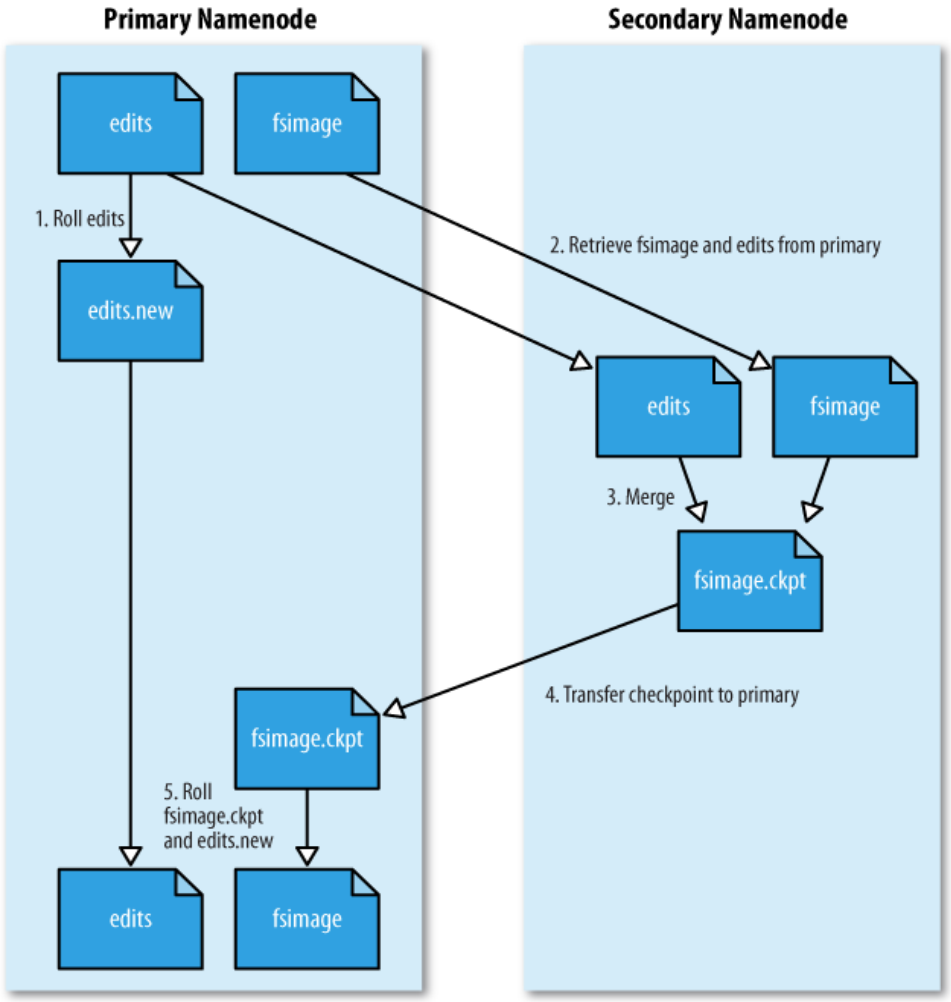
DataNode

- 保存Block
- 启动DN线程的时候会向NN汇报block信息
- 通过向NN发送心跳保持与其联系（3秒一次），如果NN 10分钟没有收到DN的心跳，则认为其已经lost，并copy其上的block到其它DN

```
-bash-3.2$ sudo ls -l /data/cache1/dfs/dn/current/
total 594992
-rw-r--r--  1 hdfs hadoop     20495 Apr 29 21:11 blk_-1265257027172478037
-rw-r--r--  1 hdfs hadoop       171 Apr 29 21:11 blk_-1265257027172478037_1896.meta
-rw-r--r--  1 hdfs hadoop       282 Apr 29 21:10 blk_1355466200351616098
-rw-r--r--  1 hdfs hadoop        11 Apr 29 21:10 blk_1355466200351616098_1411.meta
-rw-r--r--  1 hdfs hadoop  32201859 Apr 29 21:13 blk_-1363349205939896111
-rw-r--r--  1 hdfs hadoop    251587 Apr 29 21:13 blk_-1363349205939896111_2355.meta
-rw-r--r--  1 hdfs hadoop     20349 Apr 29 21:11 blk_-1486089392719042041
-rw-r--r--  1 hdfs hadoop       167 Apr 29 21:11 blk_-1486089392719042041_1593.meta
-rw-r--r--  1 hdfs hadoop   1222666 Apr 29 21:12 blk_1841498388503862976
-rw-r--r--  1 hdfs hadoop      9563 Apr 29 21:12 blk_1841498388503862976_2595.meta
-rw-r--r--  1 hdfs hadoop   9250764 May 25 08:22 blk_1922063674315104428
-rw-r--r--  1 hdfs hadoop     72279 May 25 08:22 blk_1922063674315104428_32513112.meta
-rw-r--r--  1 hdfs hadoop   4509245 Apr 29 21:15 blk_-2039669051003294713
-rw-r--r--  1 hdfs hadoop     35239 Apr 29 21:15 blk_-2039669051003294713_4346.meta
```

SecondaryNameNode(SNN)

根据时间
根据过edits log大小
只是NN的工具
没有failover机制



数据损坏(corruption)处理

- 当DN读取block的时候，它会计算checksum；
- 如果计算后的checksum，与block创建时值不一样，说明该block已经损坏。
- client读取其它DN上的block；NN标记该块已经损坏，然后复制block达到预期设置的文件备份数；
- DN在其文件创建后三周验证其checksum。

HDFS文件权限

- 与Linux文件权限类似
- r: read; w:write; x:execute，权限x对于文件忽略，对于文件夹表示是否允许访问其内容
- 如果Linux系统用户zhangsan使用hadoop命令创建一个文件，那么这个文件在HDFS中owner是zhangsan
- HDFS的权限目的：阻止好人做错事，而不是阻止坏人做坏事。HDFS相信，你告诉我你是谁，我就认为你是谁

