北风网 IBEIFENG.COM

大数据Hadoop高薪直通车课程

深入 Hadoop 2.x

讲师：轩宇（北风网版权所有)

课程大纲

1. HDFS 架构、启动过程
2. HDFS Java API使用
3. YARN 架构，应用监控
4. MapReduce 编程模型
5. MapReduce Shuffle过程

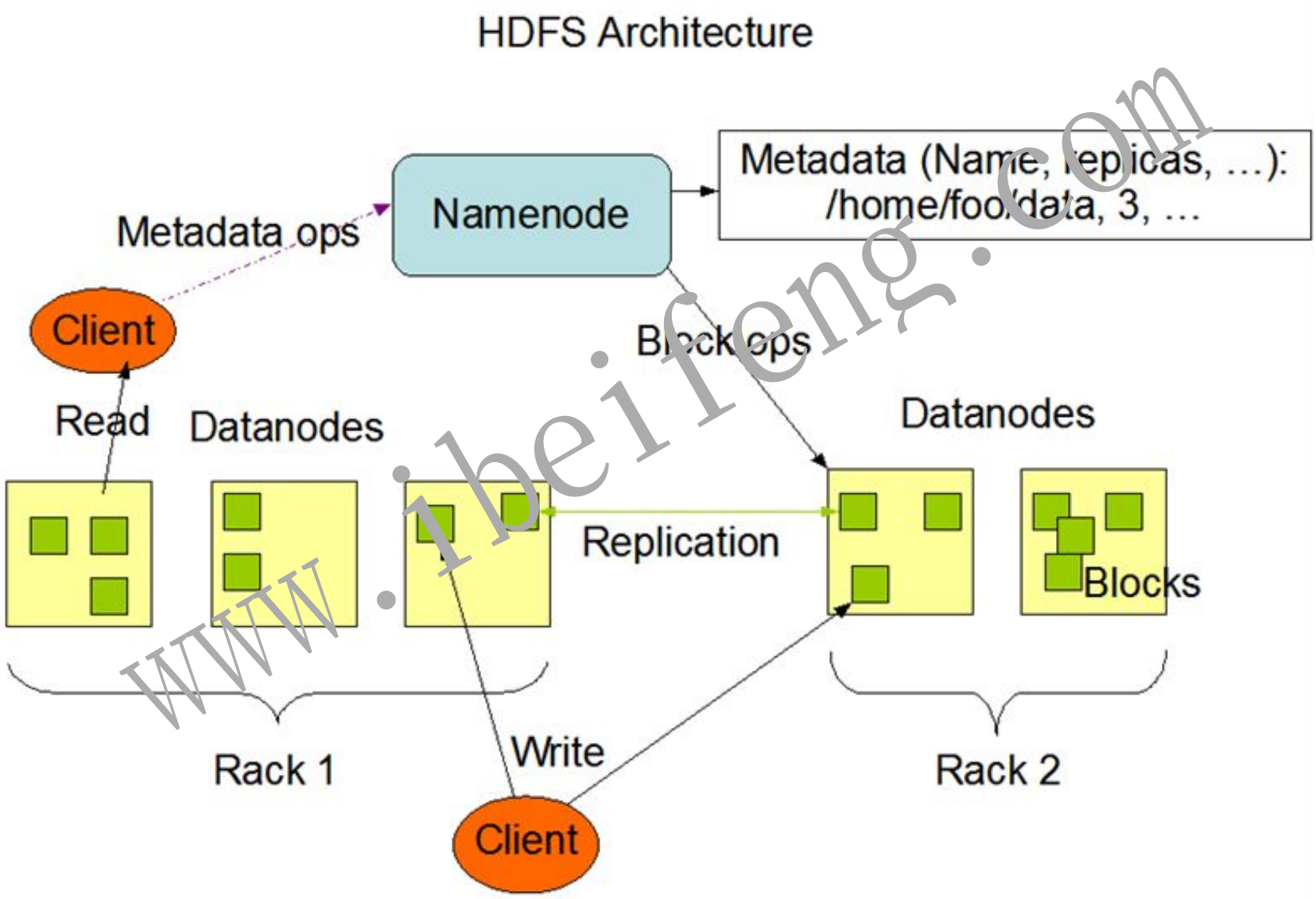
课程大纲

1. HDFS 架构、启动过程
2. HDFS Java API使用
3. YARN 架构，应用监控
4. MapReduce 编程模型
5. MapReduce Shuffle过程

HDFS 来源

- 源自于Google的GFS论文
  - 发表于2003年10月
  - HDFS是GFS克隆版
- Hadoop Distributed File System
  - 易于扩展的分布式文件系统
  - 运行在大量普通廉价机器上，提供容错机制
  - 为大量用户提供性能不错的文件存取服务

HDFS Architecture



NameNode

- Namenode 是一个中心服务器，单一节点（简化系统的设计和实现），**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，跟文件内容相关的数据流不经过NameNode，只会询问它跟那个DataNode联系，否则NameNode会成为系统的瓶颈。
- 副本存放在哪些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 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。

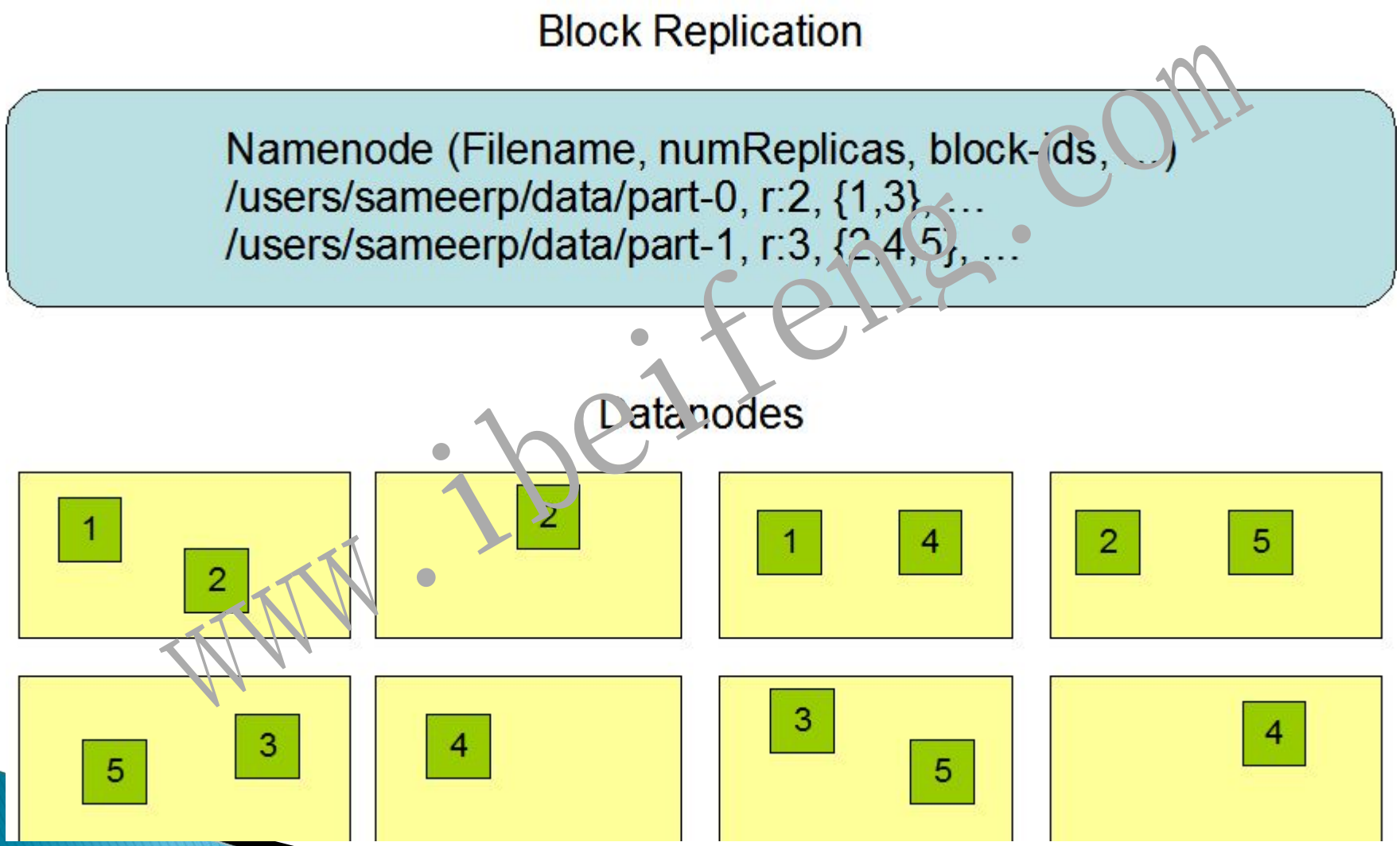
DataNode

- 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。
- 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

