大数据库系统

第6章 数据仓库Hive

第6章 数据仓库Hive

- 主要内容

6.1 Hive简介

6.2 Hive的安装

6.3 Hive常用属性配置、交互指令以及数据类型简介

6.4 Hive的基本操作

6.5 HQL的基本使用

6.6 Hive中数据导入与导出

6.7 内部表、外部表和分区表

6.8 Hive窗口函数

6.9 hiveserver2与其他hive常用命令

6.1 Hive简介

- 主要内容

6.1.1 Hive的起源

6.1.2 Hive的特性

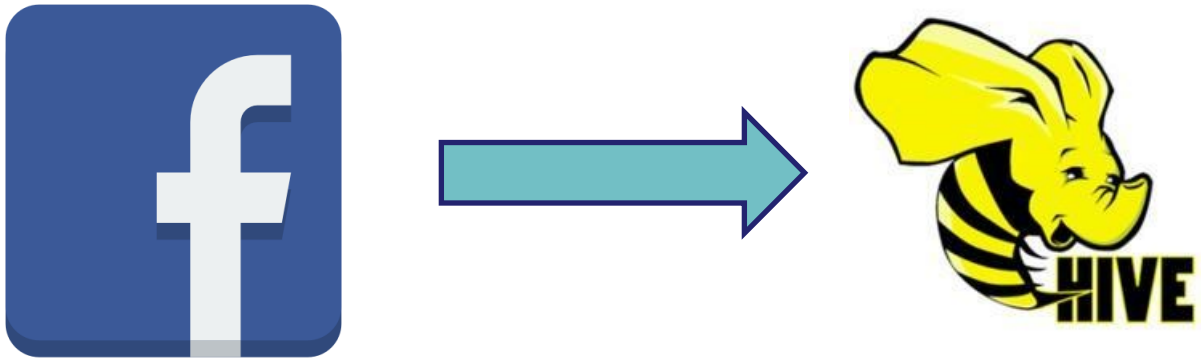
6.1.3 Hive的架构

6.1.4 Hive的运行机制

6.1.5 Hive的应用场景

6.1.1 Hive的起源

- Hive的起源
  - Hive是Facebook开发的，构建于Hadoop集群之上的数据仓库应用。2008年Facebook将Hive项目贡献给Apache，成为开源项目。



6.1.1 Hive的起源

- Hive的由来

1、Facebook的数据仓库一开始是构建于MySQL之上的

- 随着数据量的增加某些分析需要几个小时甚至几天的时间才能完成。

2、当数据仓库接近1T的时候，分析数据时Mysql的后台进程崩溃，这时决定移到Oracle

- 转移的过程也是付出了很大的代价

3、Oracle应付几T的数据还是没有问题的，但是在开始收集用户点击流的数据（每天大约400G）之后，Oracle也开始撑不住了，又要考虑新的数据仓库方案

6.1.1 Hive的起源

◆ Hive的起源

4、 Hadoop的出现有效解决了大规模数据的存储与统计分析的问题，但是MapReduce程序对于普通分析人员的使用过于复杂和繁琐

5、为了使公司中的分析人员更方便地在Hadoop下进行数据分析，开发了Hive

- Hive提供了类似于SQL的查询接口，降低了分析人员使用Hadoop进行数据分析的难度

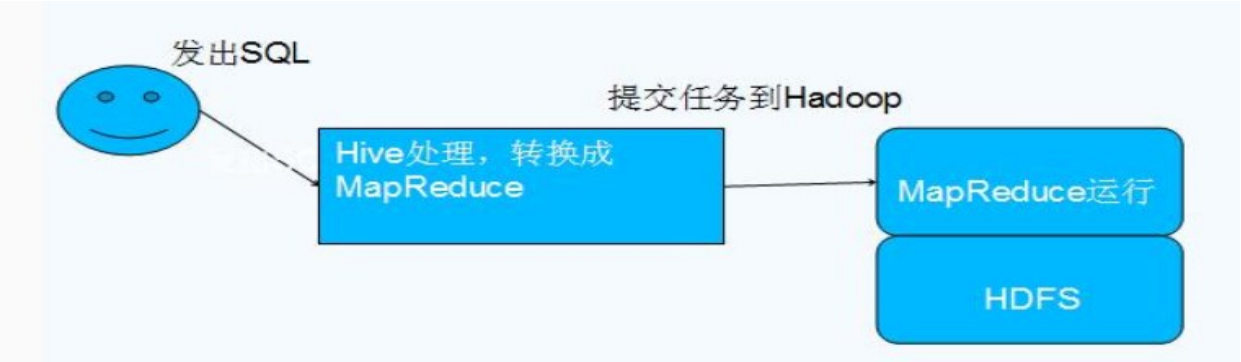
现在集群存储2.5PB的数据，并且以每天15TB的数据在增长，每天提交3000个以上的作业，大约处理55TB的数据...