HDFS文件存储

两个文件，一个文件156M，一个文件128M
在HDFS里面怎么存储？

--Block为64MB

--replication默认拷贝3份

HDFS开发常用命令

- 创建一个文件夹？
- 上传一个文件？
- 删除一个文件和文件夹？
- 查看一个文件夹里面有哪些文件？
- 查看某个文件的内容？

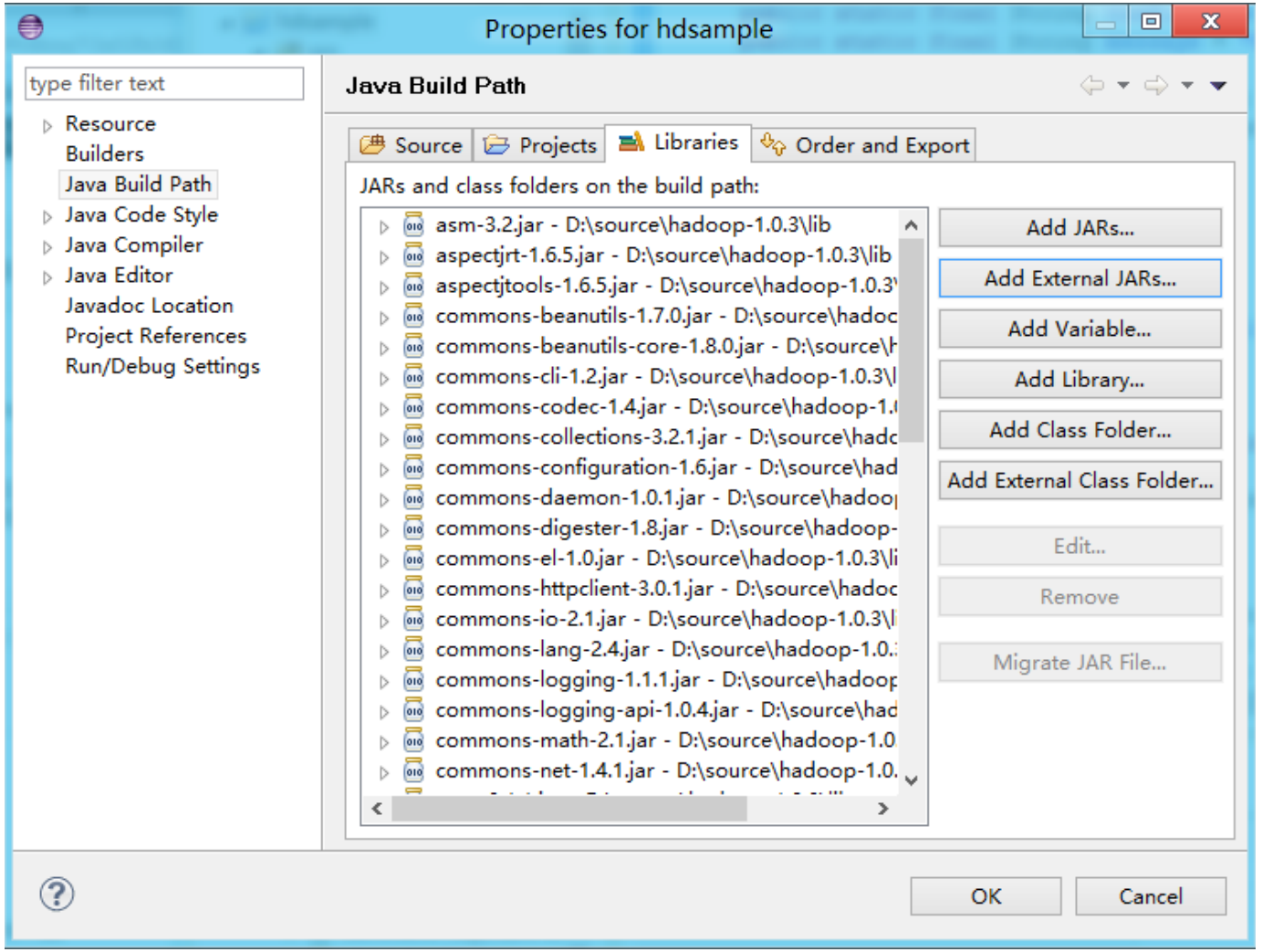
程序的编译过程

如何在eclipse中编写和编译运行HDFS的读写程序：

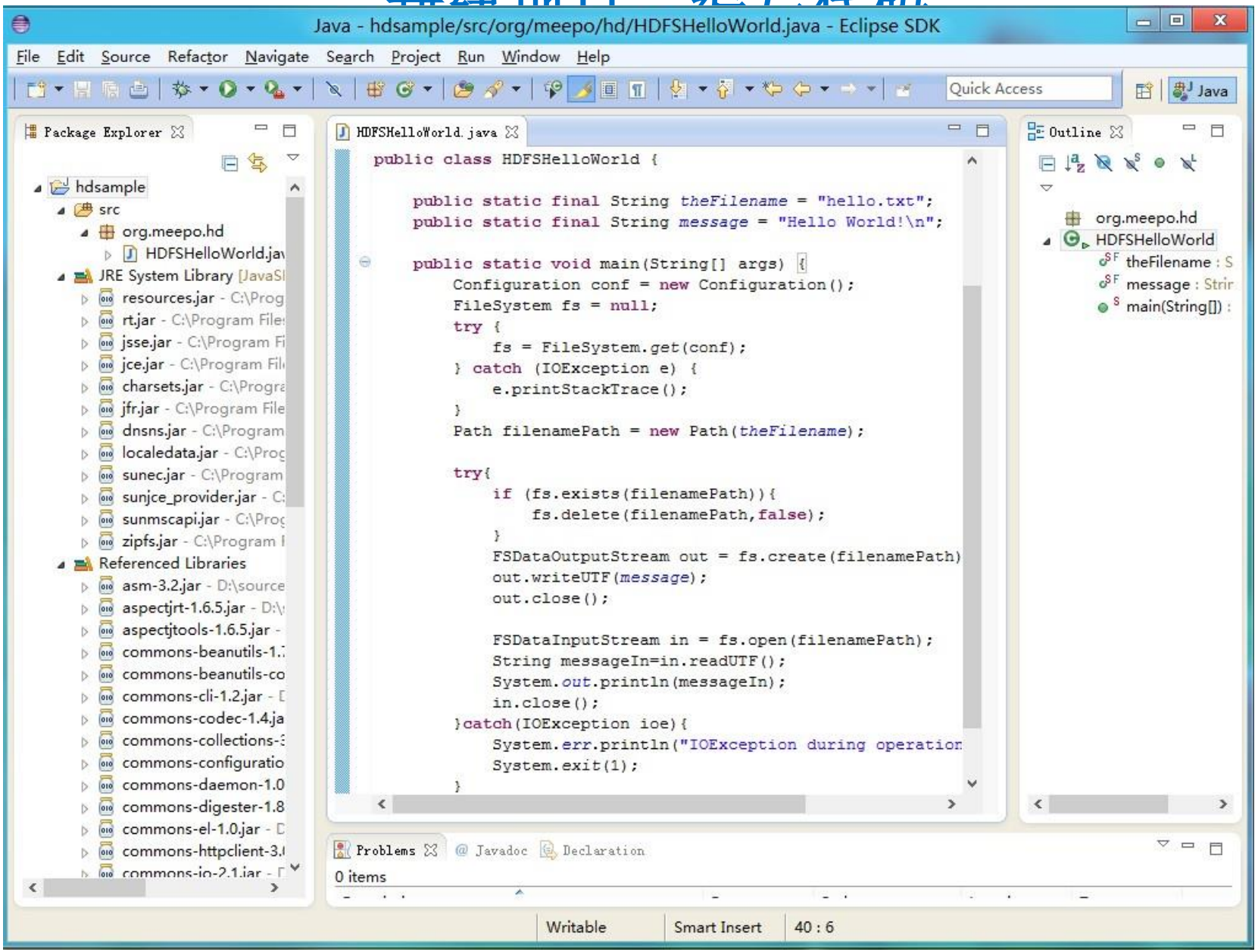
hadoop-core-x.x.x.jar和它所依赖的库导入到eclipse工程的build path中。这个首先需要先下载解压hadoop包。然后在eclipse中，通过右键工程->属性->Java Build Path->Libraries->Add External JARs，在解压后的hadoop目 录中点选hadoop-core-x.x.x.jar和lib目录下所有的.jar即可。

配置环境

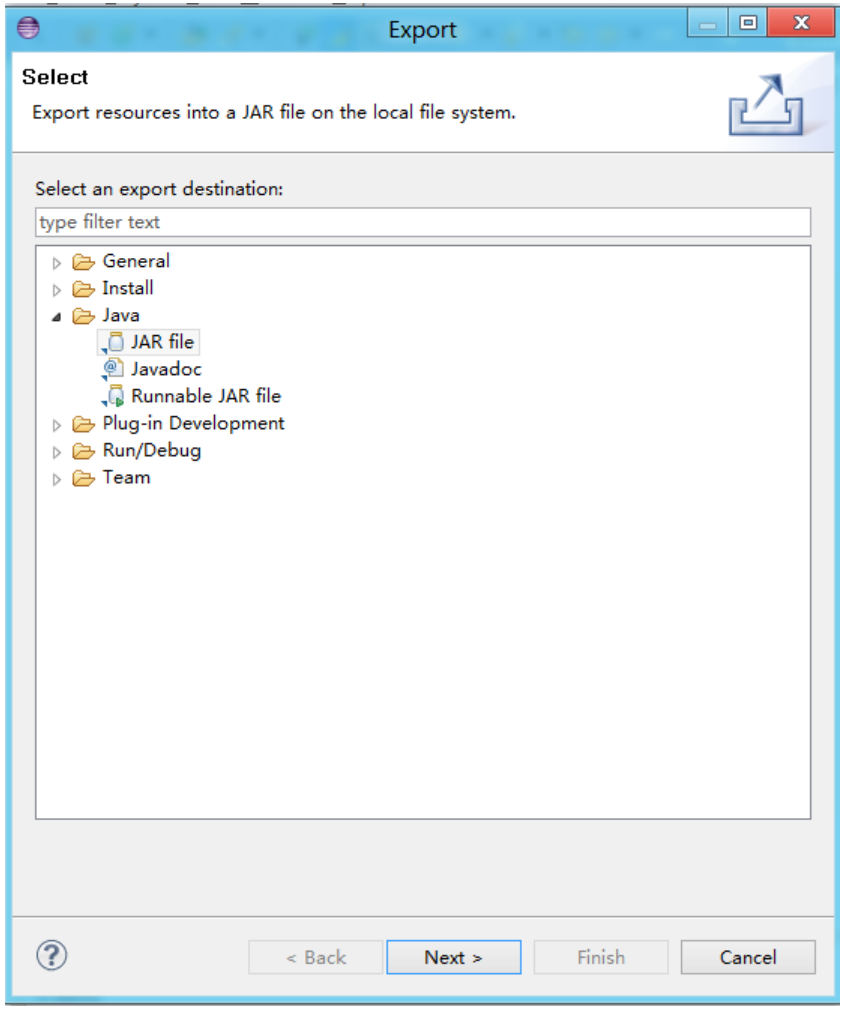
下载hadoop安装包，解压缩之后将所需要的jar包通过Add External Jars加入 项目中

