宽带网络监控教研中心

海量数据处理中的云计算

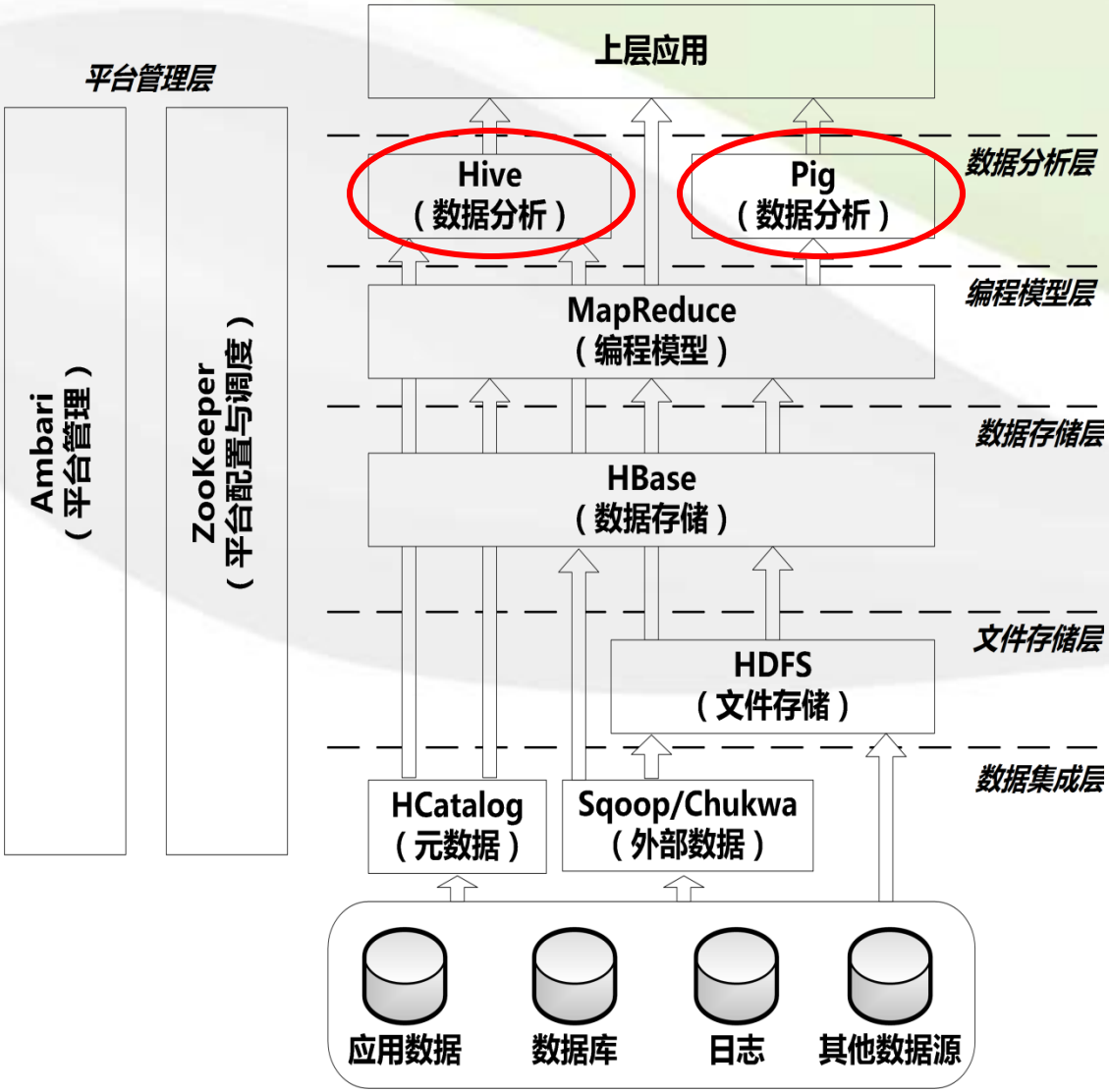
C10. Hive & Pig

北京邮电大学信息与通信工程学院

2013年春季学期

北京郵電大學
Beijing University of Posts and Telecommunications

基于Hadoop的海量数据处理框架



- 数据分析员 vs. 程序员
  - **数据抽象**
  - **操作方式**
  - **执行环境**

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

目录

Hadoop数据分析 - Hive

- 来源： Ashish Thusoo, Joydeep Sen Sarma, et al., **Facebook**, “Hive: A **Warehousing** Solution over A Map-Reduce Framework”, Proceedings of the VLDB Endowment, Aug. 2009.
- Why Hive？（http://hive.apache.org/）
  - Hive is a **data warehouse** system for Hadoop that facilitates easy data summarization, **ad-hoc queries**, and the analysis of large datasets stored in Hadoop compatible file systems. Hive provides a mechanism to **project structure** onto this data and query the data using a SQL-like language called **HiveQL**.
  - 数据库 vs. 数据仓库：存取（面向事务） vs. 分析（面向主题）

Hadoop数据分析 - 感受Hive

代码

```
#include "mapreduce/mapreduce.h"
class WordCounter : public Mapper {
 public:
  virtual void Map(const MapInput& input) {
   const string& text = input.value();
   const int n = text.size();
   for (int i = 0; i < n; ) {
   while ((i < n) && isspace(text[i]))    i++;
   int start = i;
   while ((i < n) && !isspace(text[i]))    i++;
   if (start < i)
    Emit(text.substr(start,i-start),"1"); }}};
REGISTER_MAPPER(WordCounter);
```

```
class Adder : public Reducer {
 virtual void Reduce(ReduceInput* input) {
  int64 value = 0;
  while (!input->done()) {
   value += StringToInt(input->value());
   input->NextValue();
  }
  Emit(IntToString(value)); }};
REGISTER_REDUCER(Adder);
```

```
int main(int argc, char** argv) {
  ParseCommandLineFlags(argc, argv);
  MapReduceSpecification spec;
  for (int i = 1; i < argc; i++) {
    MapReduceInput* input = spec.add_input();
    input->set_format("text");
    input->set_filepattern(argv[i]);
    input->set_mapper_class("WordCounter");
  }
  MapReduceOutput* out = spec.output();
  out->set_filebase("/gfs/test/freq");
  out->set_num_tasks(100);
  out->set_format("text");
  out->set_reducer_class("Adder");
  out->set_combiner_class("Adder");
  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  return 0;
}
```

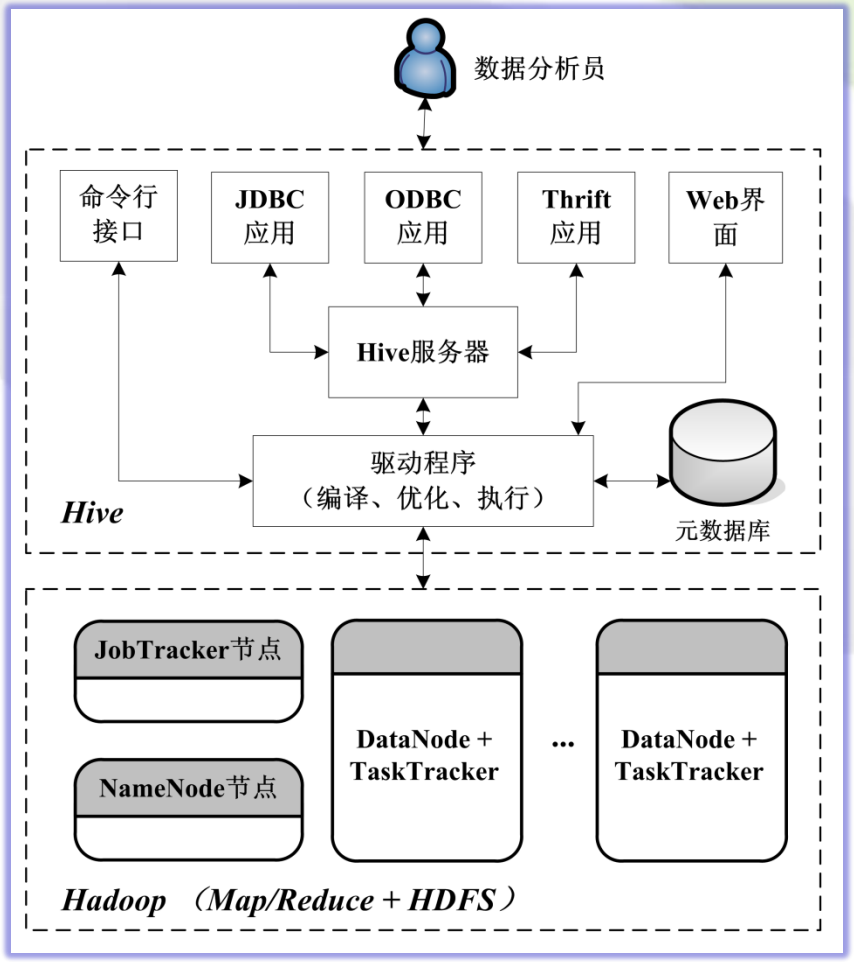
Hive

```
SELECT * FROM log WHERE date > '2012-12-01' ;
```

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构及实现
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