文件

- 文件切分成块（默认大小128M），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，以及块数据的校验和。
- 可以创建、删除、移动或重命名文件，当文件创建、写入和关闭之后不能修改文件内容。

HDFS Architecture



数据损坏（corruption）处理

- 当DataNode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。
- NameNode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
- DataNode 在其文件创建后三周验证其checksum。

NameNode 启动过程

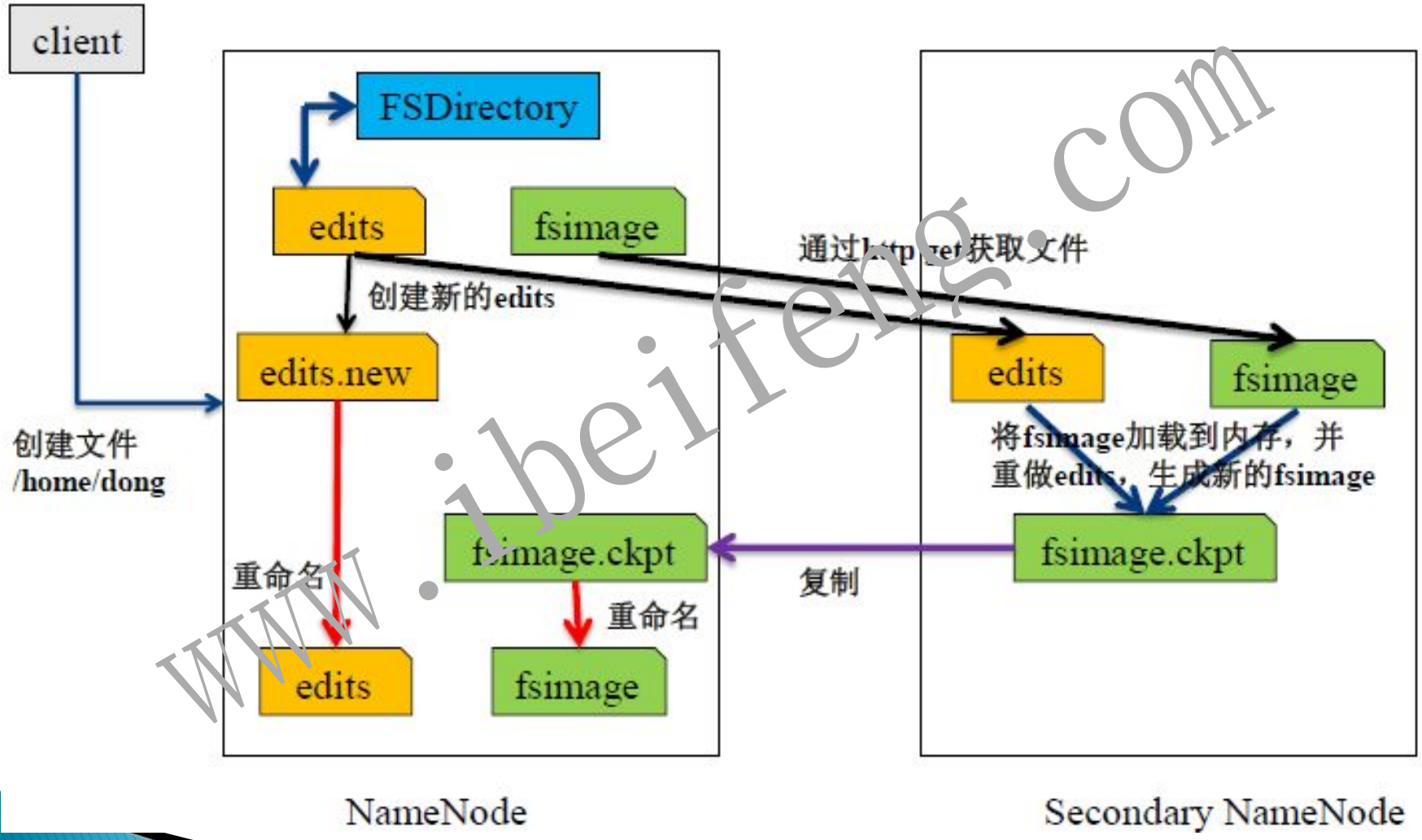
- NameNode 元数据/命名空间持久化 fsimage与edits
- NameNode 格式化，具体做什么事
  - 创建fsimage文件，存储fsimage信息
  - 创建edits文件
- NameNode 启动过程
  - 加载fsimage和edits文件
  - 生成新的fsimage和edits文件
  - 等待DataNode注册与发送Block Report
- DataNode 启动过程
  - 向NameNode注册、发送Block Report
- NameNode SafeMode 安全模式

NameNode 启动过程

- 1、Name启动的时候首先将fsimage（镜像）载入内存，并执行（replay）编辑日志editlog的的各项操作；
- 2、一旦在内存中建立文件系统元数据映射，则创建一个新的fsimage文件（这个过程不需SecondaryNameNode） 和一个空的editlog；
- 3、在安全模式下，各个datanode会向namenode发送块列表的最新情况；
- 4、此刻namenode运行在安全模式。即NameNode的文件系统对于客服端来说是只读的。(显示目录，显示文件内容等。写、删除、重命名都会失败)；
- 5、NameNode开始监听RPC和HTTP请求

  解释RPC:RPC（Remote Procedure Call Protocol）——远程过程通过协议，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议；
- 6、系统中数据块的位置并不是由namenode维护的，而是以块列表形式存储在datanode中；
- 7、在系统的正常操作期间，namenode会在内存中保留所有块信息的映射信息。

NameNode 启动过程



SafeMode 相关说明

- 安全模式下，集群属于只读状态。但是严格来说，只是保证HDFS元数据信息的访问，而不保证文件的访问，因为文件的组成Block信息此时NameNode还不一定已经知道了。所以只有NameNode已了解了Block信息的文件才能独到。而安全模式下任何对HDFS有更新的操作都会失败。
- 对于全新创建的HDFS集群，NameNode启动后不会进入安全模式，因为没有Block信息。

课程大纲

1. HDFS 架构、启动过程
2. HDFS Java API使用
3. YARN 架构，应用监控
4. MapReduce 编程模型
5. MapReduce Shuffle过程

HDFS Shell 命令

- 调用文件系统(FS)Shell命令使用 bin/hdfs dfs 的形式
- 所有的FS Shell命令使用URI路径作为参数
- URI格式是scheme://authority/path。HDFS的scheme是hdfs，对本地文件系统，scheme是file。其中scheme和authority参数都是可选的，如果未加指定，就会使用配置中指定的默认scheme。
- 例如：/parent/child可以表示成hdfs://namenode:namenodePort/parent/child，或者更简单的/parent/child（假设配置文件是namenode:namenodePort）
- 大多数FS Shell命令的行为和对应的Unix Shell命令类似

文件命令

```
[hadoop@hadoop-yarn hadoop-2.2.0]$ bin/hdfs dfs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] <path> ...]
        [-cp [-f] [-p] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-d] [-h] [-R] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
        [-tail [-f] <file>]
        [-test -[defsz] <path>]
        [-text [-ignoreCrc] <src> ...]
        [-touchz <path> ...]
        [-usage [cmd ...]]
```

管理命令

```
[hadoop@hadoop-yarn hadoop-2.2.0]$ bin/hdfs dfsadmin
Usage: java DFSAdmin
Note: Administrative commands can only be run as the HDFS superuser.
          [-report]
          [-safemode enter | leave | get | wait]
          [-allowSnapshot <snapshotDir>]
          [-disallowSnapshot <snapshotDir>]
          [-saveNamespace]
          [-rollEdits]
          [-restoreFailedStorage true|false|check]
          [-refreshNodes]
          [-finalizeUpgrade]
          [-metasave filename]
          [-refreshServiceAcl]
          [-refreshUserToGroupsMappings]
          [-refreshSuperUserGroupsConfiguration]
          [-printTopology]
          [-refreshNamenodes datanodehost:port]
          [-deleteBlockPool datanode-host:port blockpoolId [force]]
          [-setQuota <quota> <dirname>...<dirname>]
          [-clrQuota <dirname>...<dirname>]
          [-setSpaceQuota <quota> <dirname>...<dirname>]
          [-clrSpaceQuota <dirname>...<dirname>]
          [-setBalancerBandwidth <bandwidth in bytes per second>]
          [-fetchImage <local directory>]
          [-help [cmd]]
```