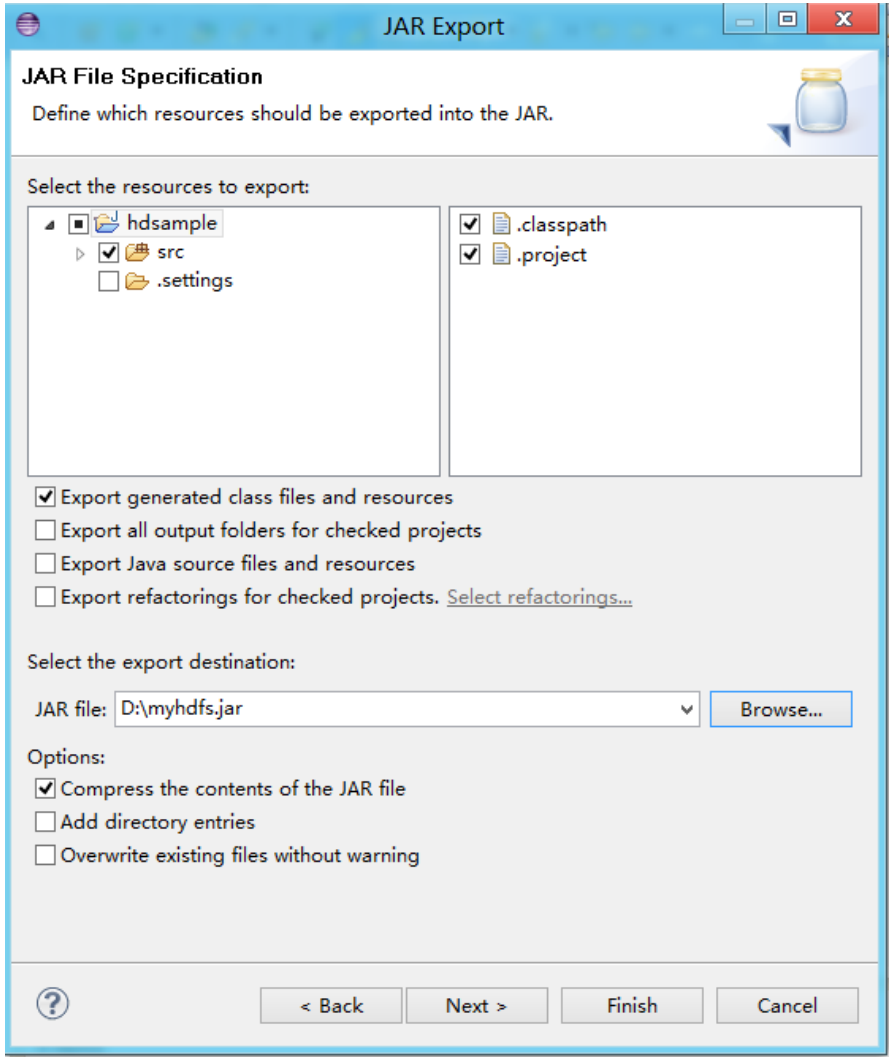


新建项目，编写代码


Java - hdsample/src/org/meepo/hd/HDFSHelloWorld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
hdsample
src
org.meepo.hd
HDFSHelloWorld.jav
JRE System Library [JavaSE
resources.jar - C:\Prog
rt.jar - C:\Program File
jsse.jar - C:\Program Fi
jce.jar - C:\Program Fil
charsets.jar - C:\Progra
jfr.jar - C:\Program File
dnsns.jar - C:\Program
localedata.jar - C:\Prog
sunec.jar - C:\Program
sunjce_provider.jar - C:
sunmscapi.jar - C:\Prog
zipfs.jar - C:\Program
Referenced Libraries
asm-3.2.jar - D:\source
aspectjrt-1.6.5.jar - D:\
aspectjtools-1.6.5.jar -
commons-beanutils-1.
commons-beanutils-co
commons-cli-1.2.jar - D
commons-codec-1.4.ja
commons-collections-3
commons-configuratio
commons-daemon-1.0
commons-digester-1.8
commons-el-1.0.jar - D
commons-httpclient-3.
commons-io-2.1.jar - D
HDFSHelloWorld.java
public class HDFSHelloWorld {
public static final String theFilename = "hello.txt";
public static final String message = "Hello World!\n";
public static void main(String[] args) {
Configuration conf = new Configuration();
FileSystem fs = null;
try {
fs = FileSystem.get(conf);
} catch (IOException e) {
e.printStackTrace();
}
Path filenamePath = new Path(theFilename);
try{
if (fs.exists(filenamePath)){
fs.delete(filenamePath,false);
}
FSDataOutputStream out = fs.create(filenamePath)
out.writeUTF(message);
out.close();
FSDataInputStream in = fs.open(filenamePath);
String messageIn=in.readUTF();
System.out.println(messageIn);
in.close();
}catch(IOException ioe){
System.err.println("IOException during operation
System.exit(1);
}
Outline
org.meepo.hd
HDFSHelloWorld
theFilename : S
message : Strir
main(String[]) :
Problems
Javadoc
Declaration
0 items
Writable
Smart Insert
40 : 6

导出jar文件


Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
Install
Java
JAR file
Javadoc
Runnable JAR file
Plug-in Development
Run/Debug
Team
< Back
Next >
Finish
Cancel


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
hdsample
src
.settings
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: D:\myhdfs.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Finish
Cancel

HDFS程序设计

HDFS核心思想

文件副本　分片保存

一次写入　多次读取

顺序写入　流式顺序读取

HDFS文件系统的特征

- 存储极大数目的信息（terabytes or petabytes），将数据 保存到大量的节点当中。支持很大单个文件。

- 提供数据的高可靠性，单个或者多个节点不工作，对系统不会
- 造成任何影响，数据仍然可用。

- 提供对这些信息的快速访问，并提供可扩展的方式。

- HDFS是针对MapReduce设计的，使得数据尽可能根据其本
- 地局部性进行访问与计算。

HDFS的设计

基于块的文件存储 块进行复制的形式放置，按照块的方式随机选择存储节点

副本的默认数目是3，并可以通过配置文件进行制定

默认的块的大小是64MB

- 减少元数据的量
- 有利于顺序读写（在磁盘上数据顺序存放）

依据Google File System的设计进行实现

HDFS系统结构中的主要模块

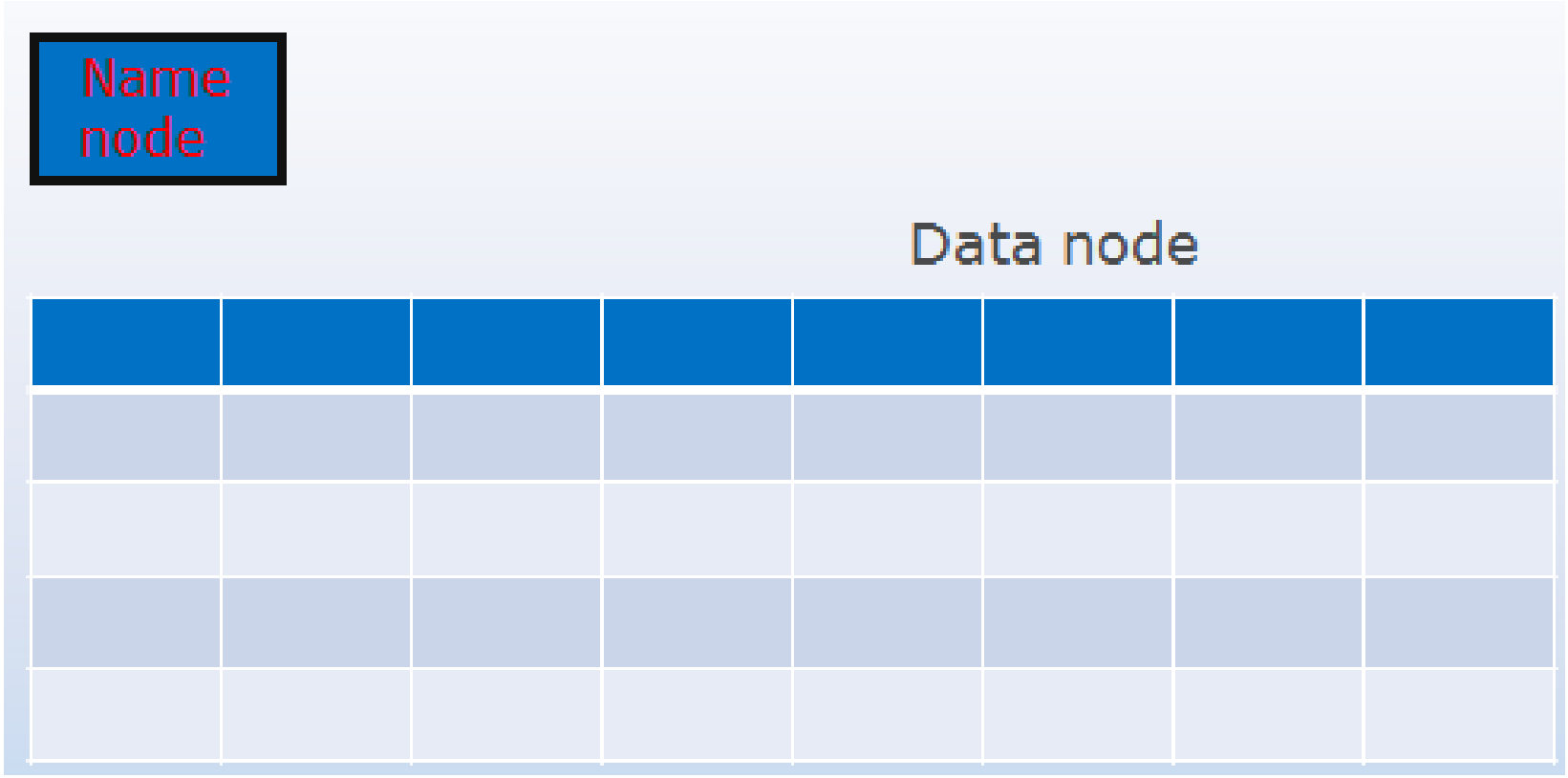
- HDFS系统结构：使用了Master与Worker的结构，具有一个单个的Master 以及多个Worker的结构，Master被称为名字节点NameNode，Worker被称 为数据节点DataNode