Hive架构与组件



- **基于Hadoop框架**
  - HDFS：文件存储
  - MapReduce：命令执行
- **用户操作接口**：
  - 命令行接口（shell）
  - Web界面
  - Hive应用
- **Hive服务器**：
  - Hive应用可以以JDBC、ODBC和Thirft接口访问指定地址和端口的Hive服务器。
- **驱动程序**：
  - 负责处理Hive语句，完成编译、优化和执行的工作
  - 生成相应的MapReduce任务与HDFS节点进行数据交互
- **元数据库**：
  - 存储Hive中与数据表相关的元数据，包括数据的库、表和分区组织等信息

Hive数据组织

- 在Hive中，数据以库（Database）、表（Table）、分区（Partition）和桶（Bucket）的层次进行组织
  - **库（Database）**：与传统的关系型数据库类似，库就是若干数据表的集合，代表了用户希望管理的一个数据集。
  - **表（Table）**：与关系型数据库中表的概念类似，表在逻辑上由存储的数据和描述表格中数据形式的相关元数据组成，每张表中的数据以序列化文件的形式存储在相应的HDFS目录下。
  - **分区（Partition）**：分区是为了在数据量过大时提高数据存储效率而对表进行划分的机制，每个表可以包含一个或多个分区，每个分区在物理上是HDFS存储表数据的目录下的子目录。
  - **桶（Bucket）：**通过对指定列值进行哈希并将结果除以桶的个数取余数的方法，将一张表或分区分到不同桶中，从而在查询少量数据只需在对应的桶中进行查找，提高查询效率。一个桶对应一个HDFS文件，存在于一个分区或一张表的目录中，文件按照字典顺序排列的。
- 分区及桶示例：

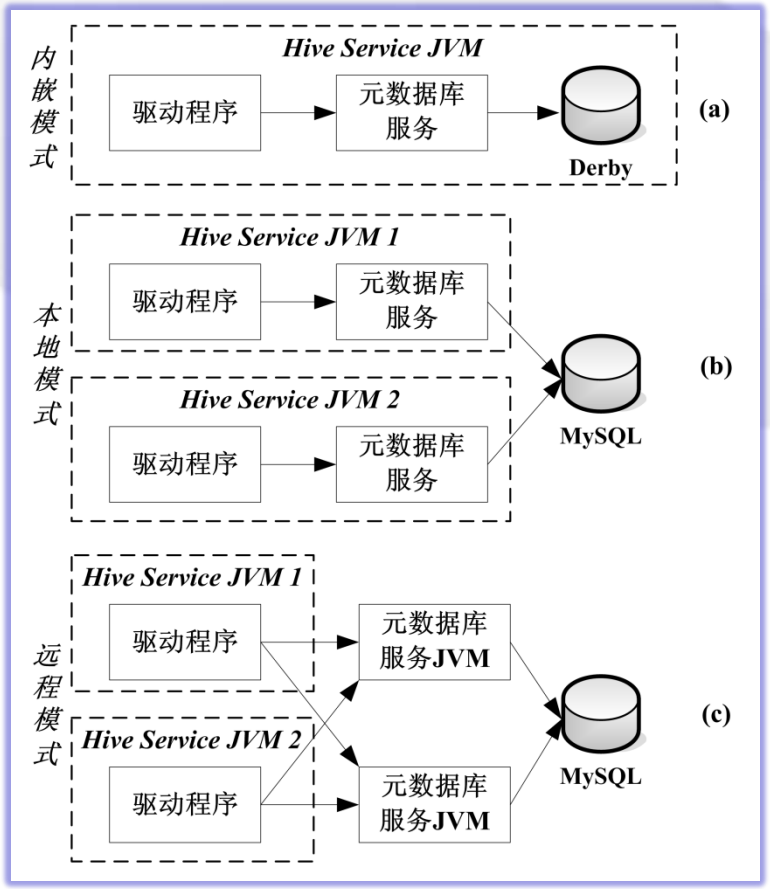
字段	**userID**	deviceID	**type**	roamType	url	time
类型	String	String	int	int	String	String

/user/hive/warehouse/logs/Type=1/file_0000
/user/hive/warehouse/logs/Type=1/file_0001
…
/user/hive/warehouse/logs/Type=2/file_0000

  - 分区后，Type列作为划分列在文件系统的目录中体现，因此此列不会出现在数据文件的列值中

Hive中的元数据

- 元数据是描述数据的库/表/分区/桶组织形式和关系的基础数据
  - 元数据集中存放在元数据库（Metastore）中
  - 采用了传统的关系数据库（MySQL或Derby）存储元数据
- 元数据的三种存储模式



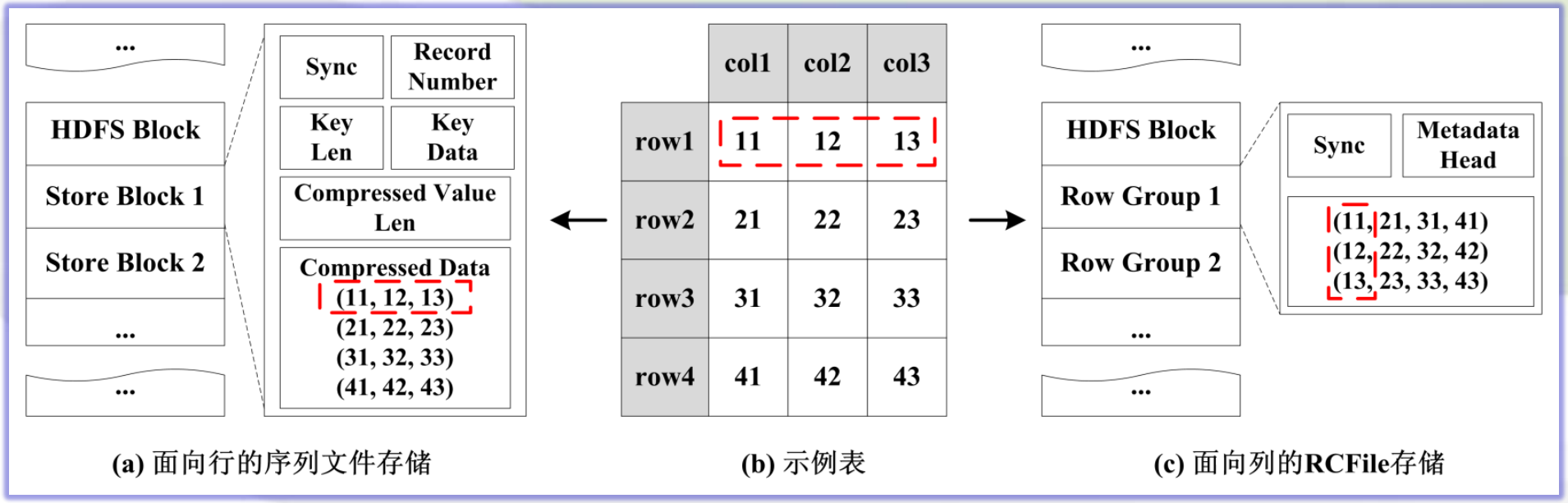
- **内嵌（Embedded）模式**：
  - Hive元数据库的默认运行模式
  - 元数据库服务和Hive服务运行在同一个JVM中
  - 元数据存储在内嵌在本地磁盘的Derby数据库中
  - 只能存在一个Hive会话，适合Hive的简单试用
- **本地（Local）模式**：
  - 使用独立的数据库作为元数据的存储组件，例如MySQL
  - 元数据存储独立，支持多个Hive服务共享一个元数据库
- **远程（Remote）模式**：
  - Hive服务和元数据库服务运行在不同的JVM中
  - Hive服务器可以访问多个元数据库服务，具有很好的可扩展性
  - 元数据库可部署于防火墙之后，通过JDBC或ODBC远程访问，具有更好的安全性

Hive数据存储 - 行格式

- Hive数据存储格式的两个维度：
  - 行格式（row format）：数据行以及包含的属性是按照怎样的格式存储到文件中
  - 文件存储格式（file format）：数据表中的数据是如何存储到文件中
- 行格式
  - SerDe（**Ser**ializer-**De**serializer）
    - 每行数据是在存储时要进行序列化操作（Serializer），即将每行数据的属性结构及属性值转换为二进制数据流存入文件中
    - 读取时要进行反序列化操作（Deserializer），即将文件中存储的二进制数据量还原为每行数据的属性结构和属性值
  - Hive提供了多种序列化反序列化接口，开发者也可以扩展自己的SerDe接口设置数据存储格式
  - 默认使用的SerDe接口为：延迟简单SerDe（LazySimpleSerDe）
    - 以回车符（ASCII码13）区分不同的行
    - 以CTRL-A（ASCII码1）区分一行中的不同列
    - 延迟（Lazy）指只有当某列数据被访问时才会进行反序列化操作，以提高数据存储效率

Hive数据存储 - 文件格式

- 文件存储格式（file format）：数据表中的数据是如何存储到文件中



(a) 面向行的序列文件存储　　(b) 示例表　　(c) 面向列的RCFile存储

文本文件存储

- 带分隔符的文本文件
- 行以回车符进行分割，列用CTRL-A分割
- 更多分隔符支持复杂数据类型的存储，例如使用CTRL-B对集合（Collection）中的元素进行分割
- 优点：结构简单，适合MapReduce程序
- 缺点：占用空间大，IO效率低