开发环境准备

- 安装Maven，用于管理项目依赖包（apache-maven-3.0.5-bin.tar）
- 安装Eclipse（eclipse-jee-kepler-SR1-linux-gtk-x86_64.tar）
- 配置Eclipse与Maven插件
- 设置Linux下Eclipse快捷键

点击eclipse窗口栏的Window ---》Perferences ---》左边搜索框输入keys ---》点击打开以后在右边的Command里面找到如下两个key对其进行修改。

key	oldValue	modefyValue
Content Assist	Ctrl + Space	Alt + /
Word Completion	Alt + /	Ctrl +Space（只要不与其他快捷键冲突）

- 设置Eclipse字体大小（Java文件和XML文件）
- 创建Maven Project，配置POM文件

使用 HDFS FS API 详解

```
public void fileSystem(String fileUri){
    Configuration conf = new Configuration();
    FileSystem fs = null;
    InputStream in = null;
    try{
        fs = FileSystem.get(URI.create(fileUri),conf);
        in = fs.open(new Path(fileUri));
        //参数:4096 是复制缓冲区的大小  false是复制结束后是否关闭数据流
        IOUtils.copyBytes(in, System.out, 4096,false);
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        IOUtils.closeStream(in);
    }
}
```

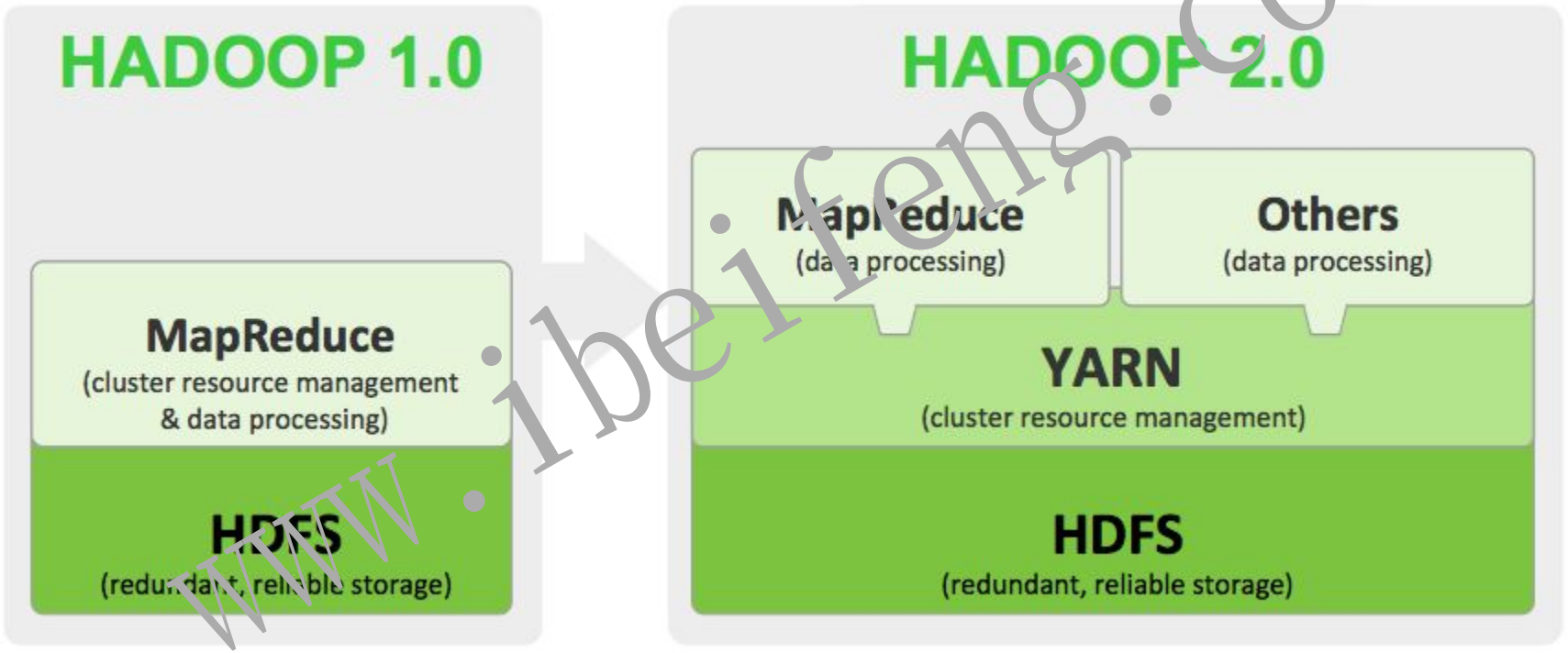
使用 HDFS FS API 详解

- 文件操作
  - 上传本地文件到HDFS
  - 读取文件
  - 在hadoop fs中新建文件，并写入
  - 重命名文件
  - 删除hadoop fs上的文件
- 目录操作
  - 读取某个目录下的所有文件
  - 在hadoop fs上创建目录
  - 删除目录
- HDFS 信息
  - 查找某个文件在HDFS集群的位置
  - 获取HDFS集群上所有节点名称信息

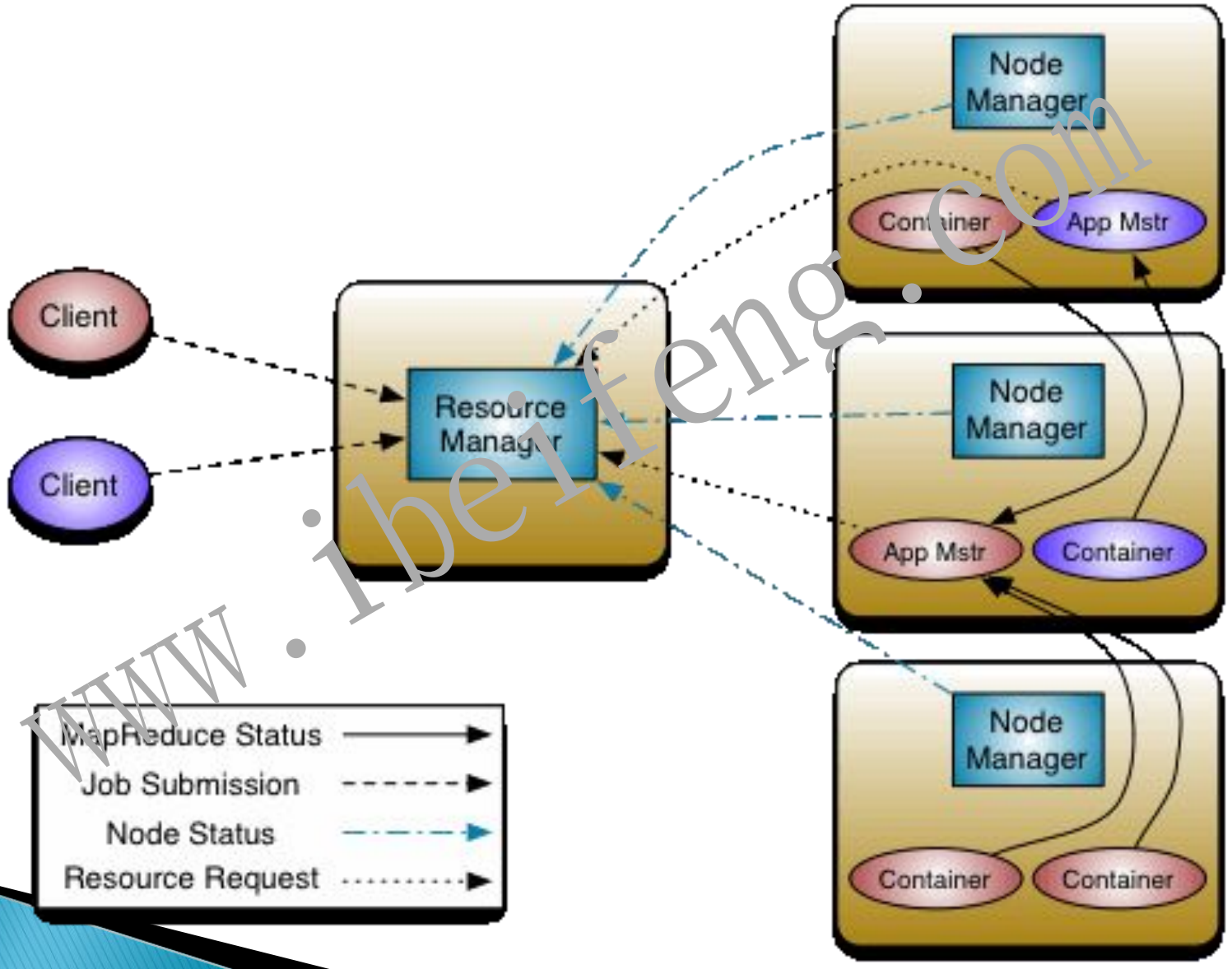
课程大纲

1. HDFS 架构、启动过程
2. HDFS Java API使用
3. YARN 架构，应用监控
4. MapReduce 编程模型
5. MapReduce Shuffle过程

Hadoop 1.x & Hadoop 2.x



YARN 架构图



YARN 架构图

Apache Hadoop NextGen MapReduce (YARN)

MapReduce has undergone a complete overhaul in hadoop-0.23 and we now have, what we call, MapReduce 2.0 (MRv2) or YARN.