6.1.1 Hive的起源

- Hive是什么？
  - Hive是基于Hadoop的数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供类SQL查询功能
    - HIVE是一个SQL解析引擎，将SQL（HQL）语句转译成M/R JOB然后在Hadoop执行

6.1.1 Hive的起源

- Hive与Hadoop的关系
  - Hadoop本身不能识别HQL，但是它通过Hive架构转化成Hadoop能识别的一个个MapReduce任务。
  - 对HQL查询语句的解释、优化、生成查询计划是由Hive完成的
  - 所有的数据都是存储在hadoop中



6.1.1 Hive的起源

- Hive是建构在Hadoop基础上的数据仓库工具

- 依赖于HDFS存储数据
- 依赖于MapReduce处理数据
- 借鉴SQL语言设计了新的查询语言HiveQL

6.1.2 Hive的特性

- Hive的特性

1、采用批处理方式处理海量数据

数据仓库存储的是静态数据，对静态数据的分析最适合采用批处理方式，不需要快速响应给出结果，数据本身也不会频繁发生变化

6.1.2 Hive的特性

◆ Hive的特性

2、可以存储、查询和分析存储在Hadoop中的大规模数据

- Hive 作为Hadoop的数据仓库处理工具，它所有的数据都存储在Hadoop兼容的文件系统中
- Hive 在加载数据过程中不会对数据进行任何的修改，只是将数据移动到HDFS中Hive 设定的目录下
- 因此，Hive 不支持对数据的改写和添加，所有的数据都是在加载的时候确定的

6.1.2 Hive的特性

- Hive的特性

3、支持索引，加快数据查询。

4、支持不同的存储类型，例如，纯文本文件、HBase 中的文件。

5、统一的元数据管理，可与impala/spark等共享元数据

6、提供了一系列对数据进行提取、转换、加载（ETL）的工具，这些工具能够很好地满足数据仓库各种应用场景

7、 灵活性和扩展性比较好，内置大量函数来操作时间、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

6.1.2 Hive的特性

- 例子

假如有这么一张表（实际的数据是海量的）

id	city	name	sex
0001	beijing	zhangli	man
0002	guizhou	lifang	woman
0003	tianjin	wangwei	man
0004	chengde	wanghe	woman
0005	beijing	lidong	man
0006	lanzhou	wuting	woman
0007	beijing	guona	woman
0008	chengde	houkuo	man

其中每一列按照"\t"来分隔，现在需要统计city为beijing的有几条记录

使用传统的MapReduce程序来实现

```java
package IT;

import java.io.IOException;
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.partition.HashPartitioner;

public class Consumer
{
    public static String path1 = "hdfs://192.168.80.80:9000/consumer.txt";
    public static String path2 = "hdfs://192.168.80.80:9000/dir";
    public static void main(String[] args) throws Exception
    {
          FileSystem fileSystem = FileSystem.get(new URI(path1) , new Configuration());
          if(fileSystem.exists(new Path(path2)))
          {
              fileSystem.delete(new Path(path2), true);
          }

          Job job = new Job(new Configuration(),"Consumer");
          FileInputFormat.setInputPaths(job, new Path(path1));
          job.setInputFormatClass(TextInputFormat.class);
          job.setMapperClass(MyMapper.class);
          job.setMapOutputKeyClass(Text.class);
          job.setMapOutputValueClass(LongWritable.class);

          job.setNumReduceTasks(1);
          job.setPartitionerClass(HashPartitioner.class);
          job.setReducerClass(MyReducer.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(LongWritable.class);
          job.setOutputFormatClass(TextOutputFormat.class);
          FileOutputFormat.setOutputPath(job, new Path(path2));
          job.waitForCompletion(true);
          //查看执行结果
          FSDataInputStream fr = fileSystem.open(new Path("hdfs://hadoop80:9000/dir/part-r-00000"));
          IOUtils.copyBytes(fr, System.out, 1024, true);
     }
    public static class MyMapper extends Mapper<LongWritable, Text, Text, LongWritable>
    {
           public static long sum = 0L;
           protected void map(LongWritable k1, Text v1,Context context) throws IOException, InterruptedException
           {
                 String[] splited = v1.toString().split("\t");
                 if(splited[1].equals("beijing"))
                 {
                     sum++;
                 }
           }
           protected void cleanup(Context context)throws IOException, InterruptedException
           {
                 String str = "beijing";
                 context.write(new Text(str),new LongWritable(sum));
           }
    }
    public static class MyReducer extends Reducer<Text, LongWritable, Text, LongWritable>
    {
           protected void reduce(Text k2, Iterable<LongWritable> v2s,Context context)throws IOException, InterruptedEx
           {
                 for (LongWritable v2 : v2s)
                {
                    context.write(k2, v2);
                }
           }
    }
}
```