- NameNode：
- 名字服务器运行在一个Master的服务器之上
- NameNode管理所有文件系统的名字空间（元数据）以及协调管理客户端对于数据 的访问
- NameNode存储以及调整整个文件系统中的元数据

- DataNode：
- 运行在集群中的绝大多数节点之上
- 管理文件系统中的数据存储
- 从NameNode中接收命令，并对数据进行组织管理上的操作（如负载均衡）
- 响应文件系统客户端的读写访问命令，提供数据服务

- 数据通信的协议使用TCP/IP协议，使用RPC进行远程信息访问请求

Master与Worker



Name Node

1、管理HDFS集群中文件系统的名字空间（Namespace），例如打开文件系统、关闭文件系统、重命名文件或者目录等；另外，对任何请求对文件系统名字空间或者属性进行修改的操作，都被Namenode记录下来。

2、管理客户端对HDFS集群中的文件系统中的文件的访问文件系统客户端与Namenode交互的过程中，取到了所请求的文件块所对应的Datanode结点，执行文件的读写操作。Namenode结点负责确定指定的文件块到具体的Datanode结点的映射关系。

3、管理Datanode结点的状态报告，包括Datanode结点的健康状态报告和其所在结点上数据块状态报告，以便能够及时处理失效的数据结点。

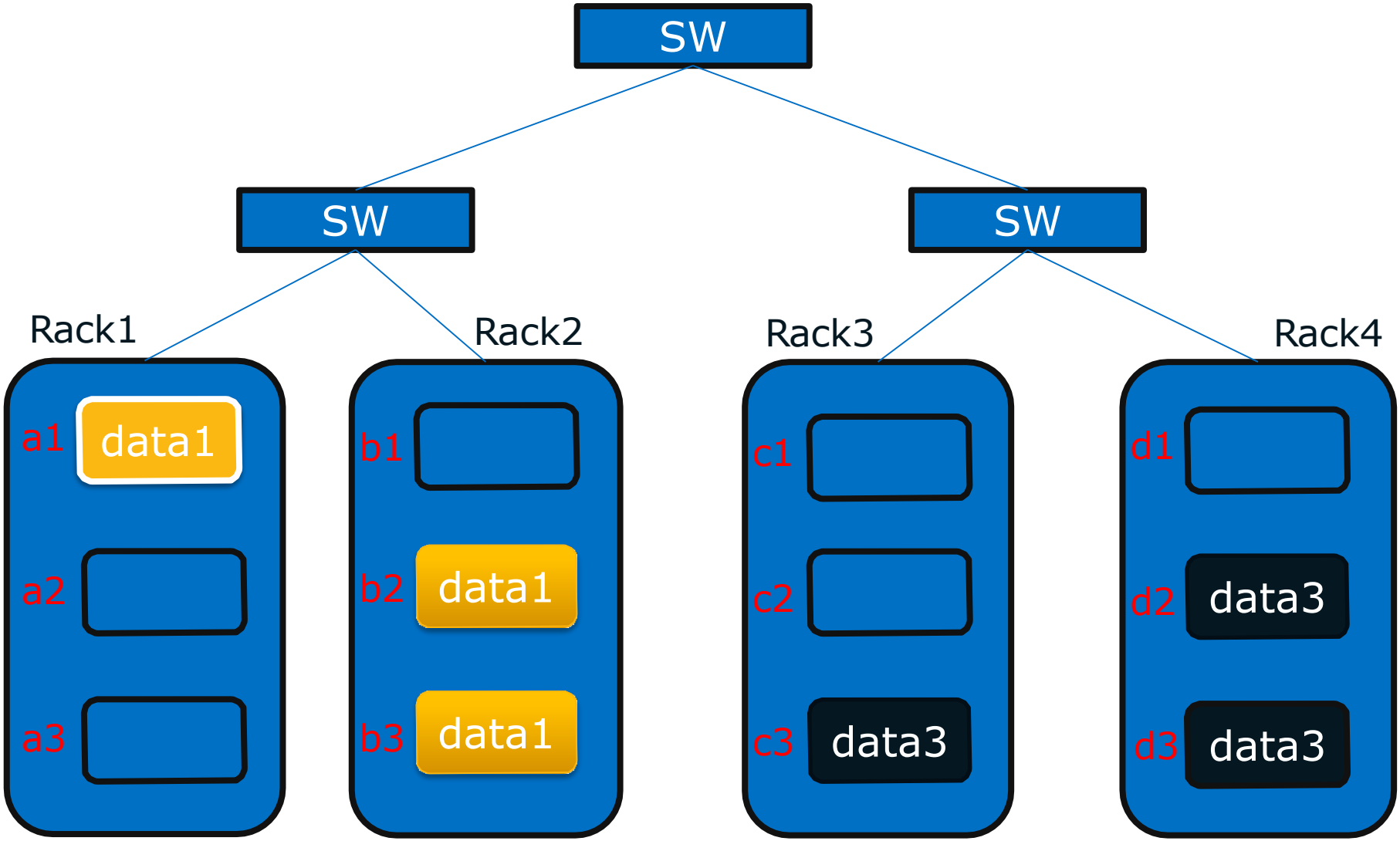
Data Node

1、负责管理它所在结点上存储的数据的读写，一般是文件系统客户端需要请求对指定数据结点进行读写操作，Datanode作为数据结点的服务进程来与文件系统客户端打交道。

2、向Namenode结点报告状态。每个Datanode结点会周期性地向Namenode发送心跳信号和文件块状态报告。

3、执行数据的流水线复制。当文件系统客户端从Namenode服务器进程获取到要进行复制的数据块列表后，完成文件块及其块副本的流水线复制。

HDFS数据分布设计


SW
SW
SW
Rack1
Rack2
Rack3
Rack4
a1
data1
a2
a3
b1
b2
data1
b3
data1
c1
c2
c3
data3
d1
d2
data3
d3
data3

HDFS可靠性

- 对于DataNode节点失效的问题：
- • 通过心跳信息来获得节点的情况
- • NameNode可以进行节点失效时候的数据重新分布
- 集群数据的重新进行负载均衡 数据的完整性通过数据校验

获得 如果元数据磁盘失效，处理的办法：使用多个元数据的

映像文

- 件FsImag，使用元数据的修改日志EditLog，以及通过检查 点的方式进行恢复

- 快照方式：快照方式可以进行数据的回滚

HDFS的交互操作

HDFS简单交互交互

someone@anynode:hadoop$ bin/hadoop dfs –ls　列目录中的内容

someone@anynode:hadoop$ bin/hadoop dfs -ls / 列根目录中的内容
Found 2 items
drwxr-xr-x - hadoop supergroup 0 2008-09-20 19:40 /hadoop
drwxr-xr-x - hadoop supergroup 0 2008-09-20 20:08 /tmp

someone@anynode:hadoop$ bin/hadoop dfs -mkdir /user 建目录

someone@anynode:hadoop$ bin/hadoop dfs -mkdir /user/someone 建用户目录，依据HDFS用户以及权限模型需要在/user下面为每一个用户建立目录

someone@anynode:hadoop$ bin/hadoop dfs -put /home/someone/interestingFile.txt /user/*yourUserName*/　从本地复制数据到HDfs文件系统中，-put等同于-copyFromLocal

Put上传整个目录
Bin/hadoop –put directory destination

Cat and Get

打印HDFS中文件foo的内容

bin/hadoop dfs – cat foo

将HDFS中的文件foo拷贝到本地

bin/hadoop dfs – get foo localFoo

HDFS交互命令行

-ls path	列目录信息，包括文件名，权限，拥有者，大小以及修改时间
-lsr path	与-ls类似，同时还要列出子目录中的内容
-du path	显示某一个目录的磁盘使用情况（每个文件，单位为字节数）
-dus path	与-du类似，但是打印一个概要信息
-mv src dest	在HDFS内部移动目录或者文件
-cp src dest	在HDFS内部复制文件或者目录

