**Hadoop及其生态圈组件**

目录

[1. Hadoop的基本概念 1](#_Toc14085899)

[1.1. Hadoop是什么 1](#_Toc14085900)

[1.2. Hadoop两大功能： 1](#_Toc14085901)

[1.3. Hadoop有3大核心组件 1](#_Toc14085902)

[1.4. Hadoop的优势 1](#_Toc14085903)

[2. Hadoop的生态圈组件 2](#_Toc14085904)

[2.1. HDFS 2](#_Toc14085905)

[2.1.1. HDFS的组成 2](#_Toc14085906)

[2.1.2. hdfs构架原则 3](#_Toc14085907)

[2.1.3. Hdfs的页面及命令 4](#_Toc14085908)

[2.2. Zookeeper（分布式协作服务） 5](#_Toc14085909)

[2.2.1. Zookeeper是什么 5](#_Toc14085910)

[2.2.2. Zookeeper的作用 5](#_Toc14085911)

[2.3. Yarn(分布式资源管理器） 6](#_Toc14085912)

[2.3.1. Yarn的概念 6](#_Toc14085913)

[2.3.2. Yarn的构成 6](#_Toc14085914)

[2.4. Oozie(工作流调度器） 8](#_Toc14085915)

[2.5. Sqoop(数据ETL/同步工具） 9](#_Toc14085916)

[2.5.1. Sqoop概念 9](#_Toc14085917)

[2.5.2. 实例 9](#_Toc14085918)

[2.6. HIVE（数据仓库） 10](#_Toc14085919)

[2.6.1. Hive的概念 10](#_Toc14085920)

[2.6.2. Hive的设计特点 10](#_Toc14085921)

[2.6.3. Hive 中的表分类 11](#_Toc14085922)

[2.6.4. 实例 12](#_Toc14085923)

[2.7. HBASE（分布式列存数据库） 12](#_Toc14085924)

[2.7.1. Hbase概念 12](#_Toc14085925)

[2.7.2. Hbase系统架构 13](#_Toc14085926)

[2.7.3. Hbase命令及实例 13](#_Toc14085927)

[2.8. Spark(内存DAG计算模型) 14](#_Toc14085928)

[2.8.1. Spark概念 14](#_Toc14085929)

[2.8.2. Spark的适用场景 15](#_Toc14085930)

[2.8.3. Spark分类 15](#_Toc14085931)

[2.8.4. Spark实例 16](#_Toc14085932)

[2.9. Impala 17](#_Toc14085933)

[2.9.1. Impala概念 17](#_Toc14085934)

[2.9.2. Impala与hive的关系 17](#_Toc14085935)

[2.9.3. Impala与hive的异同 18](#_Toc14085936)

[2.10. hue 20](#_Toc14085937)

[2.10.1. Hue概念 20](#_Toc14085938)

# Hadoop的基本概念

Hadoop是一个由Apache基金会所开发的分布式系统基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力进行高速运算和存储。

## Hadoop是什么

Hadoop是用于存储和处理（运算分析）海量数据的技术平台，是采用分布式集群的方式实现。

## Hadoop两大功能：

1. 提供海量数据的存储服务
2. 提供分析海量数据的编程框架及运行平台

## Hadoop有3大核心组件

1. HDFS：hadoop分布式文件系统，海量数据的存储（集群服务）
2. MapReduce：分布式运算框架，海量数据运算分析（采用java编写MR的程序）
3. Yarn：资源调度管理集群（可以理解为一个分布式的操作系统，管理和分配集群的硬件资源）

## Hadoop的优势

1. **高可靠可用性**。

Hadoop按位存储和处理数据的能力值得人们信赖，同时hadoop高可用体现在，如果其中的一台服务器宕机，则不影响整个数据集群，仍可以正常运行。

1. **高扩展性**。

Hadoop是在可用的计算机集簇间分配数据并完成计算任务的，这些集簇可以方便地扩展到数以千计的节点中。

1. **高效性**。

Hadoop能够在节点之间动态地移动数据，并保证各个节点的[动态平衡](https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E5%B9%B3%E8%A1%A1" \t "_blank)，因此处理速度非常快。

1. **高容错性**。

Hadoop能够自动保存数据的多个副本，并且能够自动将失败的任务重新分配。

1. **低成本**。

与一体机、商用数据仓库以及QlikView、Yonghong Z-Suite等数据集市相比，hadoop是开源的，项目的软件成本因此会大大降低。

# Hadoop的生态圈组件

## HDFS

HDFS采用了主从（Master/Slave）结构模型，一个HDFS集群是由一个NameNode和若干个DataNode组成的。其中NameNode作为主服务器，管理文件系统的命名空间和客户端对文件的访问操作；集群中的DataNode管理存储的数据。