经过上述代码，的到的结果如下：

```
16/05/22 17:16:39 INFO mapred.JobClient:   Map-Reduce Framework
16/05/22 17:16:39 INFO mapred.JobClient:     Spilled Records=2
16/05/22 17:16:39 INFO mapred.JobClient:     Map output materialized bytes=24
16/05/22 17:16:39 INFO mapred.JobClient:     Reduce input records=1
16/05/22 17:16:39 INFO mapred.JobClient:     Map input records=8
16/05/22 17:16:39 INFO mapred.JobClient:     SPLIT_RAW_BYTES=103
16/05/22 17:16:39 INFO mapred.JobClient:     Map output bytes=16
16/05/22 17:16:39 INFO mapred.JobClient:     Reduce shuffle bytes=0
16/05/22 17:16:39 INFO mapred.JobClient:     Reduce input groups=1
16/05/22 17:16:39 INFO mapred.JobClient:     Combine output records=0
16/05/22 17:16:39 INFO mapred.JobClient:     Reduce output records=1
16/05/22 17:16:39 INFO mapred.JobClient:     Map output records=1
16/05/22 17:16:39 INFO mapred.JobClient:     Combine input records=0
16/05/22 17:16:39 INFO mapred.JobClient:     Total committed heap usage (bytes)=321527808
16/05/22 17:16:39 INFO mapred.JobClient:   File Input Format Counters
16/05/22 17:16:39 INFO mapred.JobClient:     Bytes Read=209
16/05/22 17:16:39 INFO mapred.JobClient:   FileSystemCounters
16/05/22 17:16:39 INFO mapred.JobClient:     HDFS_BYTES_READ=418
16/05/22 17:16:39 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=129020
16/05/22 17:16:39 INFO mapred.JobClient:     FILE_BYTES_READ=338
16/05/22 17:16:39 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=10
16/05/22 17:16:39 INFO mapred.JobClient:   File Output Format Counters
16/05/22 17:16:39 INFO mapred.JobClient:     Bytes Written=10
beijing     3
```

可以看到输出结果为3

- 那么如果用hive来执行，如下所示

```
hive> select city,count(*)
    > from t4
    > where city='beijing'
    > group by city;
```


- 结果为：

```
hive> select city,count(*)
    > from t4
    > where city='beijing'
    > group by city;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1478233923484_0902, Tracking URL = http://hadoop22:8088/proxy/application_1478233923484_0902/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1478233923484_0902
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-11-09 11:36:36,688 Stage-1 map = 0%,  reduce = 0%
2016-11-09 11:36:42,018 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.21 sec
2016-11-09 11:36:43,062 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.21 sec
2016-11-09 11:36:44,105 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.21 sec
2016-11-09 11:36:45,149 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.21 sec
2016-11-09 11:36:46,193 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.21 sec
2016-11-09 11:36:47,237 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.21 sec
2016-11-09 11:36:48,283 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.21 sec
2016-11-09 11:36:49,329 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.7 sec
2016-11-09 11:36:50,384 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.7 sec
MapReduce Total cumulative CPU time: 3 seconds 700 msec
Ended Job = job_1478233923484_0902
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 3.7 sec   HDFS Read: 419 HDFS Write: 10 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 700 msec
OK
beijing 3
Time taken: 19.768 seconds, Fetched: 1 row(s)
```

6.1.2 Hive的特性

- Hive和数据库的异同

1、数据存储

2、数据格式

3、数据更新

4、执行延迟

6.1.2 Hive的特性

- Hive和数据库的异同

1、数据存储

- Hive 是建立在 Hadoop 之上的，所有 Hive 的数据都是存储在 HDFS 中的

- 普通关系数据库数据库则可以将数据保存在块设备或者本地文件系统中

6.1.2 Hive的特性

- Hive 和数据库的异同

2、数据格式

- Hive 中没有定义专门的数据格式，需要指定三个属性：列分隔符，行分隔符，以及读取文件数据的方法

- 数据库中，存储引擎定义了自己的数据格式。所有数据都会按照一定的组织存储

6.1.2 Hive的特性

- Hive 和数据库的异同

3、数据更新

- Hive是数据仓库，其特性决定了内容是读多写少，因此，不支持对数据的改写和删除，数据都是在加载的时候中确定好的

- 数据库中的数据通常是需要经常进行修改

6.1.2 Hive的特性

- Hive 和数据库的异同

4、执行延迟

- Hive在查询数据的时候，需要扫描整个表(或分区)，因此延迟较高，因此hive只有在处理大数据时才有优势
- 数据库在处理小数据时执行延迟较低

6.1.2 Hive的特性

- Hive与传统数据库的对比

对比项目	Hive	传统数据库
数据插入	支持批量导入	支持单条和批量导入
数据更新	不支持修改数据内容	支持
索引	支持	支持
分区	支持	支持
执行延迟	高	低
扩展性	好	有限