面向行的序列文件（图a）

- 按顺序存放的KV对
- 支持可分割并行的压缩
- 数据面向行存储
- 优点：同一行记录的位于同一HDFS节点上，适应快速数据加载和动态负载均衡
- 缺点：不支持部分列的快速查询，磁盘空间利用率低

面向列的RCFile文件（图c）

- “先水平切分，再垂直切分”
- 若干行组合为行组（Row Group），每个行组存放于一个HDFS Block中，同一行的数据存储在同一节点上
- 不同行的同一列数据顺序存放，然后再存储下一列数据
- 每个行组在存储时包含三个部分：同步标识，元数据头，存储数据的数据段
- 结合了行存储和列存储的各自优点

HQL - Hive Query Language

- 提供给数据分析人员使用的类似SQL的命令语法
  - 适应海量数据分析实际应用环境的需求
  - 为使用者提供一个与传统SQL使用习惯相近的分析语言
- 不是SQL语言所遵循的SQL-92标准中的全集：非全集，有扩展

对比点	SQL	HiveQL
更新方式	UPDATE,INSERT,DELETE	INSERT OVERWRITE TABLE （整表填充）
事务支持	支持	不支持
索引	支持	不支持
处理时延	秒级	分钟级
数据类型	整数、浮点型、定点数、文本和二进制串、时间	整数、浮点数、布尔型、字符串、数组、映射、结构
内置函数	多（数百个）	少（十几个）
多表插入	不支持	支持
查询语句	满足SQL-92标准	FROM条件中只支持单表或单视图， SORT BY只支持部分排序方式， LIMIT只可限制返回行的数量，不支持HAVING
连接查询	满足SQL-92标准	内连接、外连接、半连接和带提示的SQL-92语法
子查询	完全支持	只能在FROM条件中，不支持相关子查询
视图	视图可更新，可以实体化化也可以非实体化	视图只读，不支持实体化
扩展函数	支持用户定义函数和存储过程	支持用户定义函数和MapReduce脚本

HQL中的数据类型

分类	数据类型	描述	示例
基本数据类型	TINYINT	有符号整数，1个字节，范围从-128到127	1
	SMALLINT	有符号整数，2个字节，范围从-32768到32767	1
	INT	有符号整数，4个字节，范围从-2147483648到2147483647	1
	BIGINT	有符号整数，8个字节，范围从-92233720368547758O8到9223372036854775807	1
	FLOAT	单精度浮点数，4个字节	1.0
	DOUBLE	双精度浮点数，8个字节	1.0
	BOOLEAN	true/false	true
	STRING	字符串	'b', "a"
复杂数据类型	ARRAY	一组有序字段。字段的类型必须相同	array(1,2)
	MAP	一组无序的键/值对。键的类型必须是基本类型；值可以是任意类型。同一个映射的键的类型必须相同，值的类型也必须相同	map(1, "a")
	STRUCT	一组命名的字段。字段的类型可以不同	struct(1, "a", 2.0)

- STRUCT支持将多个不同类型的字段封装为一个结构型数据

  STRUCT<index:INT，name:STRING，weight:DOUBLE>

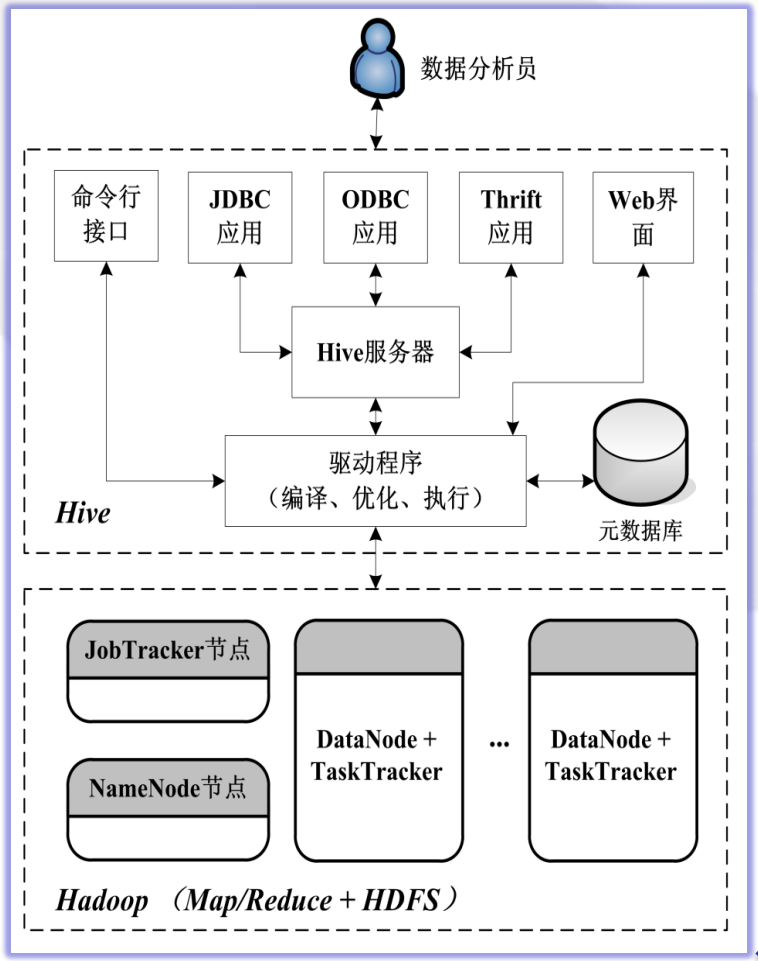
- 复杂数据类型在定义时要使用尖括号指明其数据字段的数据类型，并且允许任意层次的嵌套关系

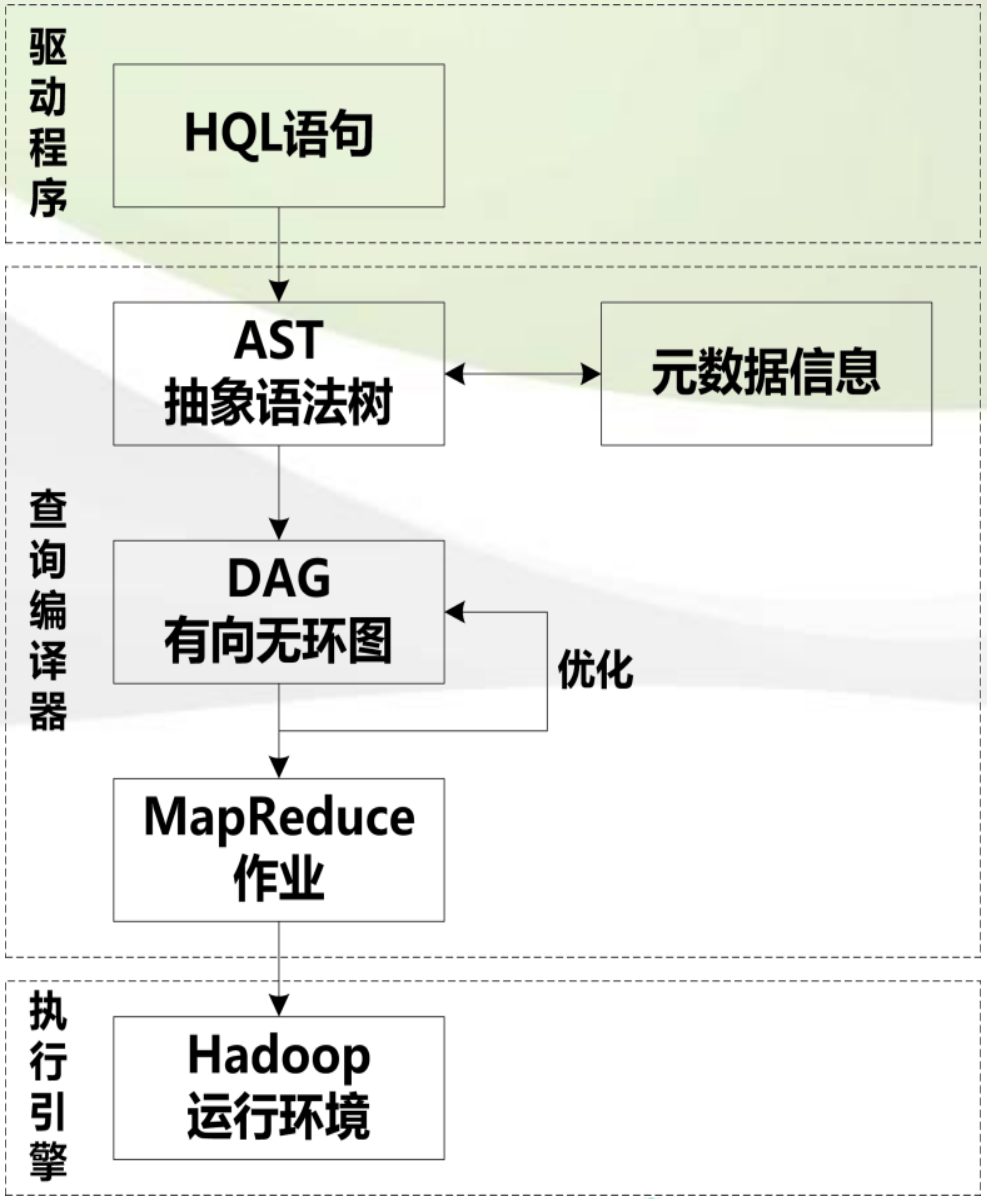
  col1 ARRAY<INT>

  col2 MAP<INT, STRING>

  col3 STRUCT<index:INT, name:STRING, weight:DOUBLE>

HQL执行流程


数据分析员
命令行接口
JDBC应用
ODBC应用
Thrift应用
Web界面
Hive服务器
驱动程序（编译、优化、执行）
元数据库
Hive
JobTracker节点
NameNode节点
DataNode + TaskTracker
...
DataNode + TaskTracker
Hadoop （Map/Reduce + HDFS）