The fundamental idea of MRv2 is to split up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. The idea is to have a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*). An application is either a single job in the classical sense of Map-Reduce jobs or a DAG of jobs.

The ResourceManager and per-node slave, the NodeManager (*NM*), form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

YARN服务组件

- YARN 总体上仍然是Master/Slave 结构，在整个资源管理框架中，ResourceManager 为Master，NodeManager 为Slave。
- ResourceManager 负责对各个NodeManager 上的资源进行统一管理和调度
- 当用户提交一个应用程序时，需要提供一个用以跟踪和管理这个程序的。
- ApplicationMaster，它负责向ResourceManager 申请资源，并要求NodeManger 启动可以占用一定资源的任务。
- 由于不同的ApplicationMaster 被分布到不同的节点上，因此它们之间不会相互影响

ResourceManager

- 全局的资源管理器，整个集群只有一个，负责集群资源的统一管理和调度分配。
- 功能
  - 处理客户端请求
  - 启动/监控ApplicationMaster
  - 监控NodeManager
  - 资源分配与调度

NodeManager

- 整个集群有多个，负责单节点资源管理和使用
- 功能
  - 单个节点上的资源管理和任务管理
  - 处理来自ResourceManager的命令
  - 处理来自ApplicationMaster的命令
- NodeManager管理抽象容器，这些容器代表着可供一个特定应用程序使用的针对每个节点的资源。
- 定时地向RM汇报本节点上的资源使用情况和各个Container的运行状态

Application Master

- 管理一个在YARN 内运行的应用程序的每个实例
- 功能
  - 数据切分
  - 为应用程序申请资源，并进一步分配给内部任务
  - 任务监控与容错
- 负责协调来自ResourceManager的资源，开通过NodeManager监视容器的执行和资源使用（CPU、内存等的资源分配）。

Container

- YARN中的资源抽象，封装某个节点上多维度资源，如内存、CPU、磁盘、网络等，当AM向RM申请资源时，RM向AM返回的资源便是用Container表示的。
- YARN 会为每个任务分配一个Container，且该任务只能使用该Container中描述的资源。
- 功能
  - 对任务运行环境的抽象
  - 描述一系列信息
  - 任务运行资源（节点、内存、CPU）
  - 任务启动命令
  - 任务运行环境

YARN 资源管理

- 资源调度和资源隔离是YARN作为一个资源管理系统，最重要和最基础的两个功能。资源调度由ResourceManager完成，而资源隔离由各个NM实现。
- ResourceManager将某个NodeManager上资源分配给任务（这就是所谓的“资源调度”）后，NodeManager需按照要求为任务提供相应的资源，甚至保证这些资源应具有独占性，为任务运行提供基础的保证，这就是所谓的资源隔离。
- 当谈及到资源时，我们通常指内存，CPU和IO三种资源。Hadoop YARN同时支持内存和CPU两种资源的调度。
- 内存资源的多少会会决定任务的生死，如果内存不够，任务可能会运行失败；相比之下，CPU资源则不同，它只会决定任务运行的快慢，不会对生死产生影响。

YARN 资源管理

- YARN允许用户配置每个节点上可用的物理内存资源，注意，这里是“可用的”，因为一个节点上的内存会被若干个服务共享，比如一部分给YARN，一部分给HDFS，一部分给HBase等，YARN配置的只是自己可以使用的，配置参数如下：

（1）yarn.nodemanager.resource.memory-mb

表示该节点上YARN可使用的物理内存总量，默认是8192（MB），注意，如果你的节点内存资源不够8GB，则需要调减小这个值，而YARN不会智能的探测节点的物理内存总量。

（2）yarn.nodemanager.vmem-pmem-ratio

任务每使用1MB物理内存，最多可使用虚拟内存量，默认是2.1。

（3）yarn.nodemanager.pmem-check-enabled

是否启动一个线程检查每个任务正使用的物理内存量，如果任务超出分配值，则直接将其杀掉，默认是true。

（4）yarn.nodemanager.vmem-check-enabled

是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是true。

（5）yarn.scheduler.minimum-allocation-mb

单个任务可申请的最少物理内存量，默认是1024（MB），如果一个任务申请的物理内存量少于该值，则该对应的值改为这个数。

（6）yarn.scheduler.maximum-allocation-mb

单个任务可申请的最多物理内存量，默认是8192（MB）。

YARN 资源管理

- 目前的CPU被划分成虚拟CPU（CPU virtual Core），这里的虚拟CPU是YARN自己引入的概念，初衷是，考虑到不同节点的CPU性能可能不同，每个CPU具有的计算能力也是不一样的，比如某个物理CPU的计算能力可能是另外一个物理CPU的2倍，这时候，你可以通过为第一个物理CPU多配置几个虚拟CPU弥补这种差异。用户提交作业时，可以指定每个任务需要的虚拟CPU个数。在YARN中，CPU相关配置参数如下：

（1）yarn.nodemanager.resource.cpu-vcores

表示该节点上YARN可使用的虚拟CPU个数，默认是8，注意，目前推荐将该值设值为与物理CPU核数数目相同。如果你的节点CPU核数不够8个，则需要调减小这个值，而YARN不会智能的探测节点的物理CPU总数。

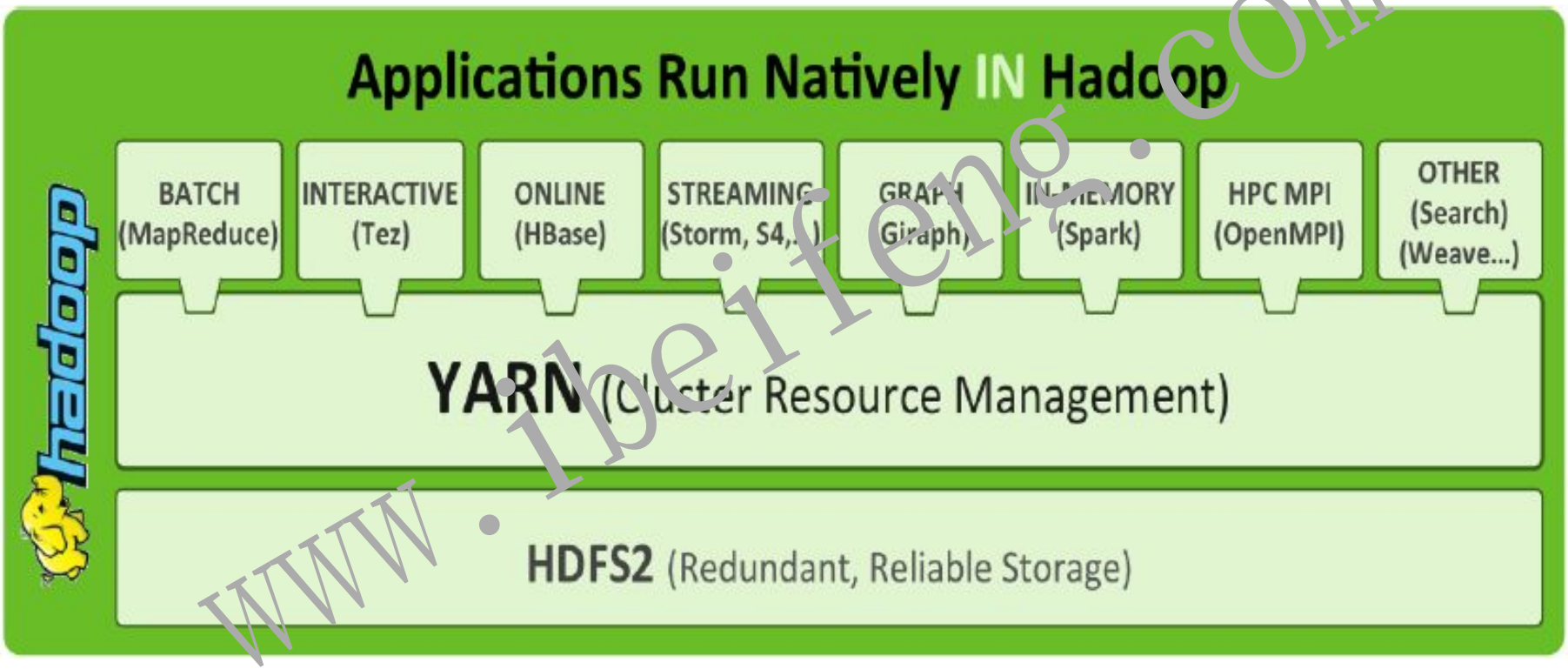
（2） yarn.scheduler.minimum-allocation-vcores