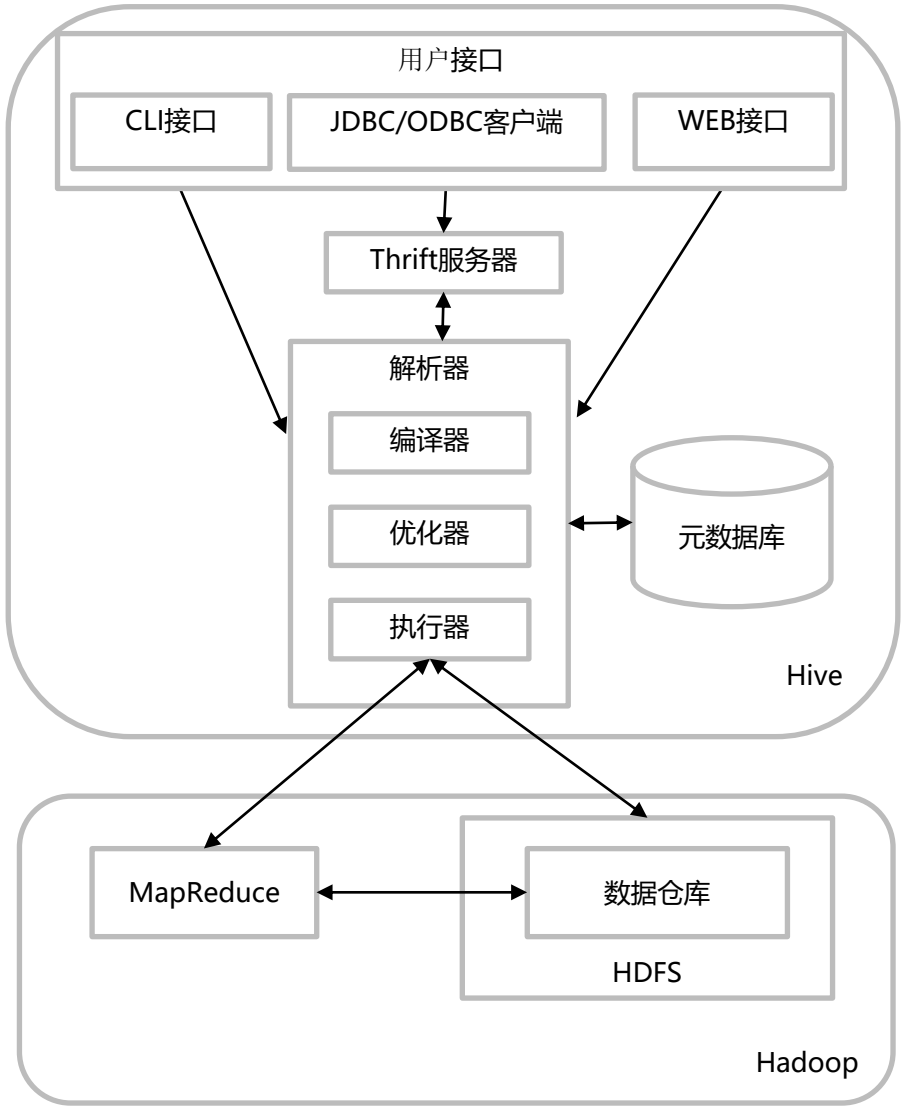
6.1.3 Hive的架构

- 主要分为3个部分

  用户接口

  Thrift服务器

  解析器（Driver）



6.1.3 Hive的架构

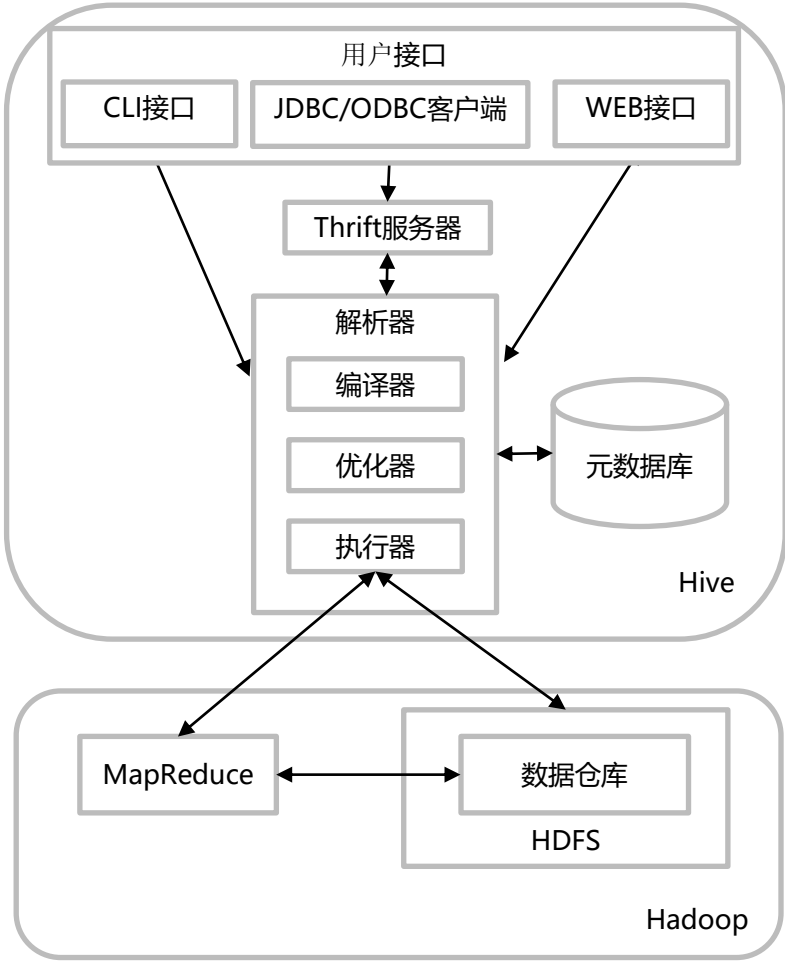
- 用户接口