驱动程序
HQL语句
查询编译器
AST 抽象语法树
元数据信息
DAG 有向无环图
优化
MapReduce 作业
执行引擎
Hadoop 运行环境

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构及实现
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

HQL实例（0） - 源数据

- 源数据
  - 源数据存放在HDFS的/data/目录下
  - 用户信息存放于user.txt文件

```
1111，25，male
2222，30，male
3333，30，female
...
```

  - 访问日志存放于log.txt文件

```
20130601，3g.qq.com，1111，portal，10
20130602，cnn.com，2222，news，20
20130602，baidu.com，3333，search，15
20130603，news.qq.com，1111，news，100
20130603，baidu.com，3333，search，15
...
```

HQL实例（1） - 创建数据表

- user.txt → user

user		
userID	**age**	**gender**
STRING	*INT*	*STRING*
1111	25	male
2222	30	male
3333	30	female

- log.txt → log

log				
date	**URL**	**userID**	**category**	**traffic**
STRING	*STRING*	*STRING*	*STRING*	*INT*
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	social	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

HQL实例（1）- 创建数据表

- Hive中的两种数据表：内部数据表、外部数据表
  - 内部数据表：Hive将把数据文件移动到它管理的数据仓库目录下
  - 外部数据表：Hive只创建并管理外部数据表的元数据，数据文件则仍使用原始文件
  - 区别
    - 创建外部表时需要指定外部数据源文件所在的位置
    - 加载数据（LOAD）和删除表（DROP）两类操作

- user表

```
CREATE TABLE user (userID STRING, age INT, gender STRING);
```

- log表

```
CREATE EXTERNAL TABLE log (date STRING, URL STRING, userID, STRING, category STRING,
    traffic INT) LOCATION '/data/log.txt';
```

HQL实例（1）- 创建数据表（cont.）

- 分区：将数据表按照某个列进行切分然后存放在不同的目录下，提高对相应列的查询效率
  - 例如：将日志文件中的大量记录按照日期进行分区存放，提高日期查询效率

```
CREATE TABLE log (date STRING, URL STRING, userID, STRING, category STRING,
    traffic INT) PARTITIONED BY (date STRING);
LOAD DATA INPATH '/data/log_20130601.txt' INTO table log PARTITIONED (date='20130601');
```

  - 注意：date没有在列描述区域出现，仅在指定分区的语句段出现
  - 查看分区命令：SHOW PARTITIONS

```
hive> SHOW PARTITIONS log;
        date=20130601
        date=20130602
```

- 桶：支持快速生成抽样样本集及提高查询效率
  - 例如：使用用户ID作为划分桶的字段，并生成5个桶

```
CREATE TABLE user (userID STRING, age INT, gender STRING)
    CLUSTERED BY (userID) INT 5 BUCKETS;
```

  - 在数据存储时将对用户ID进行哈希并用结果除以5后获得的余数决定分配到哪个桶中
  - 桶就是分区目录下的文件

HQL实例（2） - 加载数据

- 加载数据：从本地文件系统或HDFS中导入数据到数据表中
- 命令：LOAD DATA

- 内部数据表
  - 加载数据相当于一个移动操作，即将源数据文件移动到Hive管理的文件目录下
  - 对于user表，原始数据文件hdfs://data/user.txt将被移动到Hive管理的数据仓库hdfs://user/hive/warehouse/user目录下

    LOAD DATA INPATH '/data/user.txt' INTO table user;

- 外部数据表
  - 加载操作仅仅是建立元数据
  - 不会检查加载命令中的原始数据文件是否存在，留在之后进行数据操作时才进行

    LOAD DATA INPATH '/data/log.txt' INTO table log;

HQL实例（3）- 修改数据表

- 可修改表名、列名、列字段类型、增加列、替换列
- 命令：ALTER TABLE
- 将原来名为user的表重命名为new_user

  ALTER TABLE user RENAME TO new_user;

  - 内部数据表：
    - 数据文件所在的目录将被重命名为新的表名
    - 例如将hdfs://user/hive/warehouse/user目录被重命名为hdfs://user/hive/warehouse/new_user
  - 外部数据表：
    - 只修改元数据，不会操作数据文件和目录
- 增加列

  ALTER TABLE user ADD COLUMNS (addres STRING);
- 修改列名和列字段类型

  ALTER TABLE user CHANGE addres address STRING;
- 删除列

  ALTER TABLE user REPLACE COLUMNS (userID STRING, age INT, gender STRING);
- 注意：HiveQL修改表结构后，仅修改元数据中的表定义信息，并不真正更新数据文件中的内容，因此需要其他的一些机制来更新数据文件

HQL实例（4） - 复制数据表

- 将一个已有表中的数据复制到另一个表中：单表复制、多表复制和创建表时复制
- 单表复制：使用log表复制出一个同样的clong_log表

```
CREATE TABLE clone_log (date STRING, URL STRING, userID, STRING, category STRING, traffic INT);
INSERT OVERWRITE TABLE clone_log SELECT date, URL, userID, category, traffic FROM log;
```

- 多表复制：利用log表产生两个统计表，分别存放每日记录数统计和每日网站访问次数统计

```
CREATE TABLE access_count (date STRING, count INT); // 每日记录数统计表
CREATE TABLE url_access_count (date STRING, count INT); // 每日网站访问统计表
FROM log
    INSERT OVERWRITE TABLE access_count SELECT date, COUNT(1) GROUP BY date
    INSERT OVERWRITE TABLE url_access_count SELECT date, COUNT(DISTINCT URL) GROUP BY date;
```

- 创建表时复制

```
CREATE TABLE clone_log AS SELECT date, URL, userID, category, traffic FROM log;
```

HQL实例（5）- 删除数据表

- 删除数据表命令：DROP TABLE
- 例如删除clone_log_1表

  DROP TABLE clone_log_1;

- 区别：
  - 内部数据表：删除表操作是一个真正的删除操作，会将表对应的元数据和数据文件一并删除
  - 外部数据表：删除操作仅是删除元数据，真正的数据文件不会删除

HQL实例（6） - 数据查询

- 数据查询命令：SELECT … FROM …

- 支持与SQL类似的子句
  - 条件（WHERE）
  - 分组（GROUP）
  - 排序（ORDER BY）
  - 限制返回数量（LIMIT）等

```
SELECT * FROM log WHERE date > '20130601';
```

HQL实例（7）- 数据查询（Group By）

- 查询每类网页的流量和

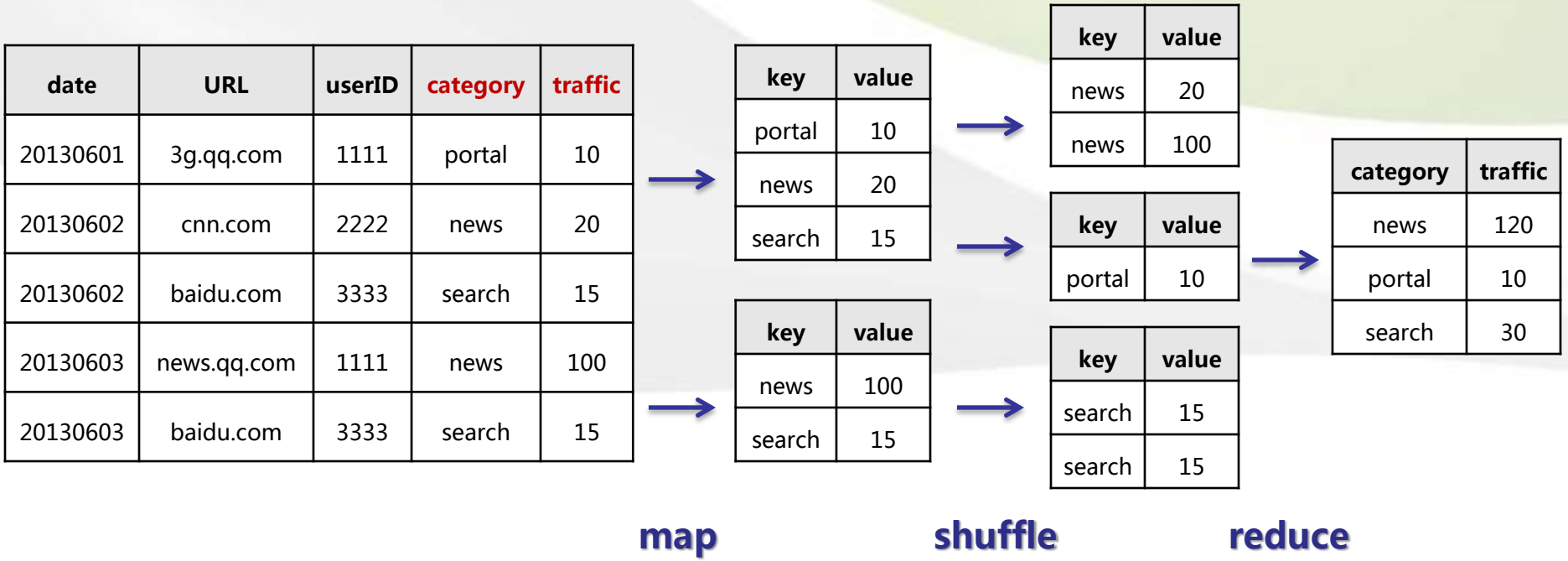
```
INSERT INTO TABLE sum_traffic
    SELECT category, SUM(traffic) FROM log GROUP BY category;
```

log				
date	**URL**	**userID**	**category**	**traffic**
STRING	*STRING*	*STRING*	*STRING*	*INT*
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	news	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

sum_traffic	
category	**traffic**
STRING	*INT*
news	120
portal	10
search	30

HQL Group By的实现

```
INSERT INTO TABLE sum_traffic
  SELECT category, SUM(traffic) FROM log GROUP BY category;
```