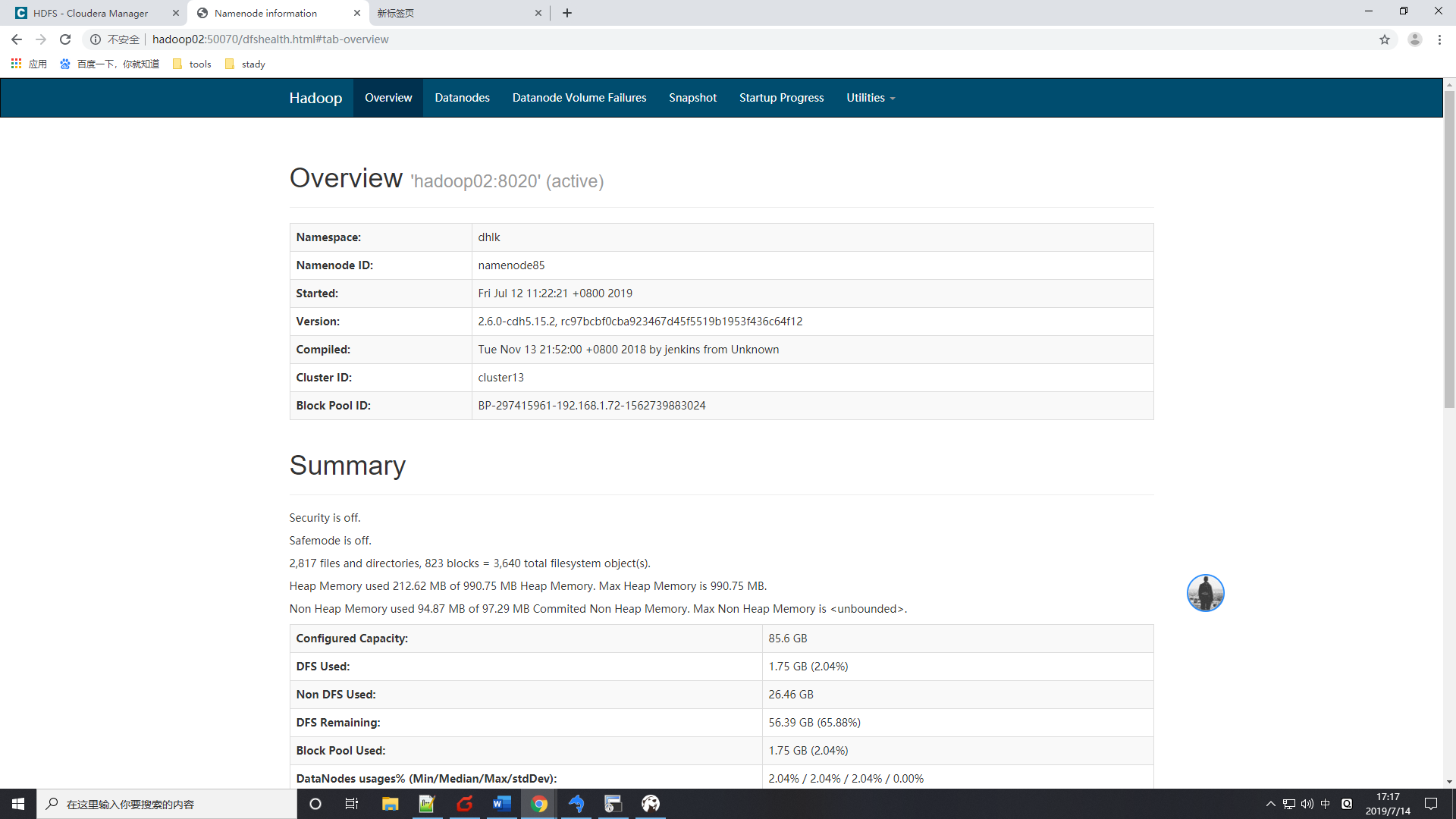
### **HDFS的组成**

1. NameNode：Master节点（也称元数据节点），是系统唯一的管理者。负责元数据的管理(名称空间和数据块映射信息);配置副本策略；处理客户端请求。
2. DataNode：数据存储节点(也称Slave节点)，存储实际的数据；执行数据块的读写；汇报存储信息给NameNode。
3. JournalNode：两个NameNode为了数据同步，会通过一组称作JournalNodes的独立进程进行相互通信。当active状态的NameNode的命名空间有任何修改时，会告知大部分的JournalNodes进程。standby状态的NameNode有能力读取JNs中的变更信息，并且一直监控edit log的变化，把变化应用于自己的命名空间。standby可以确保在集群出错时，命名空间状态已经完全同步了。
4. client：系统使用者，调用HDFS API操作文件;与NameNode交互获取文件元数据;与DataNode交互进行数据读写。注意：写数据时文件切分由Client完成。

### hdfs构架原则

1. 元数据与数据分离：文件本身的属性（即元数据）与文件所持有的数据分离。
2. 主/从架构：一个HDFS集群是由一个NameNode和一定数目的DataNode组成。
3. 一次写入多次读取：HDFS中的文件在任何时间只能有一个Writer。当文件被创建，接着写入数据，最后，一旦文件被关闭，就不能再修改。

### Hdfs的页面及命令



HDSF上操作的一些常用命令：



wordcount样例：

hadoop fs -rm -r /user/hadoop/examples/input

hadoop fs -mkdir -p /user/hadoop/examples/input

hadoop fs -put /opt/cloudera/parcels/CDH-5.15.2-1.cdh5.15.2.p0.3/lib/hadoop/LICENSE.txt /user/hadoop/examples/input

hadoop fs -rm -r /user/hadoop/examples/output-data/wordcount

hadoop jar /opt/cloudera/parcels/CDH-5.15.2-1.cdh5.15.2.p0.3/share/doc/hadoop-0.20-mapreduce/examples/hadoop-examples.jar wordcount /user/hadoop/examples/input /user/hadoop/examples/output-data/wordcount

## [**Zookeeper**](http://zookeeper.apache.org/)**（分布式协作服务）**

### Zookeeper是什么

它是一个分布式服务框架，是Apache Hadoop 的一个子项目，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等

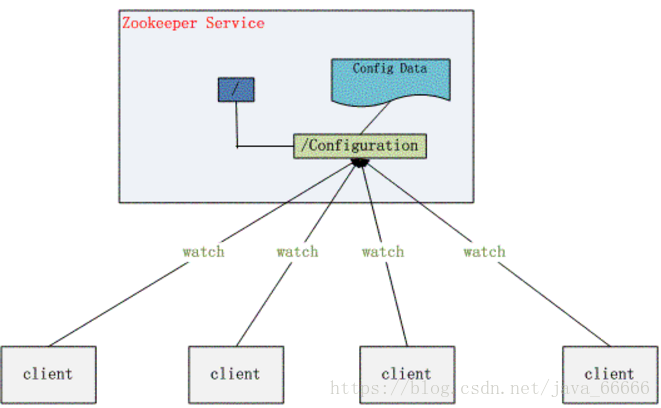
Hadoop的许多组件依赖于Zookeeper，它运行在计算机集群上面，用于管理Hadoop操作。