CLI：Cli 启动的时候，会同时启动一个 Hive 副本

JDBC客户端：封装了Thrift、java应用程序，可以通过指定的主机和端口连接到在另一个进程中运行的hive服务器

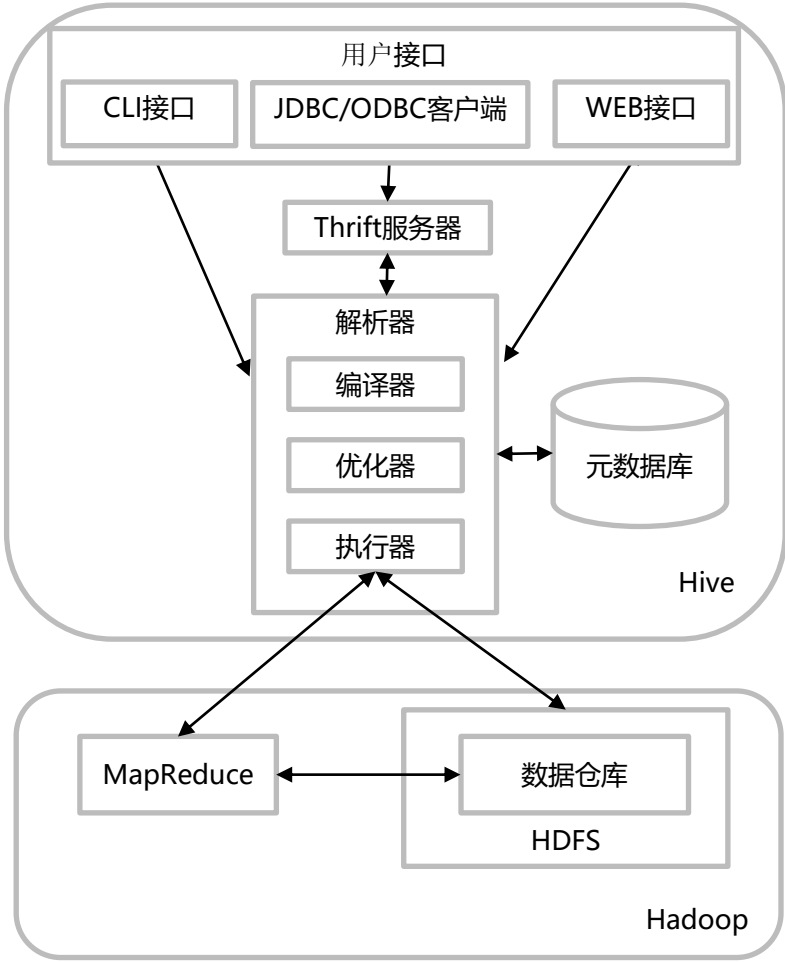
ODBC客户端：ODBC驱动允许支持ODBC协议的应用程序连接到Hive

WUI （Web GUI） 接口：是通过浏览器访问 Hive