HDFS交互命令行

-rm path	删除文件或者空的目录
-rmr path	删除文件或者目录，删除目录时递归删除所有子目录以及文件
-put localSrc dest	拷贝本地文件或者目录到HDFS中
-copyFromLocal localSrc dest	等同于-copy
-moveFromLocal localSrc dest	拷贝数据到HDFS，并且在成功后删除本地数据

HDFS编程访问接口

在程序中使用HDFS接口

HDFS接口包括：

- 命令行接口
- Hadoop MapReduce Job的隐含的输入（在这种情况下，只需要将任务交给MapReduce系统即可，会自动调度HDFS服务来提供数据）
- Java程序直接操作
- libhdfs从c/c++程序中操作

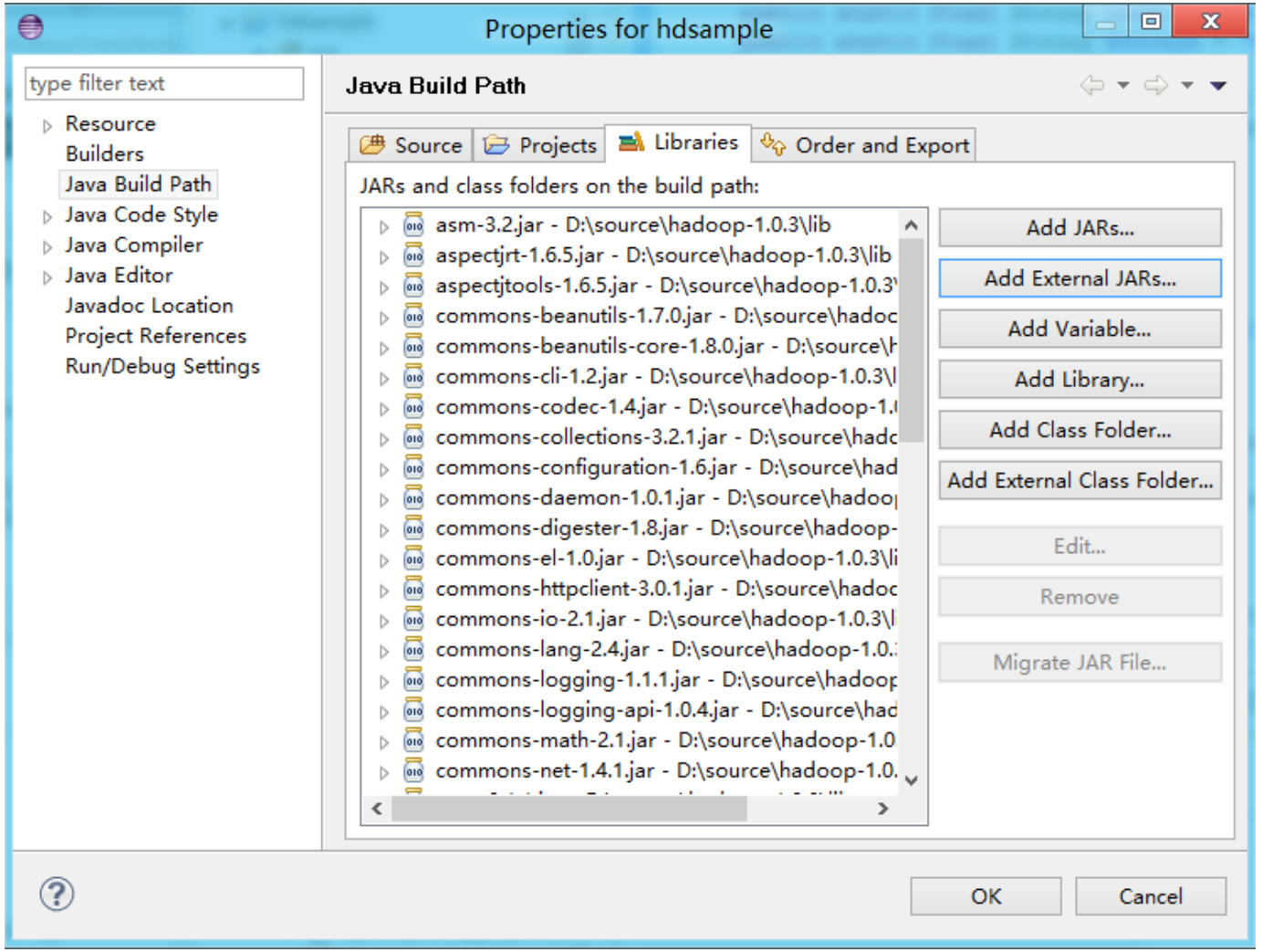
程序的编译过程

如何在eclipse中编写和编译运行HDFS的读写程序：

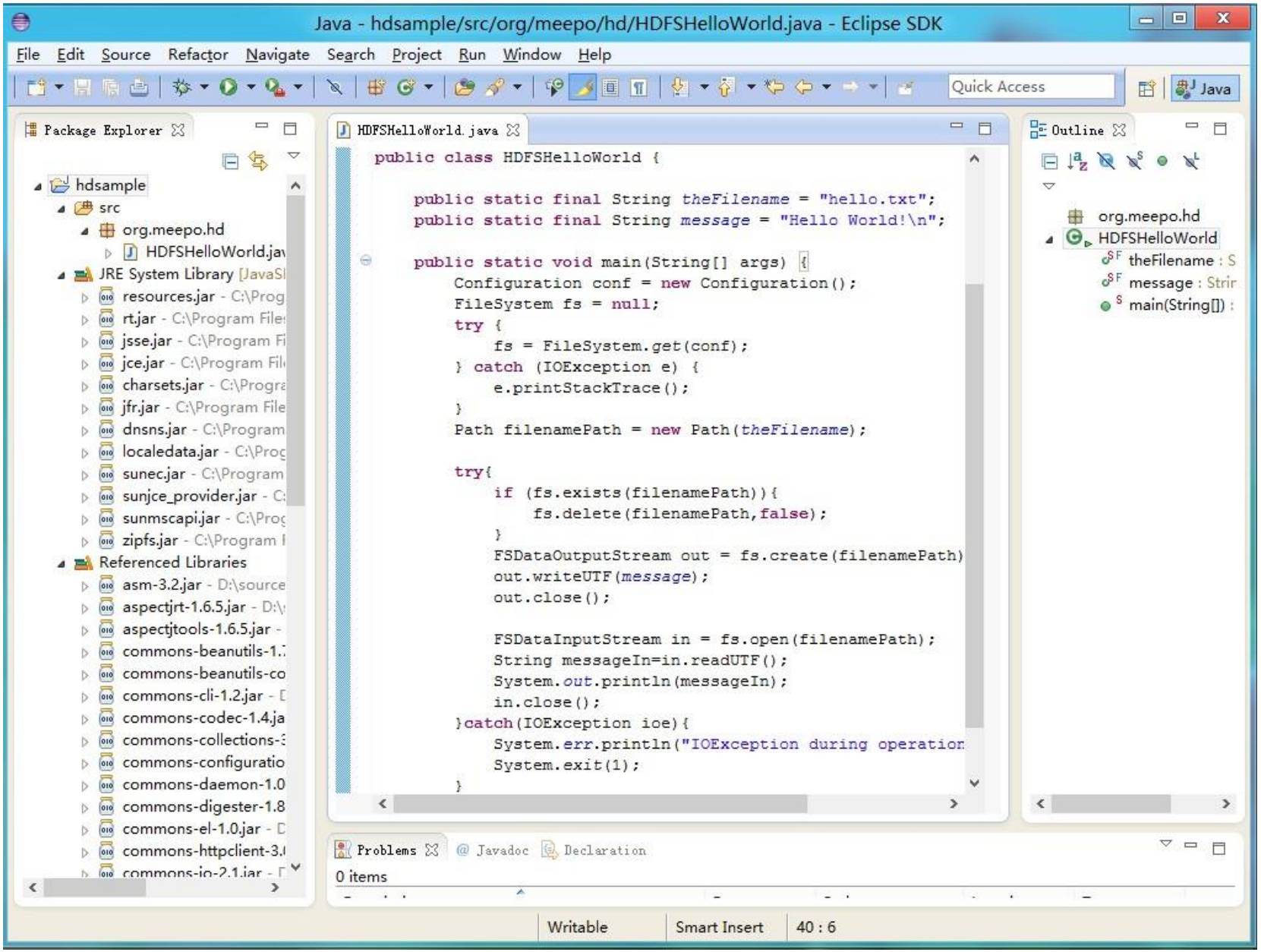
hadoop-core-x.x.x.jar和它所依赖的库导入到eclipse工程的build path中。这个首先需要先下载解压hadoop包。然后在eclipse中，通过右键工程->属性->Java Build Path->Libraries->Add External JARs，在解压后的hadoop目 录中点选hadoop-core-x.x.x.jar和lib目录下所有的.jar即可。