监听通知机制：客户端注册监听它关心的目录节点，当目录节点发生变化（数据改变、被删除、子目录节点增加删除）时，zookeeper会通知客户端。

### Zookeeper的作用

可以实现诸如分布式应用配置管理、统一命名服务、状态同步服务、集群管理等功能，举例说明：

假设我们的程序是分布式部署在多台机器上，如果我们要改变程序的配置文件，需要逐台机器去修改，非常麻烦，现在把这些配置全部放到zookeeper上去，保存在 zookeeper 的某个目录节点中，然后所有相关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 zookeeper 的通知，然后从 zookeeper 获取新的配置信息应用到系统中。



## [Yarn](http://hadoop.apache.org/)****(分布式资源管理器）****

### ****Yarn的概念****

Apache Hadoop YARN （Yet Another Resource Negotiator，另一种资源协调者）是一种新的 Hadoop 资源管理器，它是一个通用资源管理系统，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。

YARN是下一代MapReduce，即MRv2，是在第一代MapReduce基础上演变而来的，主要是为了解决原始Hadoop扩展性较差，不支持多计算框架而提出的。

### ****Yarn的构成****

Yarn主要由ResourceManager、NodeManager、ApplicationMaster和Container等组件构成。Yarn框架执行主要功能，即在集群中调度资源。

#### Resource Manager

作为全局统一的资源管理、调度、分配。Resource Manager由Scheduler和Applicatio Manager组成。Scheduler根据节点的容量、队列情况，为Application分配资源；Application Manager接受用户提交的请求，在节点中启动Application Master，并监控Application Master的状态、进行必要的重启。

#### Node Manager

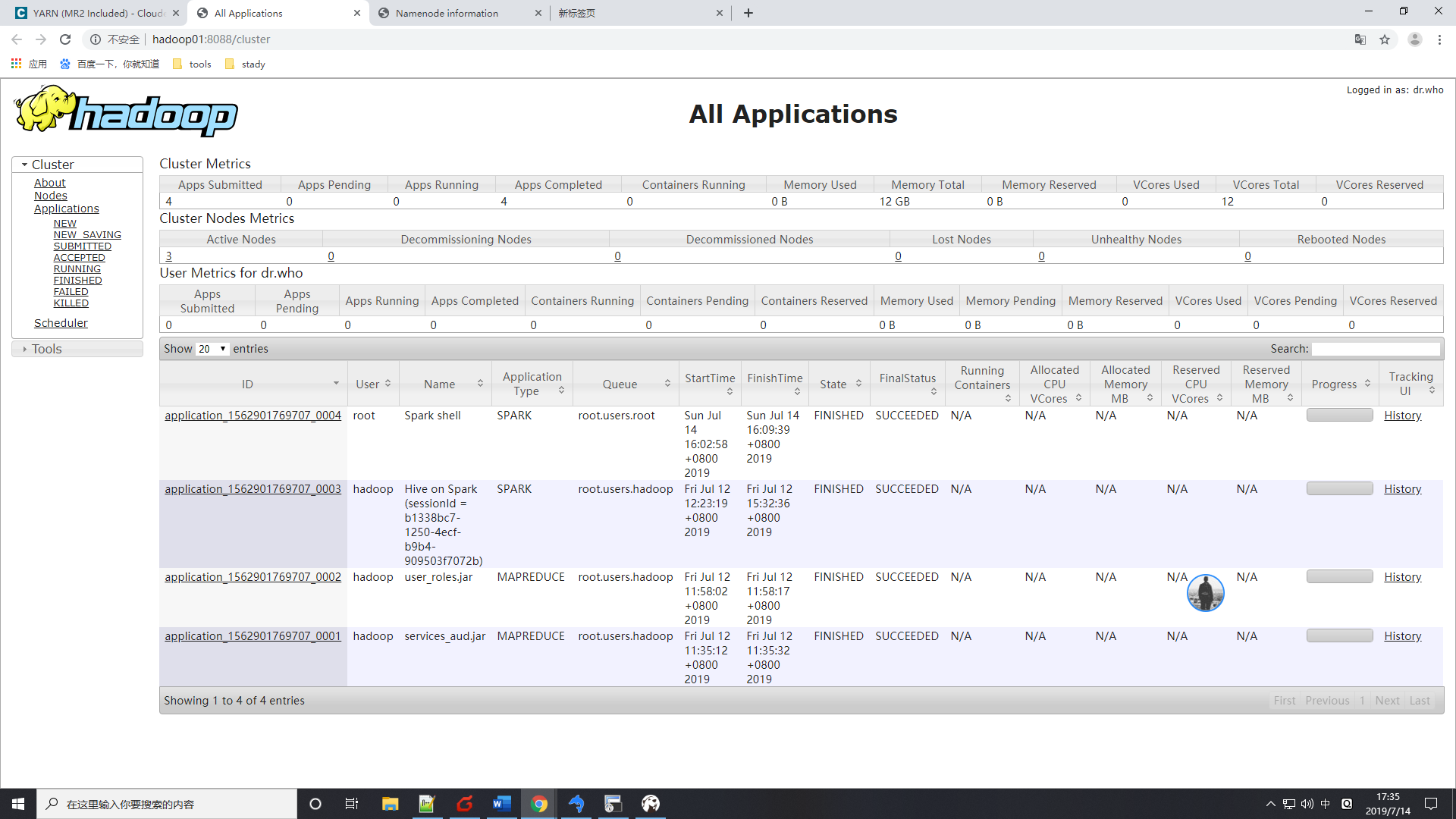
在每一个节点上都有一个Node Manager作为代理监控节点的资源使用情况（cpu, memory, disk, network）并向Resource Manager上报节点状态。