HQL实例（8） - 数据查询（Group By with Distinct）

- 查询每个网页的不同用户访问数

```
SELECT URL, COUNT(DISTINCT userID) FROM log GROUP BY URL;
```

log				
date	**URL**	**userID**	**category**	**traffic**
STRING	*STRING*	*STRING*	*STRING*	*INT*
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	news	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

result	
3g.qq.com	1
baidu.com	1
cnn.com	1
news.qq.com	1

HQL Group By with DISTINCT的实现

date	URL	userID	category	traffic
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	news	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

map

key	value
<3g.qq.com, 1111>	1
<cnn.com, 2222>	1
<baidu.com, 3333>	1

key	value
<news.qq.com, 1111>	1
<baidu.com, 3333>	1

shuffle

key	value
<3g.qq.com, 1111>	1

key	value
<baidu.com, 3333>	1
<baidu.com, 3333>	1

key	value
<cnn.com, 2222>	1

key	value
<news.qq.com, 1111>	1

reduce

URL	count
3g.qq.com	1
baidu.com	1
cnn.com	1
news.qq.com	1

SELECT URL, COUNT(DISTINCT userID) FROM log GROUP BY URL;

HQL实例（9） - 数据查询（Order By）

- ORDER BY与SQL相同

SELECT date, URL FROM log WHERE date >= '20130601' ORDER BY date ASC, URL ASC;

log				
date	**URL**	**userID**	**category**	**traffic**
STRING	*STRING*	*STRING*	*STRING*	*INT*
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	news	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

result	
date	**URL**
20130601	3g.qq.com
20130602	baidu.com
20130602	cnn.com
20130603	baidu.com
20130603	news.qq.com

- ORDER BY要实现全局排序，只能在一个reducer中执行，执行效率低

HQL实例（10） - 数据查询（更多排序方式）

- SORT BY、DISTRIBUTE BY和CLUSTER BY
  - SORT BY子句确保在查询语句执行的MapReduce程序中，同一reducer内的数据按照指定的列进行排序
  - DISTRIBUTE BY子句则可以指定具有相同指定列值的数据进入同一个reducer中进行处理
  - CLUSTER BY是在前面两个子句所指定的列相同时的简化表示

```
SELECT date, URL FROM log WHERE date >= '20130601' DISTRIBUTE BY URL SORT BY date ASC;
```

log				
date	**URL**	**userID**	**category**	**traffic**
STRING	*STRING*	*STRING*	*STRING*	*INT*
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	news	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

result	
date	**URL**
20130601	3g.qq.com
20130602	baidu.com
20130603	baidu.com
20130602	cnn.com
20130603	news.qq.com

HQL实例（11） - 数据查询（Join）

- 连接查询（Join）是将两个表中在共同数据项上相互匹配的那些行合并后进行查询操作
  - 内连接、左外连接、右外连接、全外连接
- 将从user表和log表中按照userID进行匹配查询

```
SELECT log.date, log.URL, user.userID, user.age FROM user JOIN log ON (user.userID = log.userID);
```

user		
userID	**age**	**gender**
STRING	*INT*	*STRING*
1111	25	male
2222	30	male
3333	30	female

log				
date	**URL**	**userID**	**category**	**traffic**
STRING	*STRING*	*STRING*	*STRING*	*INT*
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	social	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

result			
date	**URL**	**userID**	**age**
20130601	3g.qq.com	1111	25
20130602	cnn.com	2222	30
20130602	baidu.com	3333	30
20130603	news.qq.com	1111	25
20130603	baidu.com	3333	30

HQL Join的实现

user		
userID	age	gender
1111	25	male
2222	30	male
3333	30	female

log				
date	URL	userID	category	traffic
20130601	3g.qq.com	1111	portal	10
20130602	cnn.com	2222	social	20
20130602	baidu.com	3333	search	15
20130603	news.qq.com	1111	news	100
20130603	baidu.com	3333	search	15

result			
date	URL	userID	age
20130601	3g.qq.com	1111	25
20130602	cnn.com	2222	30
20130602	baidu.com	3333	30
20130603	news.qq.com	1111	25
20130603	baidu.com	3333	30

map

reduce

key	value
1111	<1, 25>
2222	<1, 30>
3333	<1, 30>

key	value
1111	<2, 20130601, 3g.qq.com>
2222	<2, 20130602, cnn.com>
3333	<2, 20130602, baidu.com>
1111	<2, 20130603, news.qq.com>
3333	<2, 20130603, baidu.com>

shuffle

key	value
1111	<1, 25>
1111	<2, 20130601, 3g.qq.com>
1111	<2, 20130603, news.qq.com>
2222	<1, 30>
2222	<2, 20130602, cnn.com>
3333	<1, 30>
3333	<2, 20130602, baidu.com>
3333	<2, 20130603, baidu.com>

HQL实例（10） - 视图（View）

- 视图（View）
  - 数据分析中的高级功能，是指由查询语句产生的一个虚拟表
  - 与真实的数据表类似，视图也是由一系列由若干列组成的行数据构成

- 与SQL的差异
  - HiveQL不支持视图的实体化，即只会在执行引用视图的HiveQL语句时才会执行
  - 如果需要视图进行大规模的数据变换，或某个视图的引用会频繁进行，最好采用创建新表的方式而不是采用视图

- 视图示例：
  - 产生一个2012年11月的访问记录集

    CREATE VIEW log_view AS SELECT * FROM log WHERE date >= '20130601' AND date <= '20130603';
  - 以上操作仅仅是将查询条件存储在metastore中，只有当引用视图的语句执行时查询操作才真正执行

    SELECT * FROM log_view GROUP BY userID;

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

HQL内建函数

- 内建函数
  - count()
  - sum()
  - avg()
  - max()
  - min()
  - …

```
SELECT category, SUM(traffic) FROM log GROUP BY category;
SELECT URL, COUNT(DISTINCT userID) FROM log GROUP BY URL;
```

HQL自定义函数（1） - 使用方法

- 内建函数不能满足一些特性需求，因此提供了用户自定义函数扩展功能
  - 普通自定义函数（UDF）
  - 聚集自定义函数（UDAF）
  - 表生成自定函数（UDTF）

- 自定义函数的使用步骤：
  ① 将编写好的程序进行打包，例如将myfunction类打包为myfunction.jar包，并将myfunction.jar包存放到/home/myjar目录下。
  ② 使用ADD JAR命令将myfunction.jar包注册到Hive里。命令为：

  ```
  ADD JAR /home/myjar/ myfunction.jar;
  ```

  ③ 使用CREATE TEMPORARY命令为自定义函数编定别名，命令为：

  ```
  CREATE TEMPORARY FUNCTION myfunction AS 'com.hadoopbook.hive.myfunction';
  ```

  ④ 在HQL查询语句中使用自定义函数，例如：

  ```
  SELECT myfunction(col) FROM table;
  ```

HQL自定义函数（2）- 普通自定义函数

- 普通自定义函数（User Defined Function，UDF）
  - 输入单行数据，处理后输出一行数据
  - 编写UDF函数的两个条件：
    - 继承org.apache.hadoop.hive.ql.exec.UDF类
    - 实现UDF类中的evaluate方法

- 示例：将表中的age列值加1后输出

```
1:  import org.apache.hadoop.hive.ql.exec.UDF;
2:  public class AddOne extends UDF{
3:      public int evaluate (int age) {
4:          int newAge = age + 1;
5:          return newAge;
6:      }
7:  }
```

- 使用：SELECT addone(age) FROM user;

HQL自定义函数（3）- 自定义聚合函数

- 自定义聚合函数（User-Defined Aggregate Function，UDAF）
  - 输入多行数据，处理后输出一行数据
  - 编写UDF函数的两个条件：
    - 继承 org.apache.hadoop.hive.ql.exec.UDAF类
    - 实现接口类及其5个方法 org.apache.hadoop.hive.ql.exec.UDAFEvaluator
- 例：查询user表中age列的最大值

```
1:  public class Maximum extends UDAF{
2:    public static class MaximumIntEvaluator
3:      implements UDAFEvaluator{
4:      private int result;
5:      public void init() {
6:        result = 0;
7:      }
8:      public boolean iterate(int value) {
9:        result = Math.max(result, value);
10:       return true;
11:     }
12:     public int terminatePartial() {
13:       return result;
14:     }
15:     public Boolean merge() {
16:       return true;
17:     }
18:     public int terminate{
19:       return result;
20:     }
21:   }
22: }
```