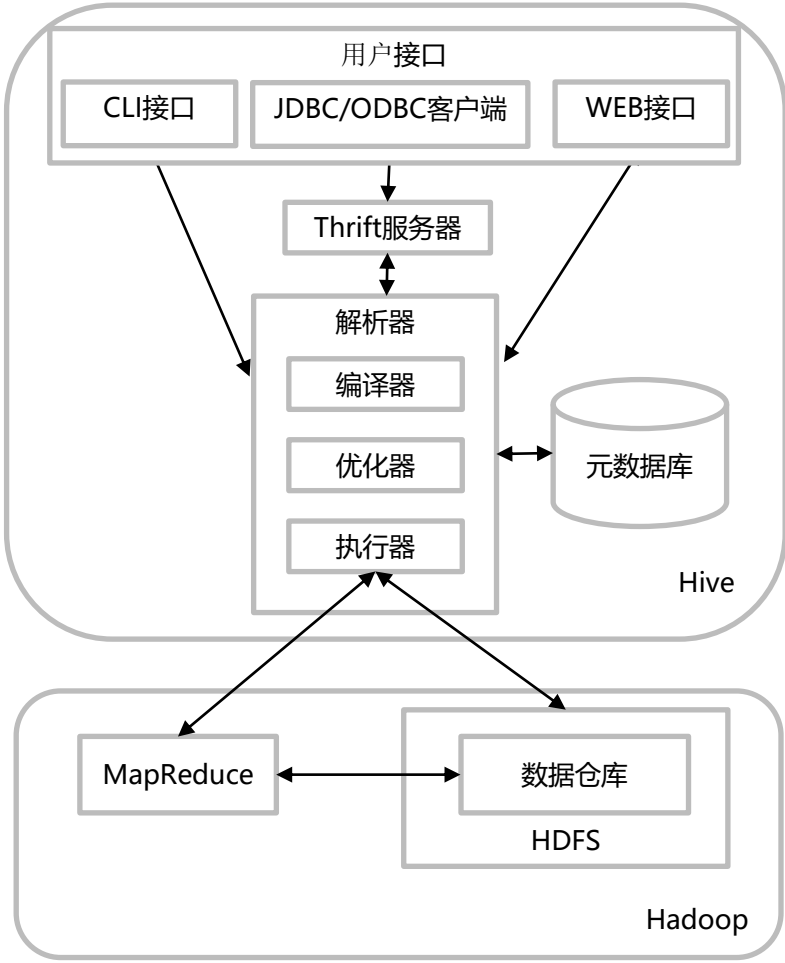
6.1.3 Hive的架构

- Thrift服务器
  - Thrift 是 Facebook 开发的一个软件框架
  - 用来进行可扩展且跨语言的服务的开发， 能让不同的编程语言调用 Hive 的接口
  - 支持C++、Java、PHP、Python和Ruby语言