#### Application Master

负责数据处理job的执行调度。Application Master与Resource Manager进行沟通，获取资源进行计算。得到资源后，与节点上的Node Manager进行沟通，在分配的Container汇总执行任务，并监控任务执行的情况。

#### Container

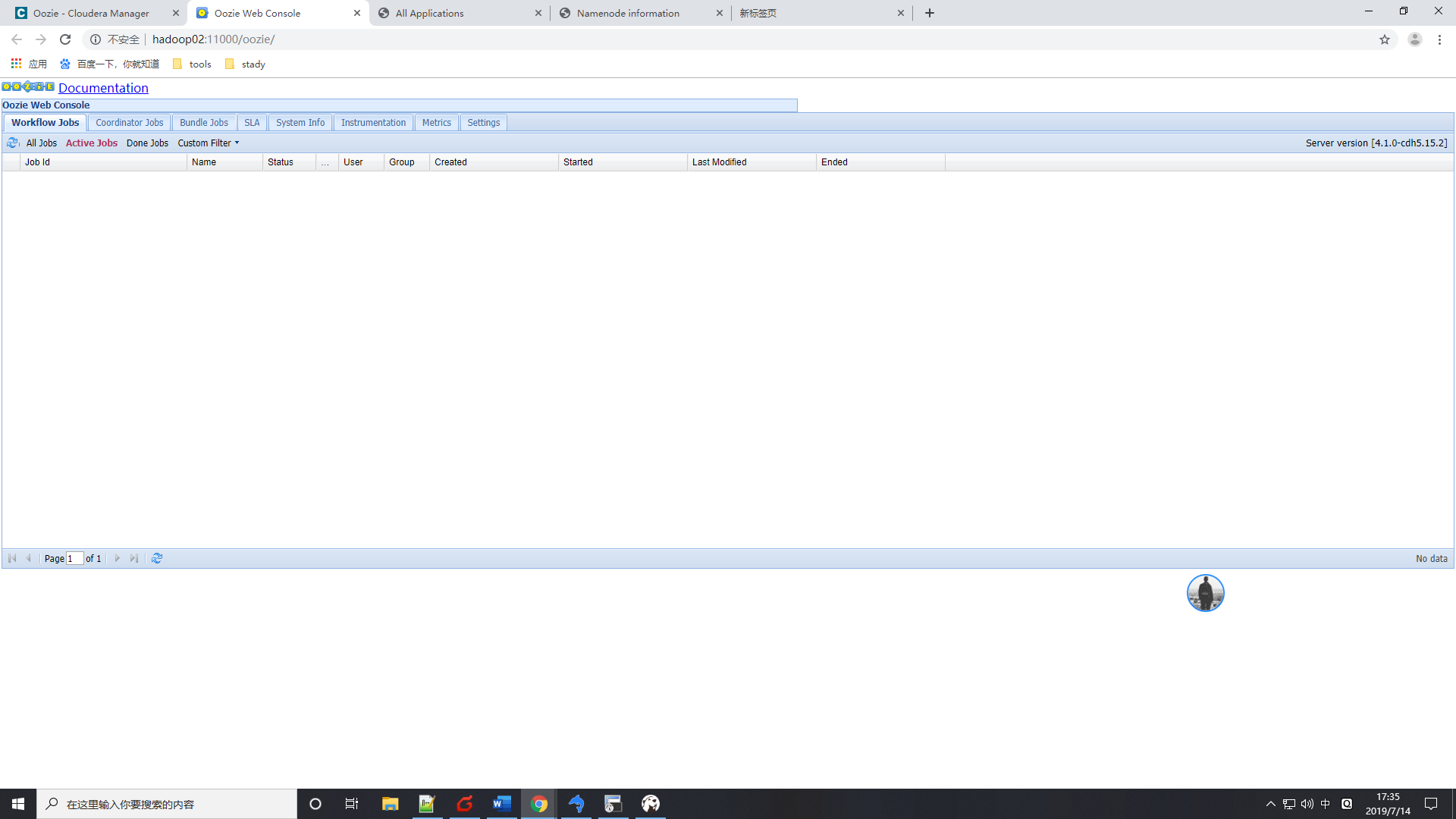
资源的一种抽象方式，它封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等，当Application Master向Resource Manager申请资源时，Resource Manager为Application Master返回的资源便是Container。



## [Oozie](http://oozie.apache.org/)****(工作流调度器）****

Oozie是一个可扩展的工作体系，集成于Hadoop的堆栈，用于协调多个MapReduce作业的执行。它能够管理一个复杂的系统，基于外部事件来执行，外部事件包括数据的定时和数据的出现。

Oozie工作流是放置在控制依赖DAG（有向无环图 Direct Acyclic Graph）中的一组动作（例如，Hadoop的Map/Reduce作业、Pig作业等），其中指定了动作执行的顺序。Oozie使用hPDL（一种XML流程定义语言）来描述这个图。

oozie job -oozie http://hadoop02:11000/oozie -config /home/hadoop/examples/oozie/examples/apps/map-reduce/job.properties -run

## [Sqoop](http://sqoop.apache.org/)****(数据ETL/同步工具）****

### [Sqoop](http://sqoop.apache.org/)****概念****

Sqoop是SQL-to-Hadoop的缩写，主要用于传统数据库和Hadoop之前传输数据。数据的导入和导出本质上是Mapreduce程序，充分利用了MR的并行化和容错性。Sqoop利用数据库技术描述数据架构，用于在关系数据库、数据仓库和Hadoop之间转移数据。