配置环境

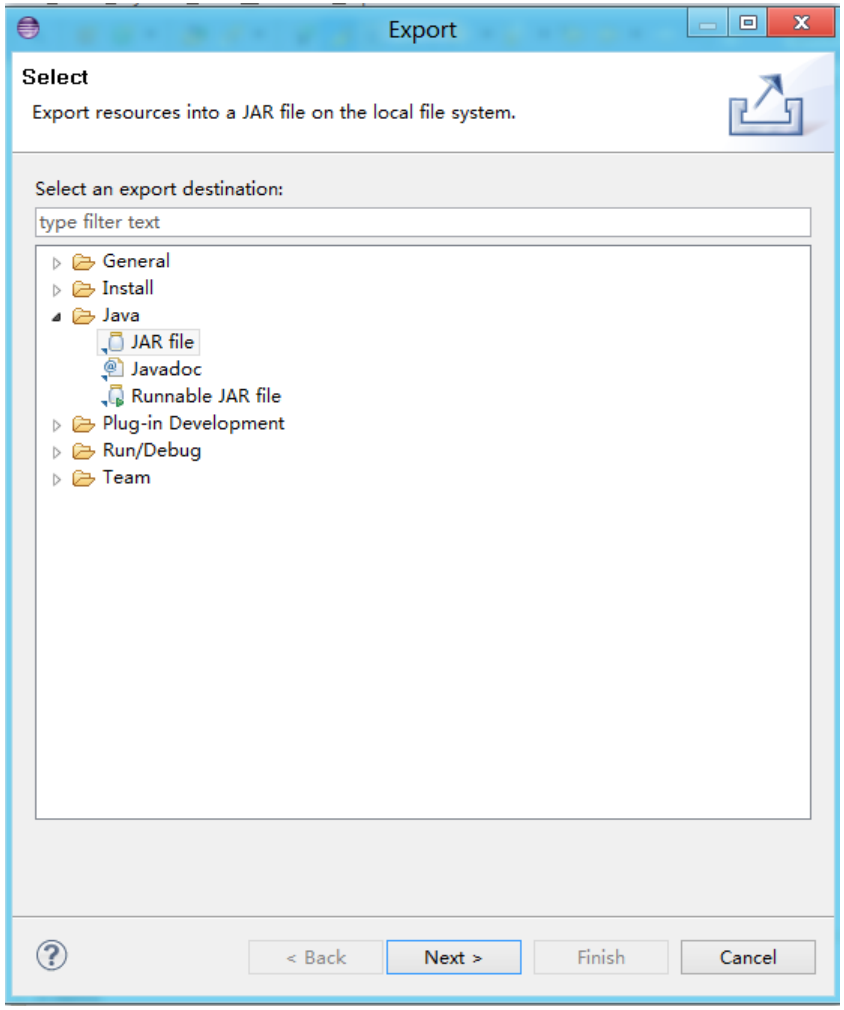
下载hadoop安装包，解压缩之后将所需要的jar包通过Add External Jars加入 项目中

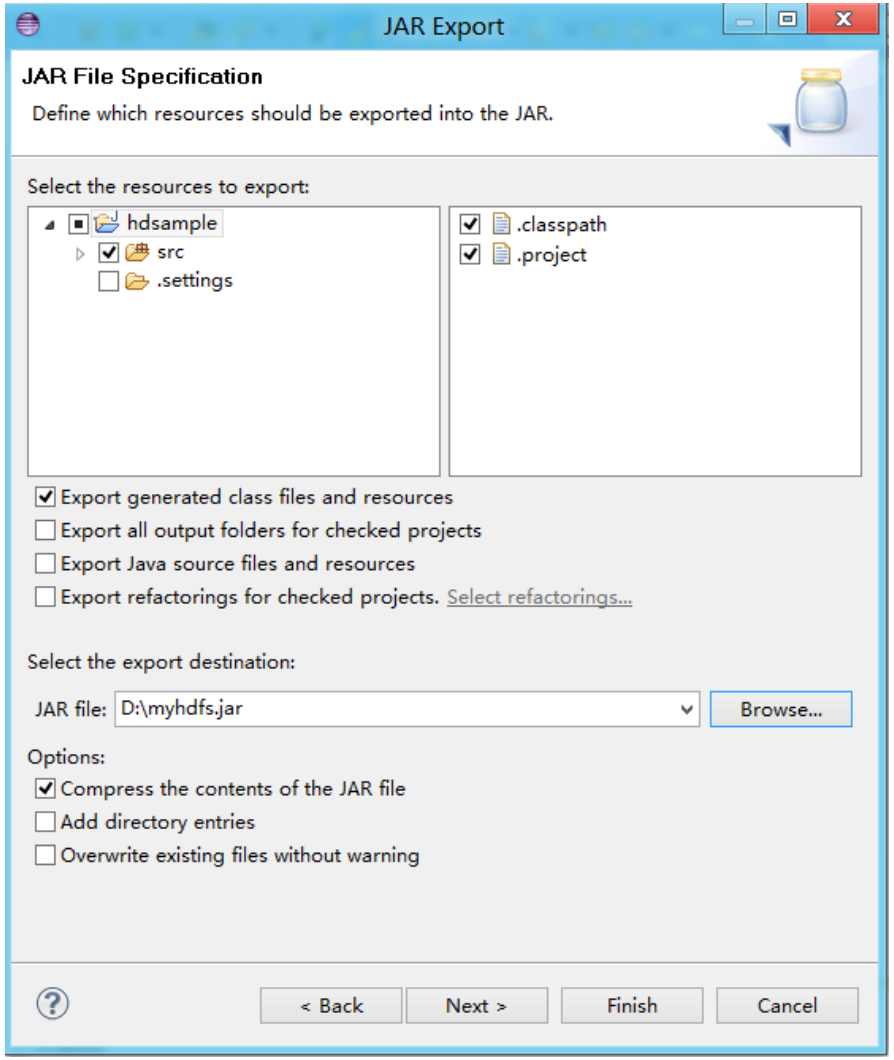


新建项目，编写代码


Java - hdsample/src/org/meepo/hd/HDFSHelloWorld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
hdsample
src
org.meepo.hd
HDFSHelloWorld.jav
JRE System Library [JavaSE
resources.jar - C:\Prog
rt.jar - C:\Program File
jsse.jar - C:\Program Fi
jce.jar - C:\Program Fil
charsets.jar - C:\Progra
jfr.jar - C:\Program File
dnsns.jar - C:\Program
localedata.jar - C:\Prog
sunec.jar - C:\Program
sunjce_provider.jar - C:
sunmscapi.jar - C:\Prog
zipfs.jar - C:\Program
Referenced Libraries
asm-3.2.jar - D:\source
aspectjrt-1.6.5.jar - D:\
aspectjtools-1.6.5.jar -
commons-beanutils-1.
commons-beanutils-co
commons-cli-1.2.jar - D
commons-codec-1.4.ja
commons-collections-3
commons-configuratio
commons-daemon-1.0
commons-digester-1.8
commons-el-1.0.jar - D
commons-httpclient-3.
commons-io-2.1.jar - D
HDFSHelloWorld.java
public class HDFSHelloWorld {
public static final String theFilename = "hello.txt";
public static final String message = "Hello World!\n";
public static void main(String[] args) {
Configuration conf = new Configuration();
FileSystem fs = null;
try {
fs = FileSystem.get(conf);
} catch (IOException e) {
e.printStackTrace();
}
Path filenamePath = new Path(theFilename);
try{
if (fs.exists(filenamePath)){
fs.delete(filenamePath,false);
}
FSDataOutputStream out = fs.create(filenamePath)
out.writeUTF(message);
out.close();
FSDataInputStream in = fs.open(filenamePath);
String messageIn=in.readUTF();
System.out.println(messageIn);
in.close();
}catch(IOException ioe){
System.err.println("IOException during operation
System.exit(1);
}
Outline
org.meepo.hd
HDFSHelloWorld
theFilename : S
message : Strir
main(String[]) :
Problems
Javadoc
Declaration
0 items
Writable
Smart Insert
40 : 6

导出jar文件


Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
Install
Java
JAR file
Javadoc
Runnable JAR file
Plug-in Development
Run/Debug
Team
< Back
Next >
Finish
Cancel


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
hdsample
src
.settings
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: D:\myhdfs.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Finish
Cancel