HQL自定义函数（4） - 表生成自定义函数

- 表生成自定义函数（User Defined Table-generating Function，UDTF）
  - 输入单行数据，处理后输出多行数据
  - 编写UDF函数的两个条件：
    - 继承类org.apache.Hadoop.hive.ql.udf.generic.GenericUDTF
    - 实现initialize(), process(), close()三个方法
- 示例：查询表中URL列的值，并将URL值以点号进行分割，分割后的每一个值作为一整行输出到结果表中

```
1:  public class StringUDTF extends GenericUDTF{
2:    public void close() throws HiveException {
3:    }
4:    public StructObjectInspector
5:      initialize(ObjectInspector[] args) {
6:    }
7:    public void process(Object[] args) {
8:      String input = args[0].toString();
9:      String[] test = input.split(".");
10:     for (int i=0; i<test.length; i++) {
11:       String result = test[i];
12:       forward(result);
13:     }
14:   }
15: }
```

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

Hive技巧（1） - Group By

- Map端聚合（相当于Combiner）

  SELECT category, SUM(traffic) FROM log GROUP BY category;

  - hive.map.aggr = true 是否在 Map 端进行聚合，默认为 true
  - hive.groupby.mapaggr.checkinterval = 100000 在 Map 端进行聚合操作的条目数目

- 数据倾斜聚合优化
  - hive.groupby.skewindata = false
  - 当设为true，生成的查询计划会以两个MapReduce Job实现负载均衡
    - 第1个Job的Map将记录随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，使相同的 Group By Key分发到不同的 Reduce 中，达到负载均衡
    - 第2个Job 再根据上一个job的结果按照 Group By Key 分布到 Reduce 中，完成最终的聚合操作。

Hive技巧（2） - 合并小文件

- hive.merge.mapfiles：是否合并Map输出文件，默认为 true
- hive.merge.mapredfiles：是否合并Reduce输出文件，默认为 false
- hive.merge.size.per.task：合并文件的大小，默认为256*1000*1000

Hive技巧（3） - Join

- Map Join时，将条目少的表/子查询放在 Join 操作符的左边

  SELECT a.val, b.val FROM a JOIN b ON (a.key = b.key)

  a是小表(小于25M），b是大表

- Common Join时，将重复关联键少的表放在左边

- 修改where子句

  SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)

  WHERE a.date='20130601' AND b.date='20130601‘

  修改为：

  SELECT a.val, b.val FROM a LEFT OUTER JOIN b

  ON (a.key=b.key AND b.date='20130601' AND a.date='20130601')

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

Hadoop数据分析 - Pig

- 来源： Christopher Olston, Benjamin Reed, Utkarsh Srivastava, et al., **Yahoo!**, "Pig Latin: A Not-so-foreign **Language for Data Processing**", ACM SIGMOD, 2008.
- Why Pig ？（http://pig.apache.org/）
  - Apache Pig is a platform for analyzing large data sets that consists of **a high-level language** for expressing **data analysis programs**, coupled with infrastructure for evaluating these programs.
  - Pig's language layer currently consists of a textual language called **Pig Latin**.

↔

Hadoop数据分析 - Pig及与Hive的对比

Hive

```
SELECT category, AVG(traffic) FROM log
    WHERE traffic > 10 GROUP BY category;
```

Pig

```
> urls = FILTER log BY traffic > 10;
> groups = GROUP urls BY category;
> output = FOREACH groups GENERATE category, AVG(urls.traffic);
```

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

Pig Latin语言

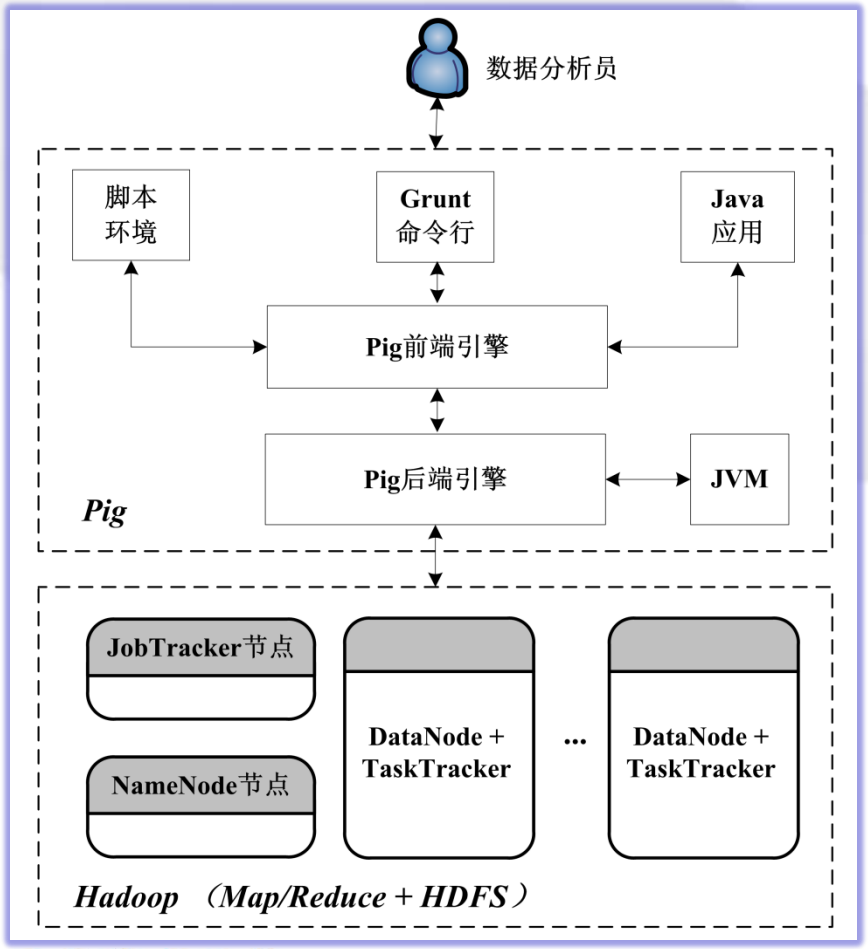
- Pig Latin语言是使用Pig进行数据分析的核心
  - 一种脚本语言
  - 基于Pig Latin编写的程序由一系列的数据操作（operation）或数据变换（transformation）组成
  - 指令由Pig执行环境进行解释和翻译，转换成一系列MapReduce作业执行
  - 关键点：
    - 语法
    - 数据类型
    - 运算符
    - 内置函数
    - 自定义函数

Pig架构与组件

- 客户端应用，可独立于集群
  - pig.properties文件中的参数

    fs.default.name=hdfs://hdfs_host/

    mapred.job.tracker=hadoop_host:8021



- **三种运行方式**
  - Grunt命令行
  - 脚本文件
  - 应用程序
- **前端引擎**：
  - 负责对Pig Latin语句进行语法检测、语句解析、逻辑检测和代码优化等任务
  - 将命令语句转化为对应的逻辑计划（Logical Plan），消除使用三种接口输入命令语句所造成的表达差异。
- **后端引擎**：
  - 以前端引擎输出的逻辑计划为输入，将其转换为特定运行环境可运行的代码
    - 本地JVM运行环境：这种方式下所有输入文件和命令执行都在节点本地执行
    - Hadoop集群运行环境：后端引擎将逻辑计划转换为一系列MapReduce作业，然后在Hadoop集群中运行

目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

Pig Latin语法与执行

- 语法：

```
/*这是注释*/
grouped_records = GROUP records BY year; --这也是注释
```

  - 脚本程序由一系列以上类似的语句构成，每个语句为一个操作
  - 每个语句可以分号结束，但非强制性（Hadoop文件系统命令，或环境诊断语句等例外）
  - 一些特殊含义的单词组成关键词列表，在列表中的单词不能在代码中作为标识符使用
    - 操作命令（LOAD，ILLUSTRATE等）
    - 系统指令（cat，ls等）
    - 表达式关键字（matches，FLATTEN等）
    - 内置函数（DIFF，MAX等）
  - 大小写敏感使用了混合规则，既保证了代码的严谨性，也保留了指令输入的灵活性
    - 系统内置的操作和命令指令输入时是大小写无关的
    - 代码中的标识符和函数名是大小写敏感的

- 执行

① 执行环境收到输入的程序后，对每个命令按次序进行解析，并检验语法，如发现错误则终止

② 解释器为每个关系操作建立逻辑计划，并加入到已解析完成的程序的逻辑计划上

③ 重复①②不断处理下一条语句，直到全部语句完成

④ 当整个程序逻辑计划全部完成后，按照输入的全部命令对数据进行操作

Pig Latin操作命令（1）- 关系操作

- Pig中的所有任务执行都是通过操作命令体现的，分为五类：
  - 关系操作、诊断操作、UDF操作、系统操作
- 关系操作（Relational Operator）
  - 读写数据和分析数据的操作集合
  - 每一个关系操作语句产生的结果称为“关系”（relation），可以类比为数据库中的一张表