### 实例

使用sqoop将数据从mysql导入hive(hive中不需要提前建表)

sqoop import --connect jdbc:mysql://hadoop01/test01?characterEncoding=utf-8 --username hadoop --password hadoop --table my\_user --hive-import --hive-table test01.my\_user

sqoop import --connect jdbc:mysql://hadoop01/test01?characterEncoding=utf-8 --username hadoop --password hadoop --table services --hive-import --hive-table test01.services

使用sqoop将数据从hive导出mysql(mysql中需要提前建表)

sqoop-export --connect jdbc:mysql://hadoop01:3306/test01?characterEncoding=UTF-8 --username root --password root --table my\_user01 --export-dir "/user/hive/warehouse/test01.db/my\_user" --input-fields-terminated-by '\001'

## [HIVE](http://hive.apache.org/)****（数据仓库）****

### Hive的概念

Hive定义了一种类似SQL的查询语言(HQL),将SQL转化为MapReduce任务在Hadoop上执行。通常用于离线分析。

HQL用于运行存储在Hadoop上的查询语句，Hive让不熟悉MapReduce开发人员也能编写数据查询语句，然后这些语句被翻译为Hadoop上面的MapReduce任务。

### Hive的设计特点

● 支持创建索引，优化数据查询。

● 不同的存储类型，例如，纯文本文件、HBase 中的文件。

● 将元数据保存在关系数据库中，大大减少了在查询过程中执行语义检查的时间。

● 可以直接使用存储在Hadoop 文件系统中的数据。

● 内置大量用户函数UDF 来操作时间、字符串和其他的数据挖掘工具，支持用户扩展UDF 函数来完成内置函数无法实现的操作。

● 类SQL 的查询方式，将SQL 查询转换为MapReduce 的job 在Hadoop集群上执行。[1]

### Hive 中的表分类

Hive的表一般分为4类：内部表、外部表、分区表和 Bucket 表

* **内部表和外部表的区别**

**删除内部表，删除表元数据和数据**

**删除外部表，删除元数据，不删除数据**

* **内部表和外部表的使用选择**

　　大多数情况，他们的区别不明显，如果数据的所有处理都在 Hive 中进行，那么倾向于 选择内部表，但是如果 Hive 和其他工具要针对相同的数据集进行处理，外部表更合适。

　　使用外部表访问存储在 HDFS 上的初始数据，然后通过 Hive 转换数据并存到内部表中

　　使用外部表的场景是针对一个数据集有多个不同的 Schema

　　通过外部表和内部表的区别和使用选择的对比可以看出来，hive 其实仅仅只是对存储在 HDFS 上的数据提供了一种新的抽象。而不是管理存储在 HDFS 上的数据。所以不管创建内部 表还是外部表，都可以对 hive 表的数据存储目录中的数据进行增删操作。

* **分区表和分桶表的区别**

　　Hive 数据表可以根据某些字段进行分区操作，细化数据管理，可以让部分查询更快。同时表和分区也可以进一步被划分为 Buckets，分桶表的原理和 MapReduce 编程中的 HashPartitioner 的原理类似。

分区和分桶都是细化数据管理，但是分区表是手动添加区分，由于 Hive 是读模式，所 以对添加进分区的数据不做模式校验，分桶表中的数据是按照某些分桶字段进行 hash 散列 形成的多个文件，所以数据的准确性也高很多。

### 实例

创建mysql测试库test01

CREATE DATABASE test01 DEFAULT CHARSET utf8 COLLATE utf8\_general\_ci;

建立mysql表

CREATE TABLE test01.my\_user(

`id` int(4) NOT NULL AUTO\_INCREMENT,

`account` varchar(255) DEFAULT NULL,

`passwd` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

);

插入数据

INSERT INTO `my\_user` VALUES ('1', 'admin', 'admin');

INSERT INTO `my\_user` VALUES ('3', 'system', 'system');

INSERT INTO `my\_user` VALUES ('5', 'lee', 'lee');

INSERT INTO `my\_user` VALUES ('6', 'les', 'les');

INSERT INTO `my\_user` VALUES ('7', 'jianxin', 'jianxin');

## [HBASE](http://hbase.apache.org/)****（分布式列存数据库）****

### Hbase概念

HBase是一个建立在HDFS之上，面向列的针对结构化数据的可伸缩、高可靠、高性能、分布式和面向列的动态模式数据库。

HBase采用了BigTable的数据模型：增强的稀疏排序映射表（Key/Value），其中，键由行关键字、列关键字和时间戳构成。HBase提供了对大规模数据的随机、实时读写访问，同时，HBase中保存的数据可以使用MapReduce来处理，它将数据存储和并行计算完美地结合在一起。

### Hbase系统架构

Hbase由HMaster和HRegionServer两部分组成，同时由zookeeper进行服务管理。

ClientHBase Client使用HBase的RPC机制与HMaster和HRegionServer进行通信，对于管理类操作，Client与HMaster进行RPC；对于数据读写类操作，Client与HRegionServer进行RPC。

#### HMaster

HMaster没有单点问题，HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master运行，HMaster在功能上主要负责Table和Region的管理工作：

* 管理用户对Table的增、删、改、查操作
* 管理HRegionServer的负载均衡，调整Region分布
* 在Region Split后，负责新Region的分配
* 在HRegionServer停机后，负责失效HRegionServer 上的Regions迁移

#### HRegionServer

HRegionServer主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBase中最核心的模块。

### Hbase命令及实例

status

create 'default:testtable','colfaml'

list

list 'testtable'

put 'testtable','myrow-1','colfaml:q1','value-1'

put 'testtable','myrow-2','colfaml:q2','value-2'

put 'testtable','myrow-3','colfaml:q3','value-3'

put 'testtable','myrow-2','colfaml:q3','value-3'

scan 'testtable'

get 'testtable','myrow-1'

get 'testtable','myrow-2'