拷贝程序到Linux主机，并运行程序

pscp (or scp) myhd.jar
hadoop@master:/home/hadoop

程序代码概览

```java
public class CheckFile {
    public static void main(String[] args) throws Exception {

        Configuration conf=new Configuration();
        FileSystem hdfs=FileSystem.get(conf);
        Path findf=new Path("/user");
        boolean isExists=hdfs.exists(findf);
        System.out.println("Exist?"+isExists);
    }
}
```

```java
public class CreateDir {

    public static void main(String[] args) throws Exception{
        Configuration conf=new Configuration();
        FileSystem hdfs=FileSystem.get(conf);
        Path dfs=new Path("/user/inputnew");
        boolean bool2=hdfs.mkdirs(dfs);
        if (bool2)
        {
            System.out.println("ok");
            FileStatus files[]=hdfs.listStatus(new Path("/user"));
            for(FileStatus file:files){
                System.out.println(file.getPath());
            }
        }
        else
        {
            System.out.println("no");
        }
    }
}
```

程序代码概览

```
public class CreateFile {

    public static void main(String[] args) throws Exception {

        Configuration conf=new Configuration();
        FileSystem hdfs=FileSystem.get(conf);
        byte[] buff="hello hadoop world!\r\n hadoop ".getBytes(
        Path dfs=new Path("/user/file.txt");
        FSDataOutputStream outputStream=hdfs.create(dfs);
        outputStream.write(buff,0,buff.length);
    }
}
```

```
public class DeleteFile {

    public static void main(String[] args) throws Exception {

        Configuration conf=new Configuration();
        FileSystem hdfs=FileSystem.get(conf);
        Path delef=new Path("/user/inputnew");
        boolean isDeleted=hdfs.delete(delef,true);
        System.out.println("Delete?"+isDeleted);
    }
}
```

HA——High Availability

为什么需要HA？

- 单点故障
- 集群容量和集群性能

Hadoop1.0尝试的方式

- Secondary NameNode。它不是HA，它只是阶段性的合并edits和fsimage，以缩短集群启动的时间。当NameNode(以下简称NN)失效的时候，Secondary NN并无法立刻提供服务，Secondary NN甚至无法保证数据完整性：如果NN数据丢失的话，在上一次合并后的文件系统的改动会丢失

Hadoop1.0尝试的方式

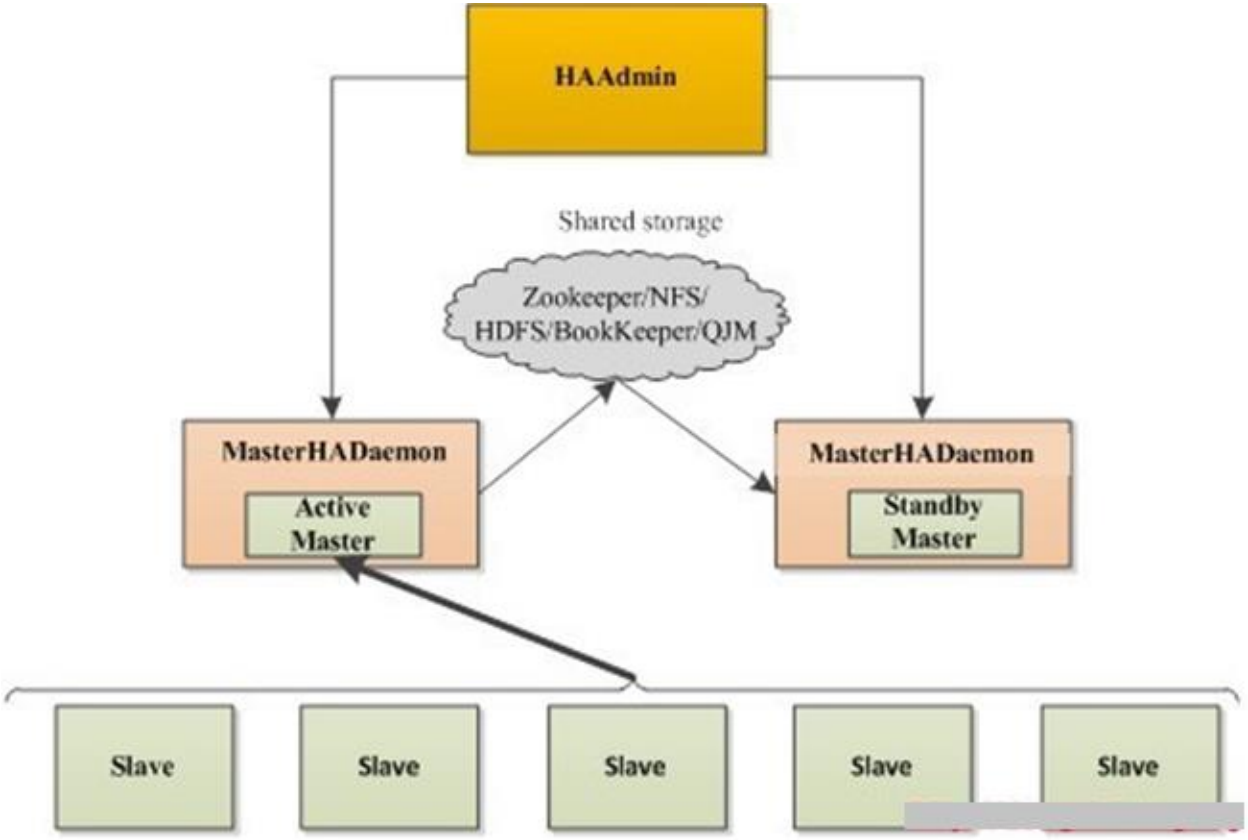
- Backup NameNode (HADOOP-4539)。它在内存中复制了NN的当前状态，算是Warm Standby，可也就仅限于此，并没有failover等。它同样是阶段性的做checkpoint，也无法保证数据完整性
- 手动把name.dir指向NFS。这是安全的Cold Standby，可以保证元数据不丢失，但集群的恢复则完全靠手动。

Hadoop1.0尝试的方式

- Facebook AvatarNode。Facebook有强大的运维做后盾，所以Avatarnode只是Hot Standby，并没有自动切换，当主NN失效的时候，需要管理员确认，然后手动把对外提供服务的虚拟IP映射到Standby NN，这样做的好处是确保不会发生脑裂的场景。
- 还有若干解决方案，基本都是依赖外部的HA机制，譬如DRBD，Linux HA，VMware的FT等等。