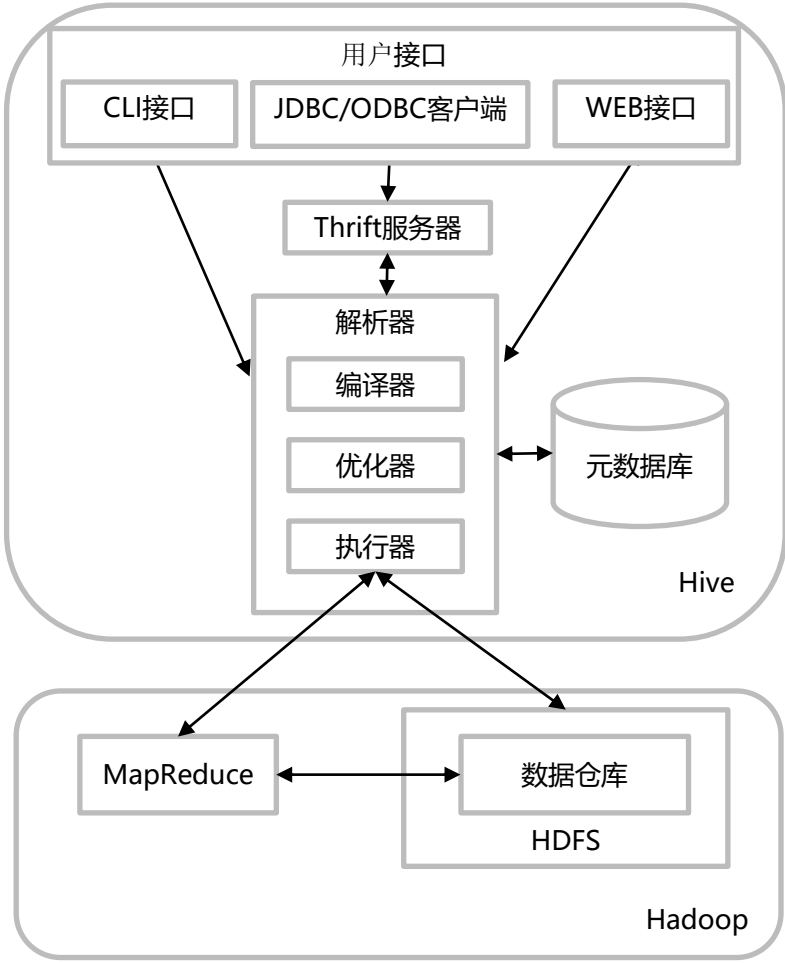
6.1.3 Hive的架构

- 解析器（Driver）
  - 包括编译器、优化器和执行器
  - 完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划的生成



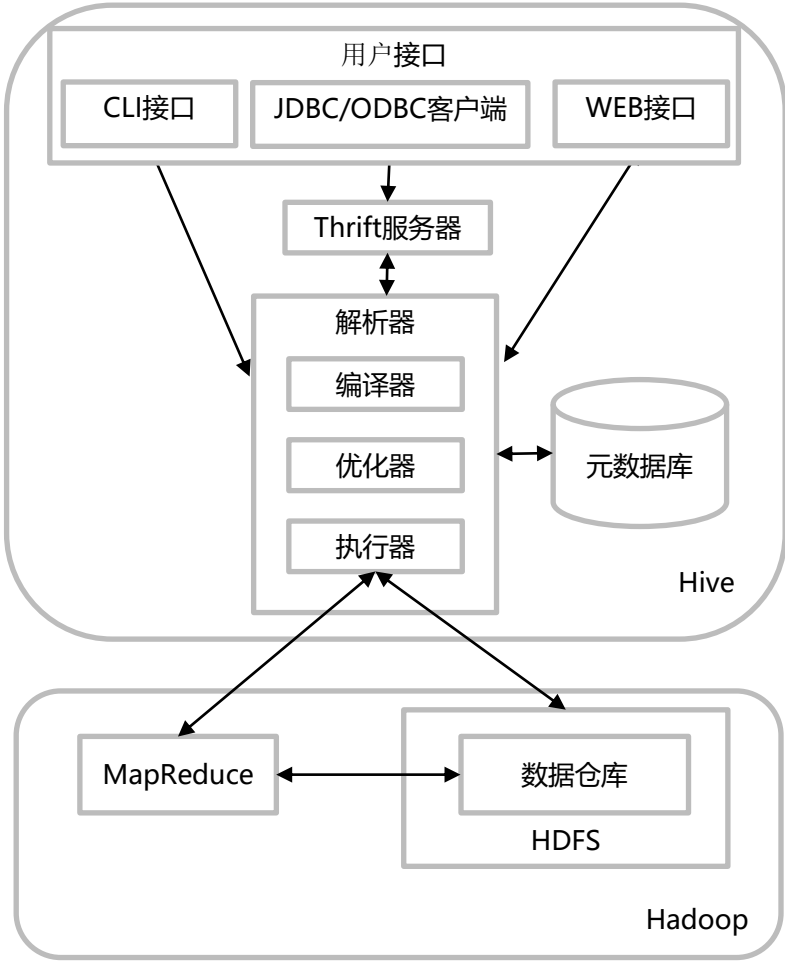
6.1.3 Hive的架构

- 元数据Metastore
  - Hive的数据由两部分组成：数据文件和元数据
  - 存放Hive库的基础信息，它存储在关系数据库中，如 mysql、derby
  - 包括了数据库信息、表的名字，表的列和分区及其属性，表的属性，表的数据所在目录等，不存储数据



6.1.3 Hive的架构

- Hadoop
  - Hive 的数据文件存储在 HDFS 中，大部分的查询由 MapReduce 完成
  - 对于包含 * 的查询，比如 select * from tbl 不会生成 MapRedcue 作业