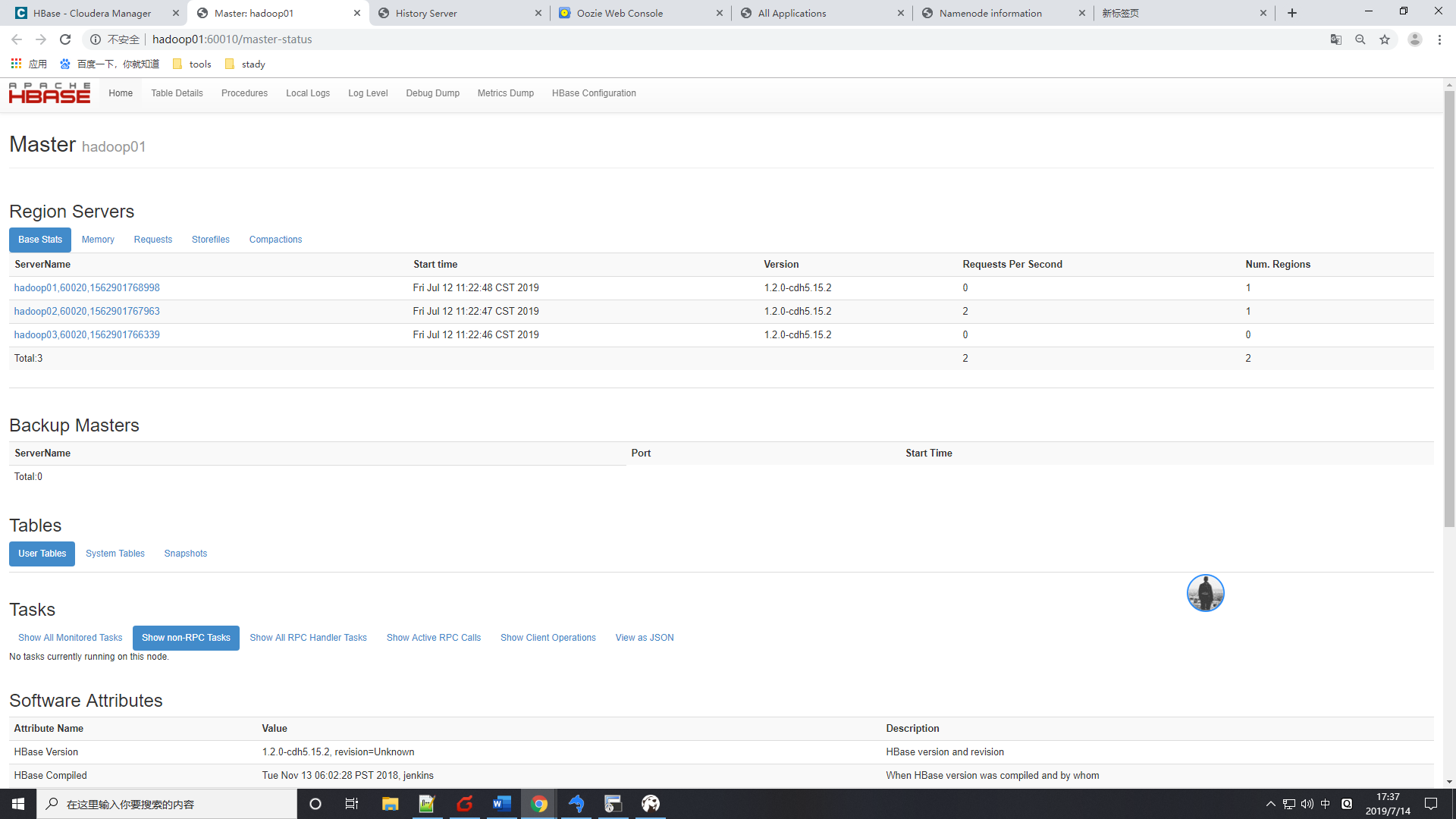
delete 'testtable','myrow-2','colfaml:q3'

scan 'testtable'

disable 'testtable'

drop 'testtable'

list

exit

## [Spark](http://spark.apache.org/)****(内存DAG计算模型)****

### ****Spark概念****

Spark是一个Apache项目，它被标榜为“快如闪电的集群计算”。它拥有一个繁荣的开源社区，并且是目前最活跃的Apache项目。Spark使用Scala语言进行实现，它是一种面向对象、函数式编程语言，能够像操作本地集合对象一样轻松地操作分布式数据集。另外，Spark不仅支持Scala编写应用程序，而且支持Java和Python等语言进行编写，特别是Scala是一种高效、可拓展的语言，能够用简洁的代码处理较为复杂的处理工作。

Spark提供了一个更快、更通用的数据处理平台。和Hadoop相比，Spark可以让你的程序在内存中运行时速度提升100倍，或者在磁盘上运行时速度提升10倍。

### Spark的适用场景

* + 复杂的批量处理（Batch Data Processing），偏重点在于处理海量数据的能力，至于处理速度可忍受，通常的时间可能是在数十分钟到数小时；
  + 基于历史数据的交互式查询（Interactive Query），通常的时间在数十秒到数十分钟之间。
  + 基于实时数据流的数据处理（Streaming Data Processing），通常在数百毫秒到数秒之间。

目前对以上三种场景需求都有比较成熟的处理框架，第一种情况可以用Hadoop的MapReduce来进行批量海量数据处理，第二种情况可以Impala进行交互式查询，对于第三中情况可以用Storm分布式处理框架处理实时流式数据。