类别	类型	操作	说明
关系操作	加载与存储	load	从文件系统或其他存储设备中加载数据存入关系
		store	将关系数据存入文件系统或其他存储设备中
		dump	将关系操作输出到控制台
	过滤	filter	从关系中过滤掉不需要的行
		distinct	从关系中删除重复的行
		foreach...generate	在关系中增加或删除字段
		stream	使用外部程序对关系进行转换
		sample	从关系中随机抽取数据样本
	分组与连接	join	将多个关系进行连接
		cogroup	在多个关系中进行数据分组
		group	在一个关系中进行数据分组
		cross	对多个关系进行交叉乘积运算
	排序	order	根据一个或多个字段对关系进行排序
		limit	限制关系的输出元组个数
	合并与分割	union	合并多个关系
		split	把一个关系分割为多个关系

Pig Latin操作命令（2）- 诊断操作和UDF操作

- 诊断操作
  - 用于对运行环境的执行情况进行显示

类别	类型	操作	说明
诊断操作	描绘关系	describe	输出关系的模版（schema）
	执行计划	explain	输出逻辑和物理执行计划
		illustrate	使用生成的输入数据子集显示逻辑计划的示例结果

- UDF操作
  - 用于支持用户自定义函数的实现

类别	类型	操作	说明
UDF操作	注册	register	在运行环境中注册一个JAR文件
	定义	define	为宏、自定义函数、streaming脚本或命令定义别名
	导入	import	从单独的文件中导入宏到脚本中

Pig Latin操作命令（3）- 系统操作

- 系统操作：
  - 文件系统操作：用于对Hadoop文件系统的文件和目录进行管理
  - 作业系统操作：用于控制脚本程序转换后产生的MapReduce作业

类别	类型	操作	说明
系统操作	文件系统	cat	输出一个或多个文件的内容
		cd	进入某个目录
		copyFromLocal	将本地文件或目录复制到Hadoop文件系统
		copyToLocal	将文件或目录从Hadoop文件系统复制到本地
		cp	把一个文件或目录复制到另一个目录下
		fs	进入Hadoop文件系统的命令交互环境
		ls	输出文件列表信息
		mkdir	创建目录
		mv	移动文件或目录
		pwd	输出当前所在目录的路径
		rm	删除文件或目录
		rmf	强行删除文件或目录
	作业系统	kill	终止某个MapReduce作业
		exec	在一个新的Grunt Shell中以批处理模式执行脚本
		help	显示可用的命令和选项的帮助信息
		quit	退出交互环境
		run	在当前Grunt Shell中执行脚本
		set	设置Pig选项和MapReduce作业属性
		sh	在Grunt环境中执行shell指令

Pig Latin数据类型

类别	类型	数据类型	描述	示例
简单类型	数值	int	32位有符号整数	1
		long	64位有符号整数	1L
		float	32位浮点数	0.1f
		double	64位浮点数	0.1
	文本	chararray	UTF-16格式的字符数组	'abc'
	字节数组	bytearray	二进制字节数组	
复杂类型	元组	tuple	任何类型字段序列构成的元组	(10, 'abc')
	包	bag	元组构成的集合	{(10, 'abc'), (2)}
	映射	map	键/值对，要求键必须为字符数组	['abc' 10]

Pig Latin表达式

类型	操作	表达式	说明	示例
取值	常量	文字	文字表述的常量	1.0, 'abc'
	字段值	$n	第n个字段的值（位置从0开始）	$1
		f	使用字段名取值	year
	投射	c.$n, c.f	在容器c（关系、包或元组）中的字段值	log.$0, log.year
	Map查找	m#k	映射m中键k对应的值	map#'abc'
	类型转换	(t)f	将字段f转换为类型t的值	(int)year
数学	加/减	x+y, x-y	加法和减法	$1+$2, $1-$2
	乘/除	x*y, x/y	乘法和除法	$1*$2, $1/$2
	取模	x%y	x除以y后的余数	$1%$2
	正/负	+x, -x	正和负值	+1, -1
条件判断	条件取值	x?y:z	如x为真，则取y，否则取z	type==1?0:1
	相等/不等	x==y, x!=y	x等于y或不等于y	year==2012, year!=2012
	大于/小于	x>y, x<y	x大于y或小于y	year>2011, year<2012
	大于等于/小于等于	x>=y, x<=y	x大于等于y或小于等于y	year>=2011, year<=2012
	正则匹配	x matches y	x与y之间的正则表达式匹配	year matches '[012]'
	空值	x is null	x是否是空值	year is null
	非空值	x is not null	x是否是非空值	year is not null
布尔操作	与	x and y	x和y的与操作	year==2012 and type==1
	或	x or y	x和y的或操作	year==2012 or type==1
	非	not x	x的非操作	not year==2012
函数	函数	fn(f1, f2, ...)	在一个或多个字段上应用函数fn	isValid(type)
平坦	平坦化	FLATTEN(f)	从包或元组中去除嵌套	FLATTEN(group)

Pig实例（1） - 源数据

- /data/log.txt

IP	Host	SP	Traffic
125.39.127.20	b8.photo.store.qq.com	qq	10
125.39.127.21	b5.photo.store.qq.com	qq	15
125.39.127.30	b25.photo.store.qq.com	qq	25
125.39.127.21	b40.photo.store.qq.com	qq	15
125.39.127.30	b32.photo.store.qq.com	qq	10
125.39.127.30	s6.photo.store.qq.com	qq	15
122.228.243.250	q.i02.wimg.taobao.com	taobao	100

Pig实例（2） - 加载（LOAD）

```
grunt> records = LOAD '/data/log.txt' AS (ip:chararray, host:chararray, sp:chararray,
traffic:int);

grunt> describe records;
records:{ip: chararray,host: chararray,sp: chararray, traffic: int}

grunt> dump records;
```

```
(125.39.127.20,b8.photo.store.qq.com,qq,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15)
(125.39.127.30,b25.photo.store.qq.com,qq,25)
(125.39.127.21,b40.photo.store.qq.com,qq,15)
(125.39.127.30,b32.photo.store.qq.com,qq,10)
(125.39.127.30,s6.photo.store.qq.com,qq,15)
(122.228.243.250,q.i02.wimg.taobao.com,taobao,100)
```

records

Pig实例（3） - 过滤（FILTER）

```
grunt> filter_records = FILTER records BY sp MATCHES '.*qq.*';

grunt> describe filter_records;
filter_records: {ip: chararray,host: chararray,sp: chararray, traffic: int}

grunt> dump filter_records;
```

records
(125.39.127.20,b8.photo.store.qq.com,qq,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15)
(125.39.127.30,b25.photo.store.qq.com,qq,25)
(125.39.127.21,b40.photo.store.qq.com,qq,15)
(125.39.127.30,b32.photo.store.qq.com,qq,10)
(125.39.127.30,s6.photo.store.qq.com,qq,15)
(122.228.243.250,q.i02.wimg.taobao.com,taobao,100)

filter_records
(125.39.127.20,b8.photo.store.qq.com,qq,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15)
(125.39.127.30,b25.photo.store.qq.com,qq,25)
(125.39.127.21,b40.photo.store.qq.com,qq,15)
(125.39.127.30,b32.photo.store.qq.com,qq,10)
(125.39.127.30,s6.photo.store.qq.com,qq,15)

Pig实例（4） - 排序（ORDER）

```
grunt> order_records = ORDER filter_records BY ip;

grunt> describe order_records;
order_records: {ip: chararray,host: chararray,sp: chararray, traffic: int}

grunt> dump order_records;
```

filter_records
(125.39.127.20,b8.photo.store.qq.com,qq,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15)
(125.39.127.30,b25.photo.store.qq.com,qq,25)
(125.39.127.21,b40.photo.store.qq.com,qq,15)
(125.39.127.30,b32.photo.store.qq.com,qq,10)
(125.39.127.30,s6.photo.store.qq.com,qq,15)

order_records
(125.39.127.20,b8.photo.store.qq.com,qq,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15)
(125.39.127.21,b40.photo.store.qq.com,qq,15)
(125.39.127.30,b25.photo.store.qq.com,qq,25)
(125.39.127.30,b32.photo.store.qq.com,qq,10)
(125.39.127.30,s6.photo.store.qq.com,qq,15)

Pig实例（5） - 分组（GROUP）

```
grunt> group_records = GROUP order_records BY ip;

grunt> describe group_records;
group_records: {group: chararray, filter_records: {(ip: chararray, host: chararray,
sp: chararray, traffic: int)}}

grunt> dump group_records;
```

```
(125.39.127.20,b8.photo.store.qq.com,qq,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15)
(125.39.127.21,b40.photo.store.qq.com,qq,15)
(125.39.127.30,b25.photo.store.qq.com,qq,25)
(125.39.127.30,b32.photo.store.qq.com,qq,10)
(125.39.127.30,s6.photo.store.qq.com,qq,15)
```

```
(125.39.127.20,{(125.39.127.20,b8.photo.store.qq.com,qq,10)})
(125.39.127.21,{(125.39.127.21,b5.photo.store.qq.com,qq,15),
(125.39.127.21,b40.photo.store.qq.com,qq,15)})
(125.39.127.30,{(125.39.127.30,b25.photo.store.qq.com,qq,25
),(125.39.127.30,b32.photo.store.qq.com,qq,10),(125.39.127.3
0,s6.photo.store.qq.com,qq,15)})
```

order_records **group_records**

Pig实例（6） - 循环处理（FOREACH）

```
grunt> count_records = FOREACH group_records GENERATE group,
COUNT(order_records.ip) as count, SUM(order_records.traffic) as sumTraffic;

grunt> describe count_records ;
count_records: {group: chararray, count: long, sumTraffic: long}

grunt> dump count_records ;
```