6.1.4 Hive的运行机制

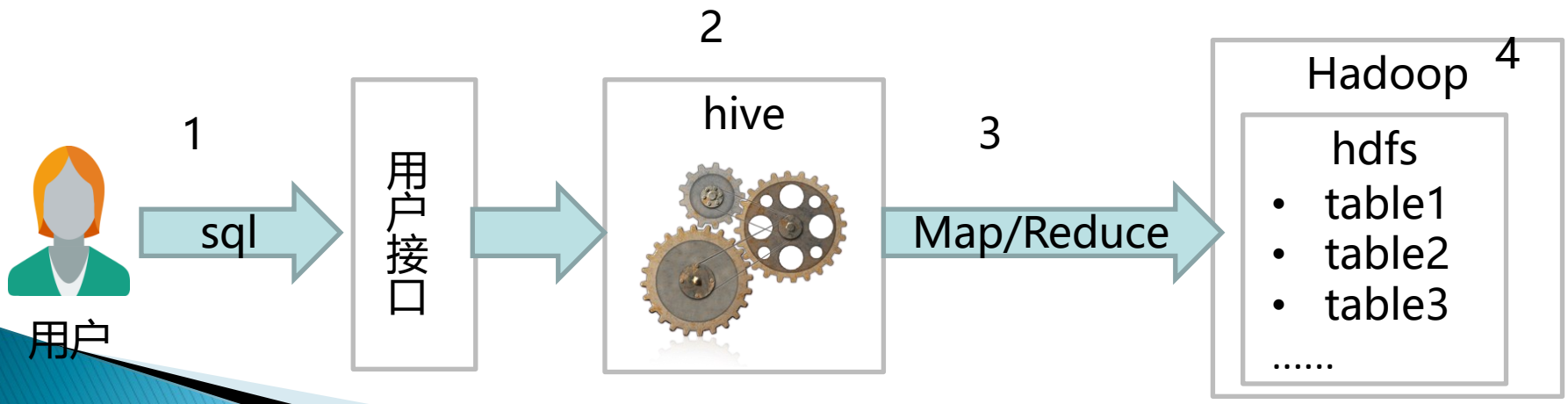
Hive的运行机制

1、用户通过用户接口连接Hive,发布Hive SQL

2、Hive解析查询并制定查询计划

3、Hive将查询转换成MapReduce作业

4、在Hadoop上执行MapReduce作业



6.1.5 Hive的应用场景

- HIVE的缺点
  - Hive的HQL表达能力有限，虽然支持大多数的SQL语句，但仍有部分无法表达
  - Hive的效率比较低，由于hive是基于hadoop，Hadoop本身是一个批处理，高延迟的计算框架，其计算是通过MapReduce来作业，具有**高延迟**性

6.1.5 Hive的应用场景

- 适用场景

  海量数据的存储处理

  海量数据的数据挖掘

  海量数据的离线分析

hive 的最佳使用场合是不可变的大数据集的批处理作业

6.1.5 Hive的应用场景

- Hive的应用

(1)日志分析：互联网公司使用hive进行日志分析

1）统计网站一个时间段内的pv、uv

- PV（Page View）访问量, 即页面浏览量或点击量，衡量网站用户访问的网页数量
- UV（Unique Visitor）独立访客，统计1天内访问某站点的用户数

2）用户消费行为记录

(2)海量结构化数据离线分析

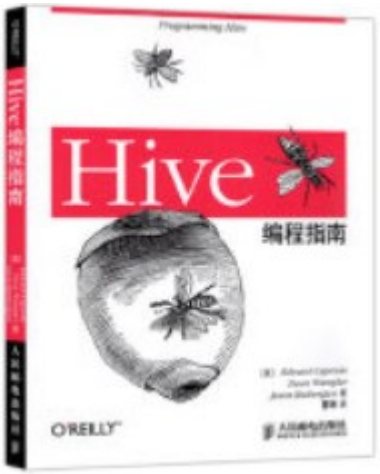
6.1.5 Hive的应用场景

◆ 不适用场景

联机交互式实时查询

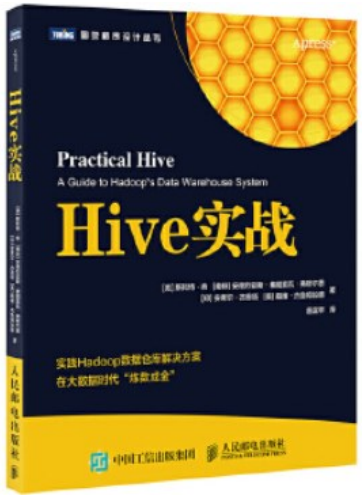
- Hadoop 通常都有较高的延迟并且在作业提交和调度的时候需要大量的开销
- Hive不能够在大规模数据集上实现低延迟快速的查询
- Hive在几百MB的数据集上执行查询一般有分钟级的时间延迟
- Hive并非为联机事务处理而设计
- Hive并不提供实时的查询和基于行级的数据更新操作

◆ 学习Hive推荐书籍



书名： Hive编程指南
作者： (美)卡普廖洛
出版社：人民邮电出版社

书名： Hive性能调优实战
作者：林志煌
出版社：机械工业出版社



书名： Hive实战
作者： 斯科特·肖
出版社：人民邮电出版社

总结

- Hive简介

  Hive的起源

  Hive的作用

  Hive的特性、优缺点

  Hive与传统数据库的区别

  Hive的应用场景

  Hive的系统架构