### Spark分类

#### Spark-shell

spark的shell作为一个强大的交互式数据分析工具，它可以使用Scala(在Java虚拟机上运行现有的Java库的一个很好方式)或Python在Spark目录里使用下面的方式开始运行：

**./bin/spark-shell**

#### Spark Streaming

SparkStreaming是一个对实时数据流进行高通量、容错处理的流式处理系统，可以对多种数据源（如Kdfka、Flume、Twitter、Zero和TCP 套接字）进行类似Map、Reduce和Join等复杂操作，并将结果保存到外部文件系统、数据库或应用到实时仪表盘。

#### Spark SQL

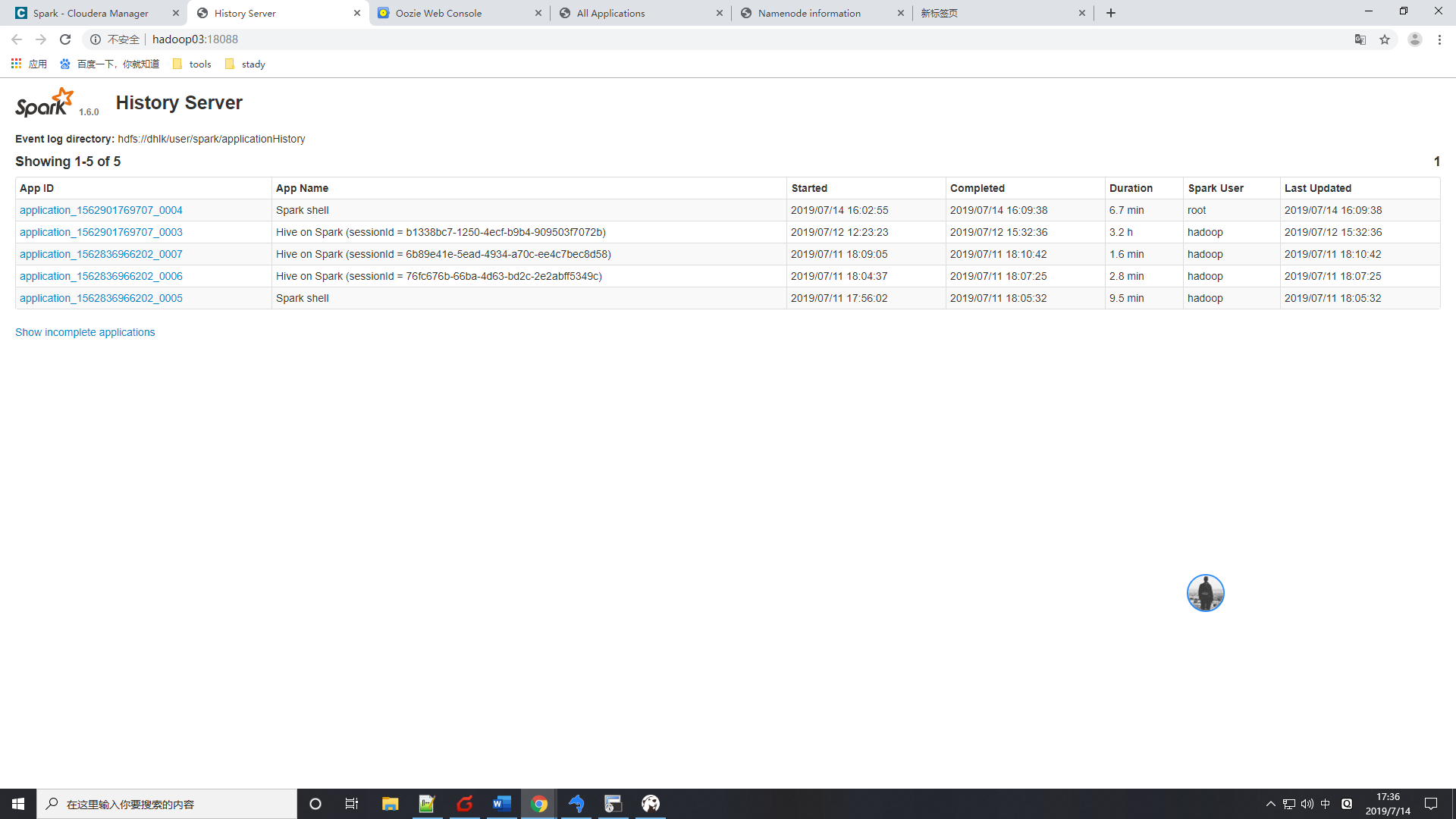
Spark SQL是Spark处理数据的一个模块，跟基本的Spark RDD的API不同，Spark SQL中提供的接口将会提供给Spark更多关于结构化数据和计算的信息。其本质是，Spark SQL使用这些额外的信息去执行额外的优化，这儿有几种和Spark SQL进行交互的方法，包括SQL和Dataset API，当使用相同的执行引擎时，API或其它语言对于计算的表达都是相互独立的，这种统一意味着开发人员可以轻松地在不同的API之间进行切换。

### Spark实例

计算圆周率的例子：

spark-submit --class org.apache.spark.examples.SparkPi --master yarn-client /opt/cloudera/parcels/CDH/lib/spark/examples/lib/spark-examples-1.6.0-cdh5.15.2-hadoop2.6.0-cdh5.15.2.jar

spark-submit --class org.apache.spark.examples.SparkPi --master yarn-cluster /opt/cloudera/parcels/CDH/lib/spark/examples/lib/spark-examples-1.6.0-cdh5.15.2-hadoop2.6.0-cdh5.15.2.jar



## ****Impala****

### ****Impala概念****

Impala是用于处理存储在Hadoop集群中的大量数据的MPP（大规模并行处理）SQL查询引擎。 它是一个用C ++和Java编写的开源软件。 与其他Hadoop的SQL引擎相比，它提供了高性能和低延迟。换句话说，Impala是性能最高的SQL引擎（提供类似RDBMS的体验），它提供了访问存储在Hadoop分布式文件系统中的数据的最快方法。