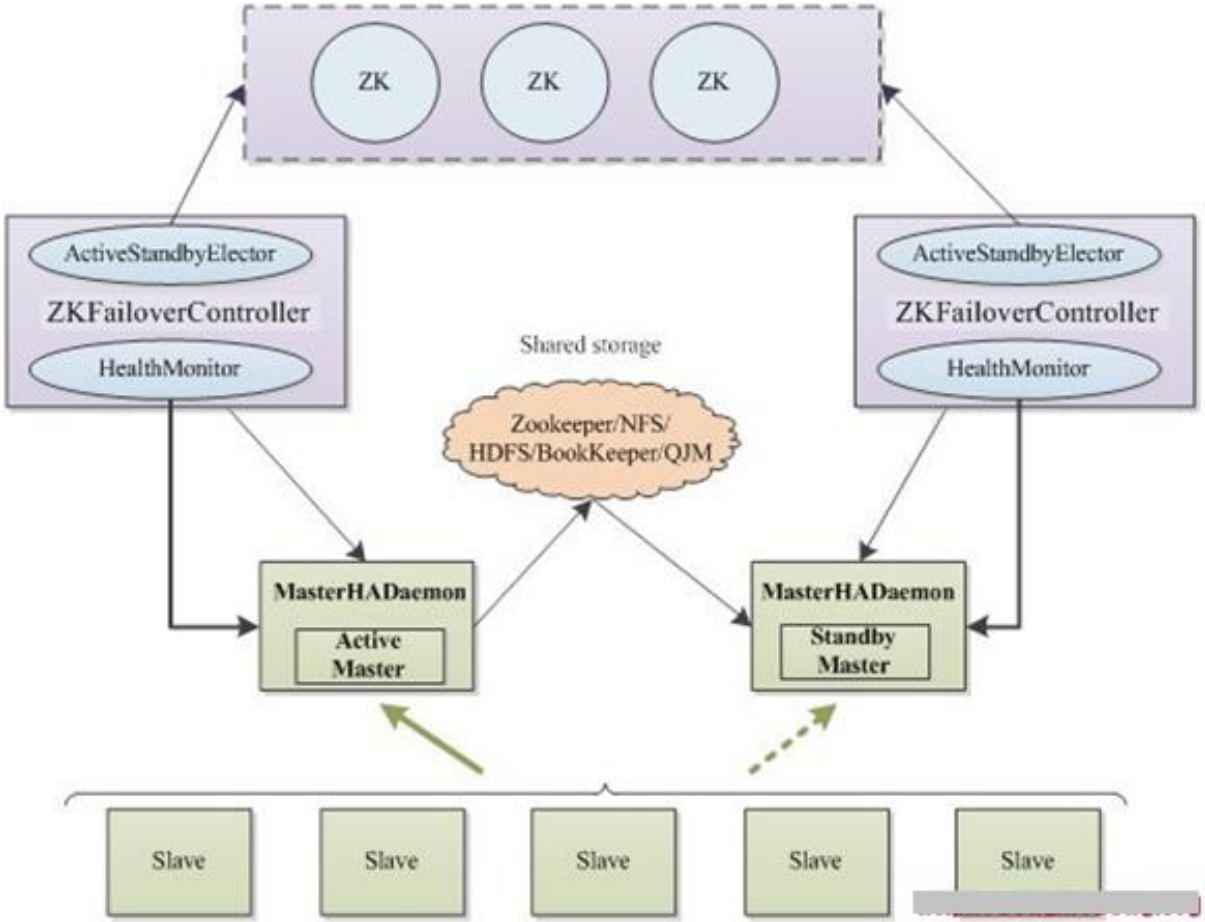
Hadoop HA解决方案架构

- 手动模式



Hadoop HA解决方案架构

- 自动模式



Hadoop HA 主要组件

- **MasterHADaemon**：与Master服务运行在同一个进程中，可接收外部RPC命令，以控制Master服务的启动和停
- **SharedStorage**：共享存储系统，active master将信息写入共享存储系统，而standby master则读取该信息以保持与active master的同步，从而减少切换时间。常用的共享存储系统有zookeeper（被YARN HA采用）、NFS（被HDFS HA采用）、HDFS（被MapReduce HA采用）和类bookeeper系统（被HDFS HA采用）

Hadoop HA 主要组件

- **ZKFailoverController**：基于Zookeeper实现的切换控制器，主要由两个核心组件构成：ActiveStandbyElector和HealthMonitor，其中，ActiveStandbyElector负责与zookeeper集群交互，通过尝试获取全局锁，以判断所管理的master进入active还是standby状态；HealthMonitor负责监控各个活动master的状态，以根据它们状态进行状态切换。

Hadoop HA 主要组件

- **Zookeeper集群**：核心功能通过维护一把全局锁控制整个集群有且仅有一个active master。当然，如果ShardStorge采用了zookeeper，则还会记录一些其他状态和运行时信息。

解决HA考虑两个问题

- 脑裂（brain-split）

  脑裂是指在主备切换时，由于切换不彻底或其他原因，导致客户端和Slave误以为出现两个active master，最终使得整个集群处于混乱状态。解决脑裂问题，通常采用隔离（Fencing）机制，包括三个方面：

  - ✓ 共享存储fencing：确保只有一个Master往共享存储中写数据
  - ✓ 客户端fencing：确保只有一个Master可以响应客户端的请求
  - ✓ Slave fencing：确保只有一个Master可以向Slave下发命令

解决HA考虑两个问题

- 切换对外透明

  为了保证整个切换是对外透明的，Hadoop应保证所有客户端和Slave能自动重定向到新的active master上，这通常是通过若干次尝试连接旧master不成功后，再重新尝试链接新master完成的，整个过程有一定延迟。在新版本的Hadoop RPC中，用户可自行设置RPC客户端尝试机制、尝试次数和尝试超时时间等参数。

HA基本原理——以NN为例

- HA 机制有两个namenode，一个是active namenode，状态是active；另外一个是standby namenode，状态是standby。两者的状态是可以切换的，但不能同时两个都是active状态，最多只有1个是active状态。只有active namenode提供对外的服务，standby namenode是不对外服务的。active namenode和standby namenode之间通过NFS或者JN（journalnode，QJM方式）来同步数据。

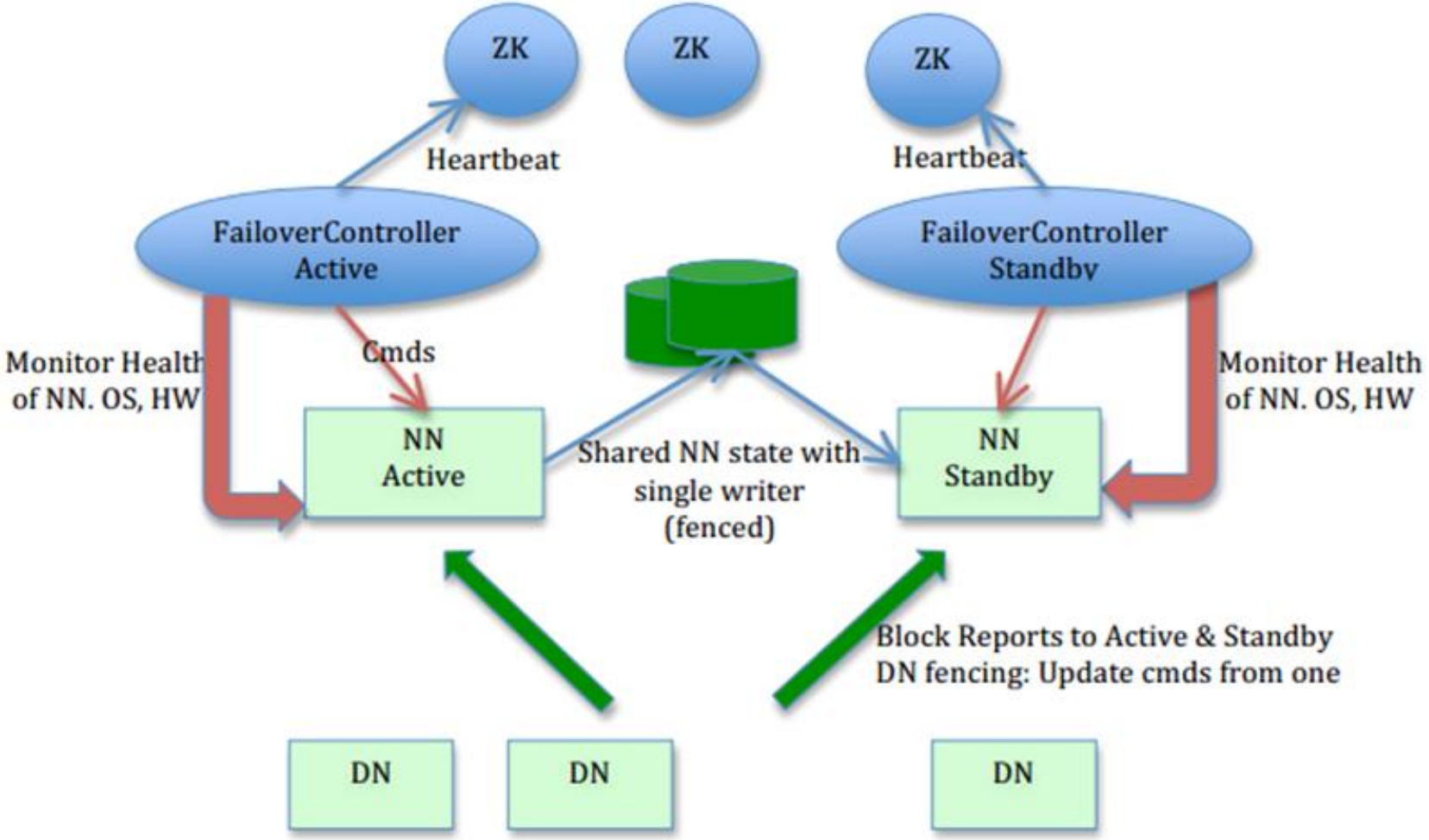
HA基本原理——以NN为例

- active namenode会把最近的操作记录写到本地的一个edits文件中（edits file），并传输到NFS或者JN中。standby namenode定期的检查，从NFS或者JN把最近的edit文件读过来，然后把edits文件和fsimage文件合并成一个新的fsimage，合并完成之后会通知active namenode获取这个新fsimage。active namenode获得这个新的fsimage文件之后，替换原来旧的fsimage文件

HA基本原理——以NN为例

- standby namenode可以随时切换成active namenode（譬如active namenode挂了）。
- hadoop1.0的secondarynamenode，checkpointnode，buckcupnode的功能：合并edits文件和fsimage文件，使fsimage文件一直保持更新。所以启动了hadoop2.0的HA机制之后，secondarynamenode，checkpointnode，buckcupnode这些都不需要了。

Hadoop 2.0 HA实现方式



Hadoop 2.0 HA实现方式

- 利用共享存储来在两个NN间同步edits信息

  通过NN内部每次元数据变动后的flush操作，加上NFS的close-to-open，数据的一致性得到了保证。

- DN同时向两个NN汇报块信息

  让Standby NN保持集群最新状态的必需步骤

Hadoop 2.0 HA实现方式

- 用于监视和控制NN进程的FailoverController进程

  使用ZooKeeper来做同步锁，但用户可以方便的把这个ZooKeeper FailoverController替换为其他的HA方案或leader选举方案。

Question