单个任务可申请的最小虚拟CPU个数，默认是1，如果一个任务申请的CPU个数少于该数，则该对应的值改为这个数。

（3）yarn.scheduler.maximum-allocation-vcores

单个任务可申请的最多虚拟CPU个数，默认是32。

以YARN为核心的生态系统



Apache Slider

Apache Slider: Dynamic YARN Applications

Apache Slider is a YARN application to deploy existing distributed applications on YARN, monitor them, and make them larger or smaller as desired -even while the application is running.

Applications can be stopped then started; the distribution of the deployed application across the YARN cluster is persisted —enabling a best-effort placement close to the previous locations. Applications which remember the previous placement of data (such as HBase) can exhibit fast start-up times from this feature.

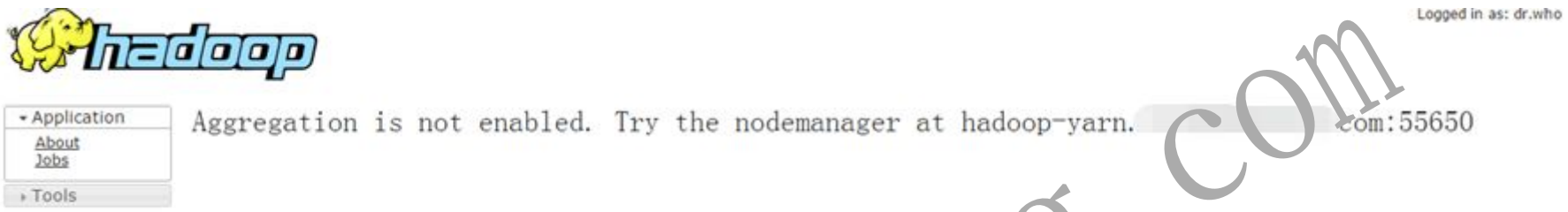
YARN itself monitors the health of "YARN containers" hosting parts of the deployed application -it notifies the Slider manager application of container failure. Slider then asks YARN for a new container, into which Slider deploys a replacement for the failed component. As a result, Slider can keep the size of managed applications consistent with the specified configuration, even in the face of failures of servers in the cluster -as well as parts of the application itself

Some of the features are:

- Allows users to create on-demand applications in a YARN cluster
- Allow different users/applications to run different versions of the application.
- Allow users to configure different application instances differently
- Stop / Suspend / Resume application instances as needed
- Expand / shrink application instances as needed

The Slider tool is a Java command line application.

日志聚合



启用日志聚合 yarn.log-aggregation-enable	☑ 默认值	是否启用日志聚合
日志聚合保留期 yarn.log-aggregation.retain-seconds	7 天 默认值	删除聚合日志前要保留它们多久。

日志聚合

```
<property>
    <name>yarn.log-aggregation-enable</name>
    <value>false</value>
</property>

<property>
    <name>yarn.log-aggregation.retain-seconds</name>
    <value>-1</value>
</property>

<property>
    <name>yarn.log-aggregation.retain-check-interval-seconds</name>
    <value>-1</value>
</property>

<property>
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>/tmp/logs</value>
</property>

<property>
    <name>yarn.nodemanager.remote-app-log-dir-suffix</name>
    <value>logs</value>
</property>
```

日志聚合

- Application
  - About
  - Jobs
- Tools

```
Log Type: stderr
Log Length: 222
log4j:WARN No appenders could be found for logger (org.apache.hadoop.ipc.Server).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Log Type: stdout
Log Length: 0

Log Type: syslog
Log Length: 33298
Showing 4096 bytes of 33298 total. Click here for the full log.
b_1416808157495_0001_1.jhist to hdfs://hadoop-ya                    :8020/tmp/hadoop-yarn/staging/history/done_intermediate/cyhp/job_1416808157495_0001-1416808[illegible]
2014-11-24 00:51:19,233 INFO [eventHandlingThread] org.apache.hadoop.mapreduce.jobhistory.JobHistoryEventHandler: Copied to done location: hdfs://hadoop-yar[illegible]
2014-11-24 00:51:19,236 INFO [eventHandlingThread] org.apache.hadoop.mapreduce.jobhistory.JobHistoryEventHandler: Copying hdfs://hadoop-yarn.cloudyhado[illegible]
2014-11-24 00:51:19,262 INFO [eventHandlingThread] org.apache.hadoop.mapreduce.jobhistory.JobHistoryEventHandler: Copied to done location: hdfs://hado[illegible]
2014-11-24 00:51:19,266 INFO [eventHandlingThread] org.apache.hadoop.mapreduce.jobhistory.JobHistoryEventHandler: Moved tmp to done: hdfs://hadoop-y[illegible]
2014-11-24 00:51:19,268 INFO [eventHandlingThread] org.apache.hadoop.mapreduce.jobhistory.JobHistoryEventHandler: Moved tmp to done: hdfs://hadoop-yar[illegible]
2014-11-24 00:51:19,270 INFO [eventHandlingThread] org.apache.hadoop.mapreduce.jobhistory.JobHistoryEventHandler: Moved tmp to done: hdfs://hadoop-yarn.[illegible]
2014-11-24 00:51:19,271 INFO [Thread-72] org.apache.hadoop.mapreduce.jobhistory.JobHistoryEventHandler: Stopped JobHistoryEventHandler. super.stop()
2014-11-24 00:51:19,271 ERROR [uber-EventHandler] org.apache.hadoop.mapred.LocalContainerLauncher: Returning, interrupted : java.lang.InterruptedException
2014-11-24 00:51:19,272 INFO [Thread-72] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Setting job diagnostics to
2014-11-24 00:51:19,272 INFO [Thread-72] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: History url is http://hadoop-yarn[illegible]/jobhistory/job/
2014-11-24 00:51:19,279 INFO [Thread-72] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Waiting for application to be [illegible]
2014-11-24 00:51:20,295 INFO [Thread-72] org.apache.hadoop.mapreduce.v2.app.MRAppMaster: Deleting staging directory hdfs://hadoop-y[illegible]/tmp/hadoop-yarn
2014-11-24 00:51:20,305 INFO [Thread-72] org.apache.hadoop.ipc.Server: Stopping server on 40317
2014-11-24 00:51:20,312 INFO [IPC Server listener on 40317] org.apache.hadoop.ipc.Server: Stopping IPC Server listener on 40317
2014-11-24 00:51:20,314 INFO [IPC Server Responder] org.apache.hadoop.ipc.Server: Stopping IPC Server Responder
2014-11-24 00:51:20,315 INFO [TaskHeartbeatHandler PingChecker] org.apache.hadoop.mapreduce.v2.app.TaskHeartbeatHandler: TaskHeartbeatHandler thread interrupted
```

课程大纲

1. HDFS 架构、启动过程
2. HDFS Java API使用
3. YARN 架构，应用监控
4. MapReduce 编程模型
5. MapReduce Shuffle过程

MapReduce 编程模型

- 一种分布式计算模型，解决海量数据的计算问题
- MapReduce将整个并行计算过程抽象到两个函数
  - Map(映射)：对一些独立元素组成的列表的每一个元素进行指定的操作，可以高度并行。
  - Reduce(化简)：对一个列表的元素进行合并。
- 一个简单的MapReduce程序只需要指定map()、reduce()、input和output，剩下的事由框架完成。