Impala通过使用标准组件（如HDFS，HBase，Metastore，YARN和Sentry）将传统分析数据库的SQL支持和多用户性能与Apache Hadoop的可扩展性和灵活性相结合。使用Impala，与其他SQL引擎（如Hive）相比，用户可以使用SQL查询以更快的方式与HDFS或HBase进行通信。

Impala可以读取Hadoop使用的几乎所有文件格式，如Parquet，Avro，RCFile。Impala将相同的元数据，SQL语法（Hive SQL），ODBC驱动程序和用户界面（Hue Beeswax）用作Apache Hive，为面向批量或实时查询提供熟悉且统一的平台。与Apache Hive不同，Impala不基于MapReduce算法。 它实现了一个基于守护进程的分布式架构，它负责在同一台机器上运行的查询执行的所有方面。因此，它减少了使用MapReduce的延迟，这使Impala比Apache Hive快。

刷新元数据：INVALIDATE METADATA;

### ****Impala与hive的关系****

Impala 与Hive都是构建在Hadoop之上的数据查询工具各有不同的侧重适应面，但从客户端使用来看Impala与Hive有很多的共同之处，如数据表元数 据、ODBC/JDBC驱动、SQL语法、灵活的文件格式、存储资源池等。Impala与Hive在Hadoop中的关系如图 2所示。Hive适合于长时间的批处理查询分析，而Impala适合于实时交互式SQL查询，Impala给数据分析人员提供了快速实验、验证想法的大数 据分析工具。可以先使用hive进行数据转换处理，之后使用Impala在Hive处理后的结果数据集上进行快速的数据分析。

### ****Impala与hive的异同****

**数据存储：**使用相同的存储数据池都支持把数据存储于HDFS, HBase。

**元数据：**两者使用相同的元数据。

**SQL解释处理：**比较相似都是通过词法分析生成执行计划。

**执行计划：**

* **Hive**: 依赖于MapReduce执行框架，执行计划分成 map->shuffle->reduce->map->shuffle->reduce…的模型。如果一个Query会 被编译成多轮MapReduce，则会有更多的写中间结果。由于MapReduce执行框架本身的特点，过多的中间过程会增加整个Query的执行时间。
* **Impala**: 把执行计划表现为一棵完整的执行计划树，可以更自然地分发执行计划到各个Impalad执行查询，而不用像Hive那样把它组合成管道型的 map->reduce模式，以此保证Impala有更好的并发性和避免不必要的中间sort与shuffle。

**数据流：**

* **Hive**: 采用推的方式，每一个计算节点计算完成后将数据主动推给后续节点。
* **Impala**: 采用拉的方式，后续节点通过getNext主动向前面节点要数据，以此方式数据可以流式的返回给客户端，且只要有1条数据被处理完，就可以立即展现出来，而不用等到全部处理完成，更符合SQL交互式查询使用。

**内存使用：**

* **Hive:** 在执行过程中如果内存放不下所有数据，则会使用外存，以保证Query能顺序执行完。每一轮MapReduce结束，中间结果也会写入HDFS中，同样由于MapReduce执行架构的特性，shuffle过程也会有写本地磁盘的操作。
* **Impala:** 在遇到内存放不下数据时，当前版本1.0.1是直接返回错误，而不会利用外存，以后版本应该会进行改进。这使用得Impala目前处理Query会受到一 定的限制，最好还是与Hive配合使用。Impala在多个阶段之间利用网络传输数据，在执行过程不会有写磁盘的操作（insert除外）。

**任务调度：**

* **Hive:** 任务调度依赖于Hadoop的调度策略。
* **Impala:** 调度由自己完成，目前只有一种调度器simple-schedule，它会尽量满足数据的局部性，扫描数据的进程尽量靠近数据本身所在的物理机器。调度器 目前还比较简单，在SimpleScheduler::GetBackend中可以看到，现在还没有考虑负载，网络IO状况等因素进行调度。但目前 Impala已经有对执行过程的性能统计分析，应该以后版本会利用这些统计信息进行调度吧。

**容错：**

* **Hive:** 依赖于Hadoop的容错能力。
* **Impala:** 在查询过程中，没有容错逻辑，如果在执行过程中发生故障，则直接返回错误（这与Impala的设计有关，因为Impala定位于实时查询，一次查询失败， 再查一次就好了，再查一次的成本很低）。但从整体来看，Impala是能很好的容错，所有的Impalad是对等的结构，用户可以向任何一个 Impalad提交查询，如果一个Impalad失效，其上正在运行的所有Query都将失败，但用户可以重新提交查询由其它Impalad代替执行，不 会影响服务。对于State Store目前只有一个，但当State Store失效，也不会影响服务，每个Impalad都缓存了State Store的信息，只是不能再更新集群状态，有可能会把执行任务分配给已经失效的Impalad执行，导致本次Query失败。

**适用面：**

* **Hive:** 复杂的批处理查询任务，数据转换任务。
* **Impala：**实时数据分析，因为不支持UDF，能处理的问题域有一定的限制，与Hive配合使用,对Hive的结果数据集进行实时分析。

## ****hue****

### ****Hue概念****

**Hue是cdh专门的一套web管理器，它包括3个部分hue ui，hue server，hue db。hue提供所有的cdh组件的shell界面的接口。你可以在hue编写mr，查看修改hdfs的文件，管理hive的元数据，运行Sqoop，编写Oozie工作流等大量工作。**

**官方demo运行界面：**

<https://demo.gethue.com/hue/accounts/login?next=/hue>

用户名：demo 密码：demo