MapReduce 编程模型

- MapReduce将作业的整个运行过程分为两个阶段：Map阶段和Reduce阶段
- Map阶段由一定数量的Map Task组成
  - ✓ 输入数据格式解析：InputFormat
  - ✓ 输入数据处理：Mapper
  - ✓ 数据分组：Partitioner
- Reduce阶段由一定数量的Reduce Task组成
  - ✓ 数据远程拷贝
  - ✓ 数据按照key排序
  - ✓ 数据处理：Reducer
  - ✓ 数据输出格式：OutputFormat

编写 MapReduce 程序

- 基于MapReduce 计算模型编写分布式并行程序非常简单，程序员的主要编码工作就是**实现Map 和Reduce函数**。
- 其它的**并行编程中的种种复杂问题**，如分布式存储，工作调度，负载平衡，容错处理，网络通信等，**均由YARN框架负责处理**。

MapReduce 八股文

- MapReduce中，map和reduce函数遵循如下常规格式：

  map: (K1, V1) → list(K2, V2)

  reduce: (K2, list(V2)) → list(K3, V3)

- Mapper的基类：

```
protected void map(KEY key, VALUE value, Context context) throws
        IOException, InterruptedException {
 }
```

- Reducer的基类：

```
protected void reduce(KEY key, Iterable<VALUE> values,
        Context context) throws IOException, InterruptedException {
 }
```

- Context是上下文对象

MapReduce 八股文

```
Class MR{
  static public Class Mapper ...{
    //Map代码块
  }
  static public Class Reducer ...{
    //Reduce代码块
  }
  main(){
    Configuration conf = new Configuration();
    Job job = new Job(conf, "job name");
    job.setJarByClass(thisMainClass.class);
    job.setMapperClass(Mapper.class);
    job.setReduceClass(Reducer.class);
    FileInputFormat.addInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    //其他配置参数代码
    job.waitForCompletion(true);
  }
}
```

Mapper区

Reducer区

Driver区

WordCount程序

```
public class WordCount extends Configured implements Tool
{
    public static class WordCountMapper
            extends Mapper<Object, Text, Text, IntWritable>
    {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            String[] words = value.toString().split(" ") ;
            for (String str: words)
            {
                word.set(str);
                context.write(word, one);
            }
        }
    }
```

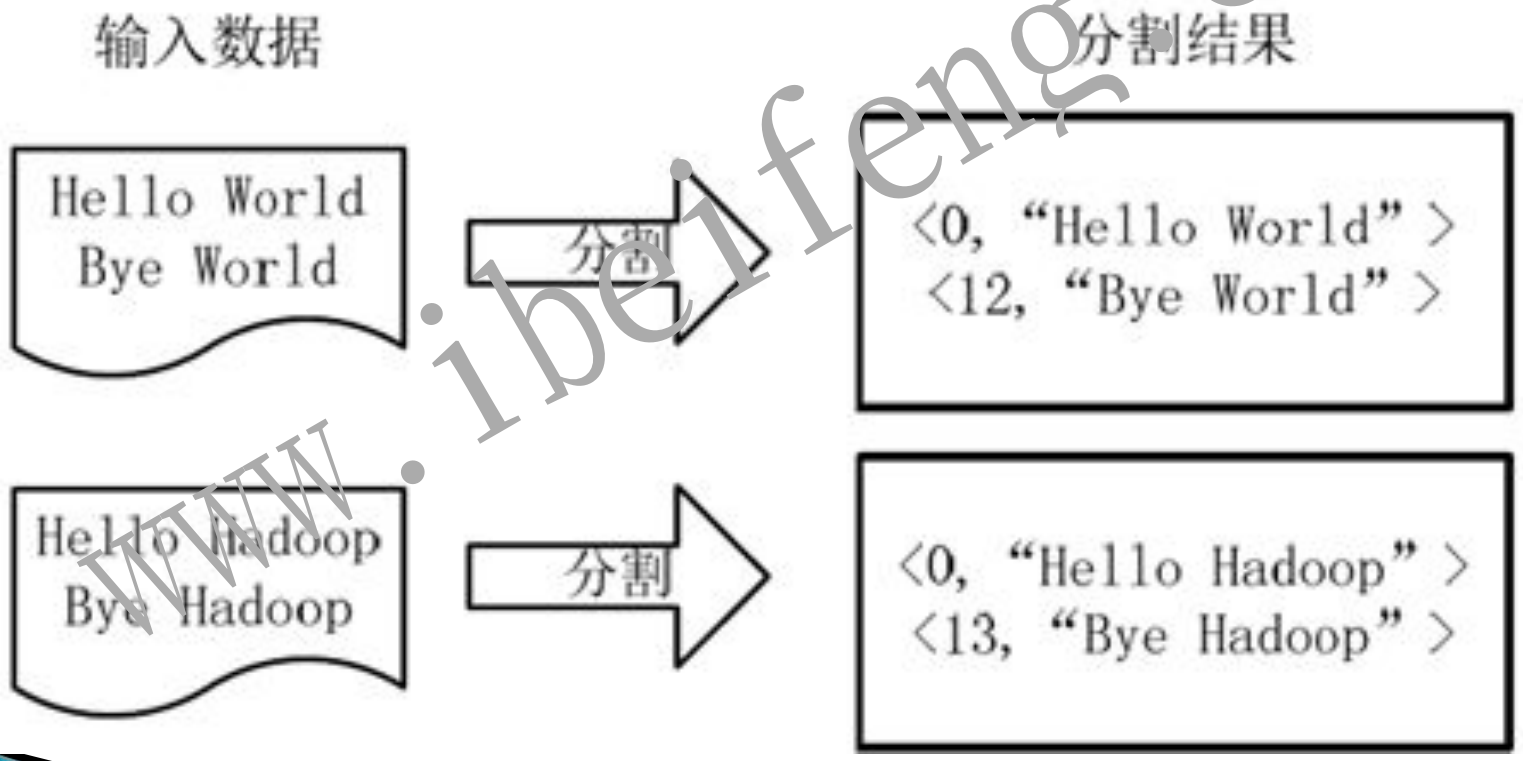
```
public static class WordCountReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
    ) throws IOException, InterruptedException {
        int total = 0;
        for (IntWritable val : values) {
            total++ ;
        }
        context.write(key, new IntWritable(total));
```

WordCount程序

```
    }
}

public int run(String[] args) throws Exception {
    Configuration conf = getConf();

    args = new GenericOptionsParser(conf, args)
    .getRemainingArgs();

    Job job = Job.getInstance(conf);

    job.setJarByClass(WordCount.class);
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    return (job.waitForCompletion(true) ? 0 : 1);
}

public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new WordCount(), args);
    System.exit(exitCode);
}
}
```

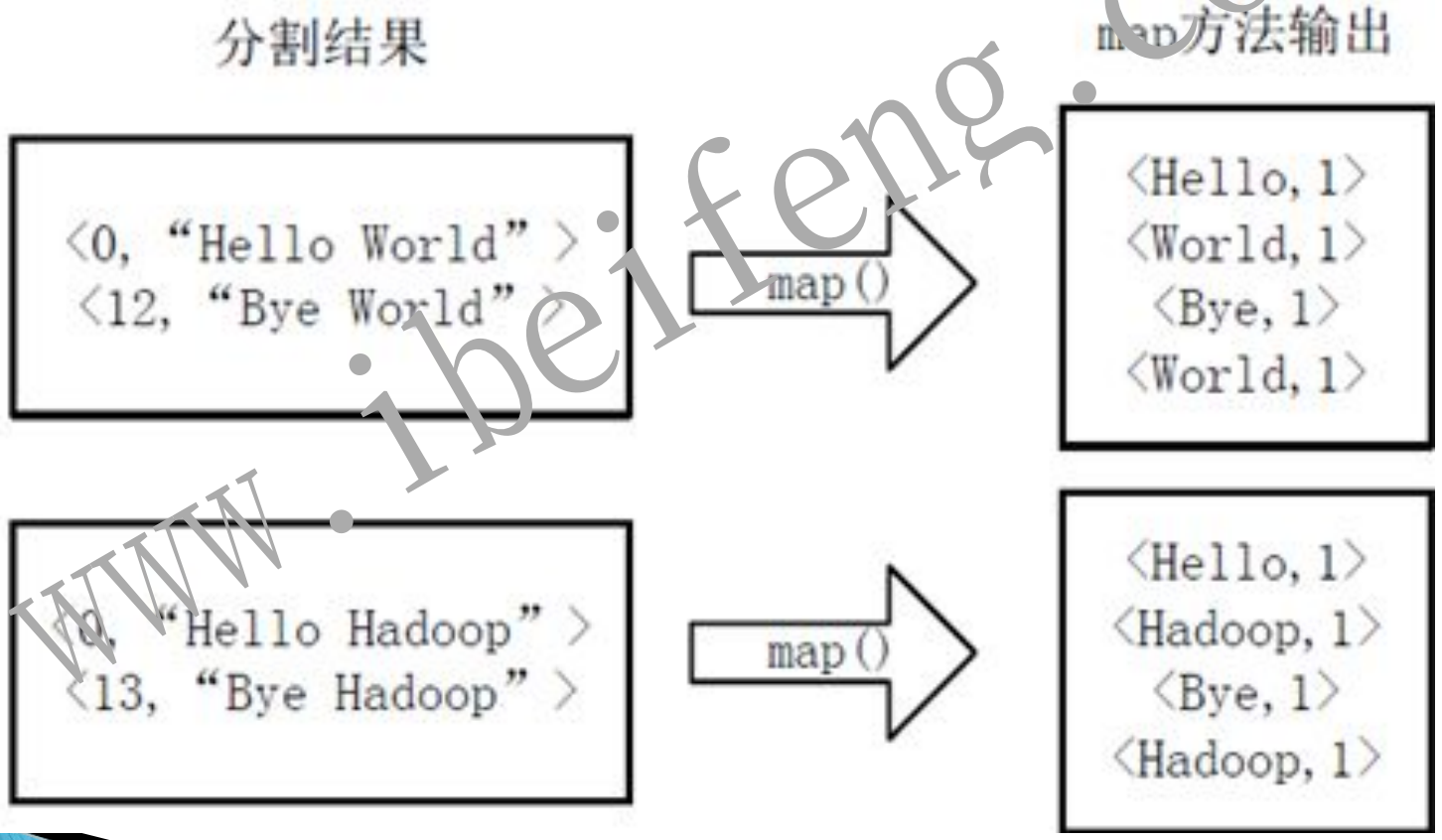
WordCount 处理过程

- 将文件拆分成splits，由于测试用的文件较小，所以每个文件为一个split，并将文件按行分割形成<key,value>对，下图所示。这一步由MapReduce框架自动完成，其中偏移量（即key值）包括了回车所占的字符数（Windows/Linux环境不同）。



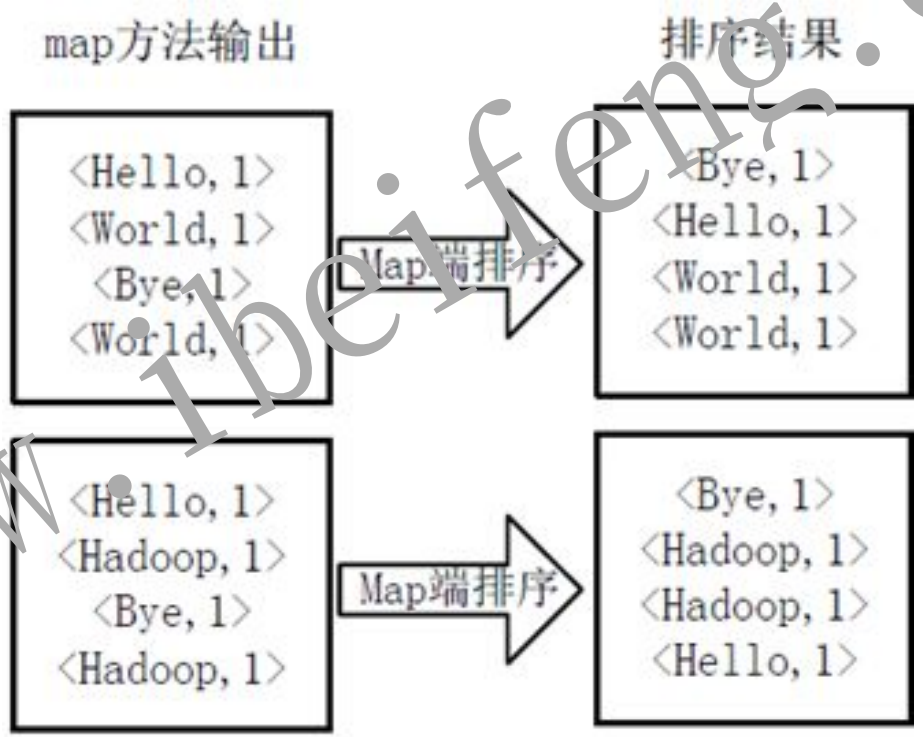
WordCount 处理过程

- 将分割好的<key,value>对交给用户定义的map方法进行处理，生成新的<key,value>对，下图所示。



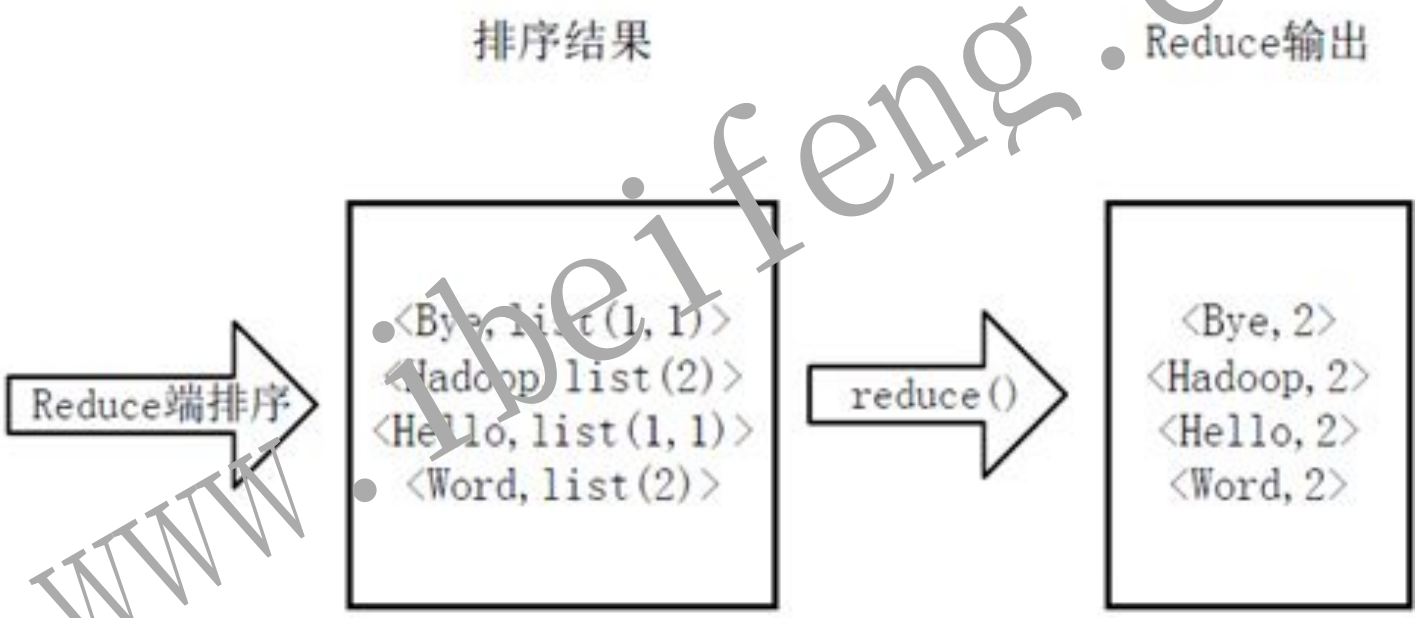
WordCount 处理过程

- 得到map方法输出的<key,value>对后，Mapper会将它们按照key值进行排序，得到Mapper的最终输出结果。下图所示。



WordCount 处理过程

- Reducer先对从Mapper接收的数据进行排序，再交由用户自定义的reduce方法进行处理，得到新的<key,value>对，并作为WordCount的输出结果，下图所示。



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
  - 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
- Java值对象的比较：

  重写 toString()、hashCode()、equals()方法

课程大纲

1. HDFS 架构、启动过程
2. HDFS Java API使用
3. YARN 架构，应用监控
4. MapReduce 编程模型
5. MapReduce Shuffle过程

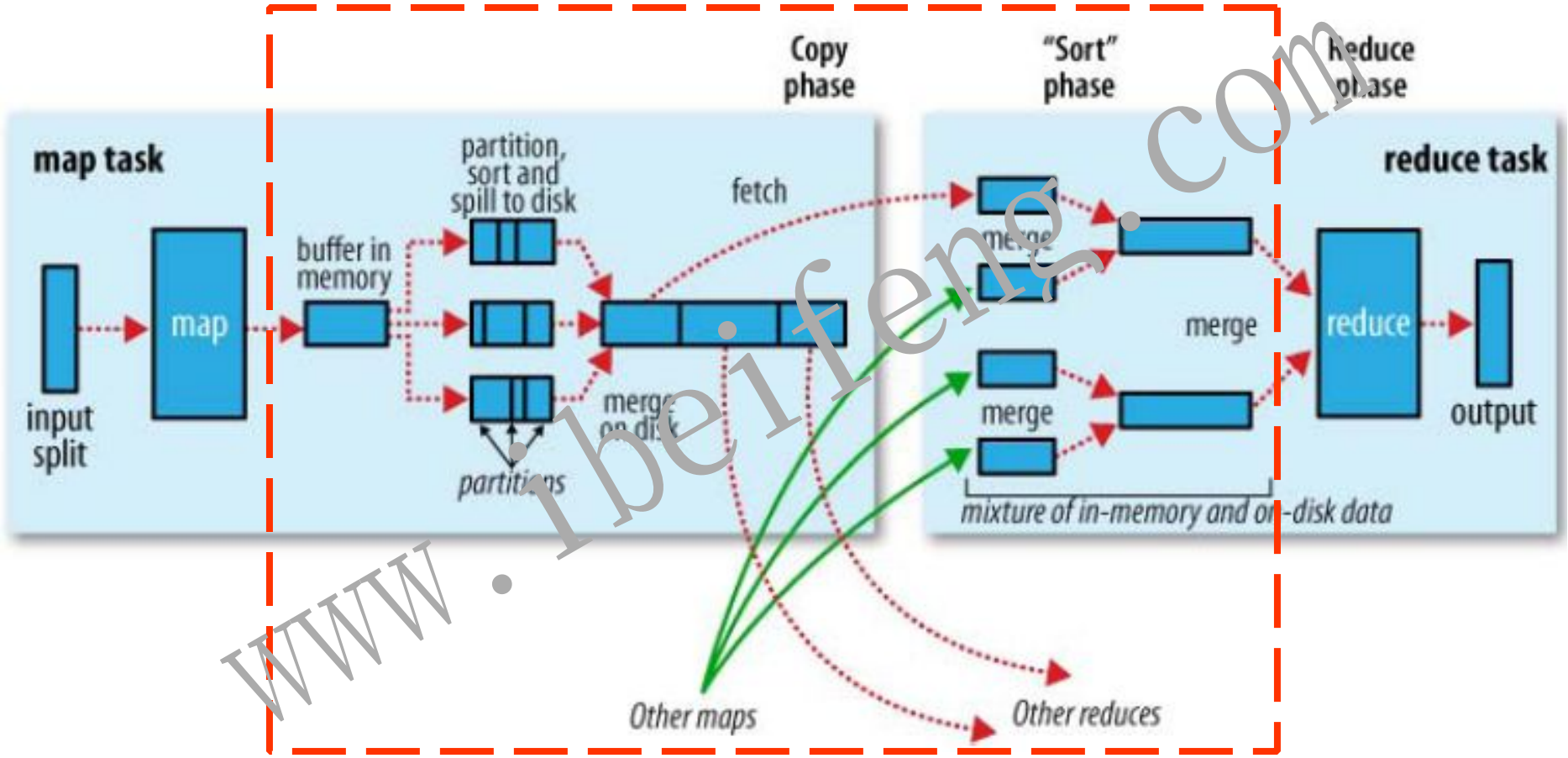
MapReduce 编程模板

- MapReduce 编程模板类结构图

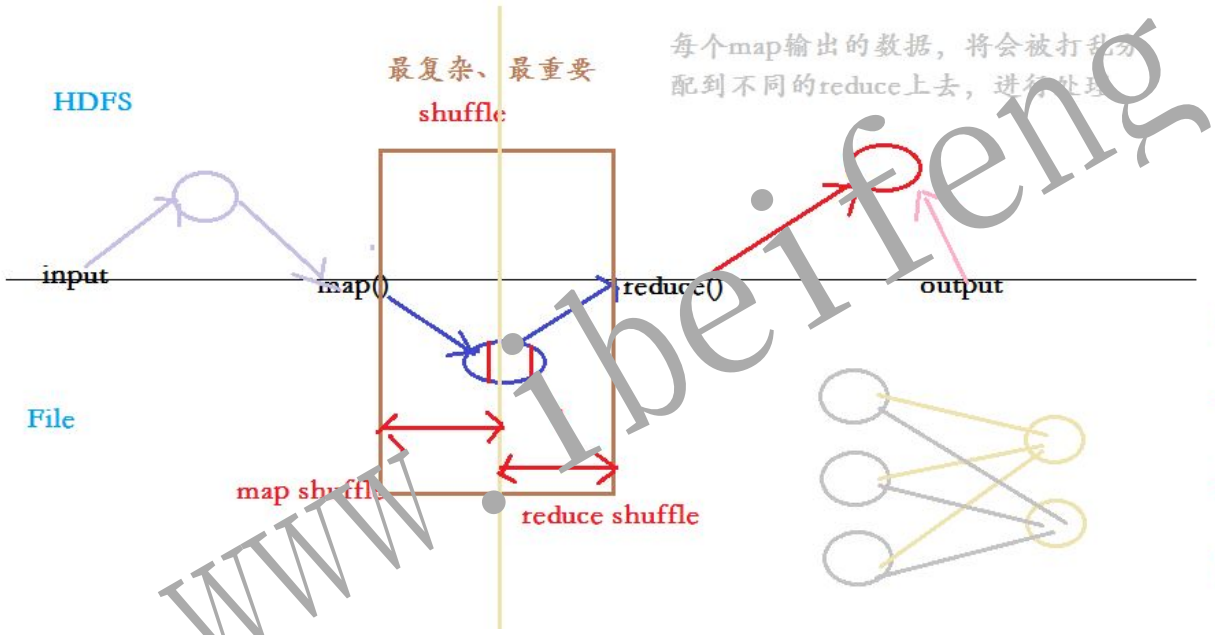
Shuffle 概念

- 意思：洗牌或弄乱。
- Collections.shuffle(List)：随机地打乱参数list里的元素顺序。
- MapReduce里Shuffle：描述着数据从map task输出到reduce task输入的这段过程。

Shuffle 过程



Shuffle 过程



1）分区partitioner
map输出的那些数据分配给那些reduce处理，过程

2）Sort
分区中的数据进行排序

3) Combiner　并不是都有

4）拷贝　（排序）
从每个map输出数据中拷贝属于自己数据，从分区

5）分组Grouping
将相同key的value放在一起，集合

MapReduce 调优

- Reduce Task Number
- Map Task 输出压缩
- Shuffle Phase 参数

```
<property>
  <name>mapreduce.task.io.sort.factor</name>
  <value>10</value>
  <description>The number of streams to merge at once while sorting
  files.  This determines the number of open file handles.</description>
</property>

<property>
  <name>mapreduce.task.io.sort.mb</name>
  <value>100</value>
  <description>The total amount of buffer memory to use while sorting
  files, in megabytes.  By default, gives each merge stream 1MB, which
  should minimize seeks.</description>
</property>

<property>
  <name>mapreduce.map.sort.spill.percent</name>
  <value>0.80</value>
  <description>The soft limit in the serialization buffer. Once reached, a
  thread will begin to spill the contents to disk in the background. Note that
  collection will not block if this threshold is exceeded while a spill is
  already in progress, so spills may be larger than this threshold when it is
  set to less than .5</description>
</property>
```

```
<property>
  <name>mapreduce.map.cpu.vcores</name>
  <value>1</value>
  <description>
      The number of virtual cores r
  </description>
</property>

<property>
  <name>mapreduce.reduce.cpu.vcores</name>
  <value>1</value>
  <description>
      The number of virtual cores required for each reduce task.
  </description>
</property>
```

本课程版权归北风网所有

欢迎访问我们的官方网站

www.ibeifeng.com