(125.39.127.20,{(125.39.127.20,b8.photo.store.qq.com,qq,10)})
(125.39.127.21,{(125.39.127.21,b5.photo.store.qq.com,qq,15),
(125.39.127.21,b40.photo.store.qq.com,qq,15)})
(125.39.127.30,{(125.39.127.30,b25.photo.store.qq.com,qq,25
),(125.39.127.30,b32.photo.store.qq.com,qq,10),(125.39.127.3
0,s6.photo.store.qq.com,qq,15)})

group_records

(125.39.127.20,1,10)
(125.39.127.21,2,30)
(125.39.127.30,3,50)

count_records

Pig实例（7） - 联结（JOIN）

```
grunt> join_records = JOIN order_records BY ip, count_records BY group;
grunt> describe join_records;
join_records: {order_records::ip: chararray, order_records ::host: chararray,
order_records ::sp: chararray, order_records::traffic:int, count_records::group:chararray,
count_records::count: long, count_records::sumTraffic: long}
grunt> dump join_records;
```

```
(125.39.127.20,b8.photo.store.qq.com,qq,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15)
(125.39.127.21,b40.photo.store.qq.com,qq,15)
(125.39.127.30,b25.photo.store.qq.com,qq,25)
(125.39.127.30,b32.photo.store.qq.com,qq,10)
(125.39.127.30,s6.photo.store.qq.com,qq,15)
```

```
(125.39.127.20,1,10)
(125.39.127.21,2,30)
(125.39.127.30,3,50)
```

→

```
(125.39.127.20,b8.photo.store.qq.com,qq,10,125.39.127.20,1,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15,125.39.127.21,2,30)
(125.39.127.21,b40.photo.store.qq.com,qq,15,125.39.127.21,2,30)
(125.39.127.30,b25.photo.store.qq.com,qq,25,125.39.127.30,3,50)
(125.39.127.30,b32.photo.store.qq.com,qq,10,125.39.127.30,3,50)
(125.39.127.30,s6.photo.store.qq.com,qq,15,125.39.127.30,3,50)
```

join_records

Pig实例（8） - 结果

```
grunt> end_records = FOREACH join_records GENERATE order_records::ip,
count_records::count, order_records::host, order_records::sp, order_records::traffic,
(double)order_records::traffic/(double)count_records::sumTraffic as percent:double;

grunt> describe end_records;
end_records：{order_records ::ip: chararray,count_records::count: long,
order_records ::host: chararray, order_records ::sp: chararray, percent: double}

grunt> dump end_records;
```

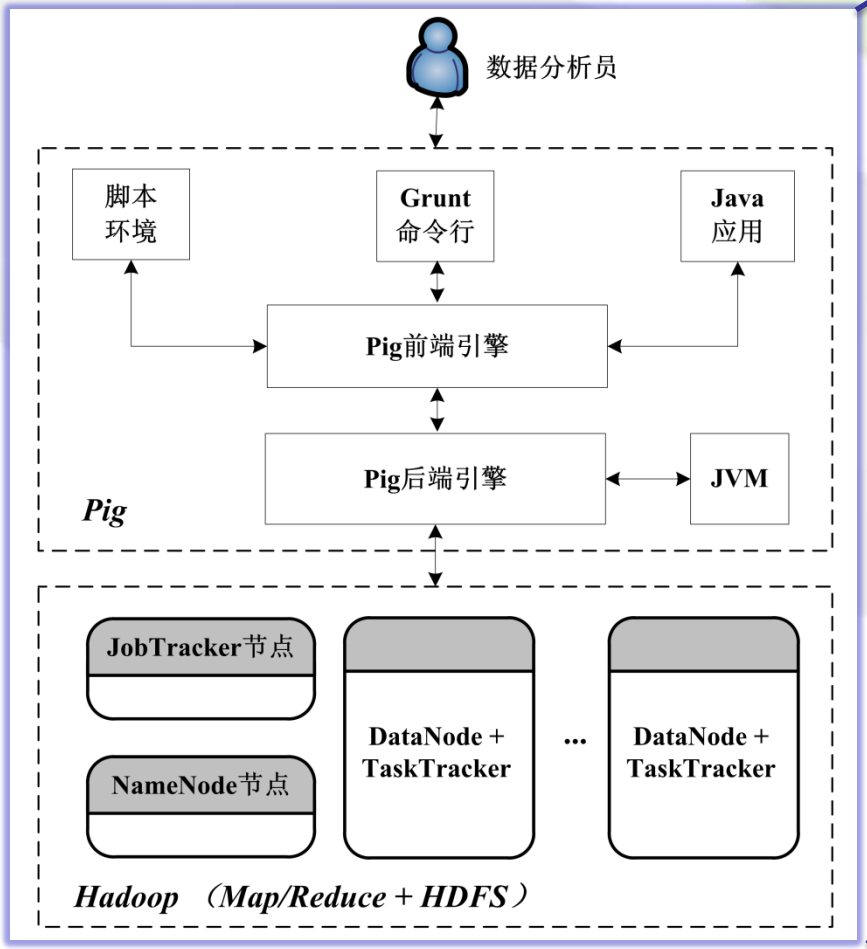
(125.39.127.20,b8.photo.store.qq.com,qq,10,125.39.127.20,1,10)
(125.39.127.21,b5.photo.store.qq.com,qq,15,125.39.127.21,2,30)
(125.39.127.21,b40.photo.store.qq.com,qq,15,125.39.127.21,2,30)
(125.39.127.30,b25.photo.store.qq.com,qq,25,125.39.127.30,3,50)
(125.39.127.30,b32.photo.store.qq.com,qq,10,125.39.127.30,3,50)
(125.39.127.30,s6.photo.store.qq.com,qq,15,125.39.127.30,3,50)

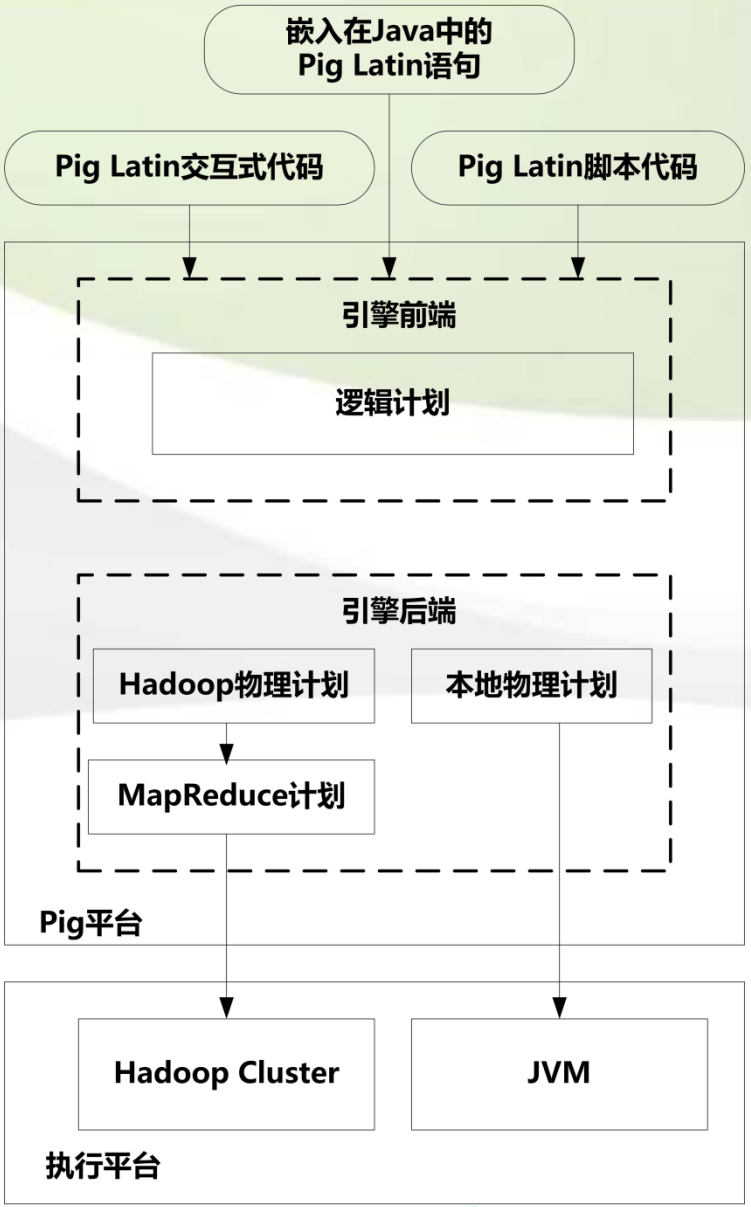
join_records

(125.39.127.20,1,b8.photo.store.qq.com,qq,10,1.0)
(125.39.127.21,2,b5.photo.store.qq.com,qq,15,0.5)
(125.39.127.21,2,b40.photo.store.qq.com,qq,15,0.5)
(125.39.127.30,3,b25.photo.store.qq.com,qq,25,0.5)
(125.39.127.30,3,b32.photo.store.qq.com,qq,10,0.2)
(125.39.127.30,3,s6.photo.store.qq.com,qq,15,0.3)

end_records

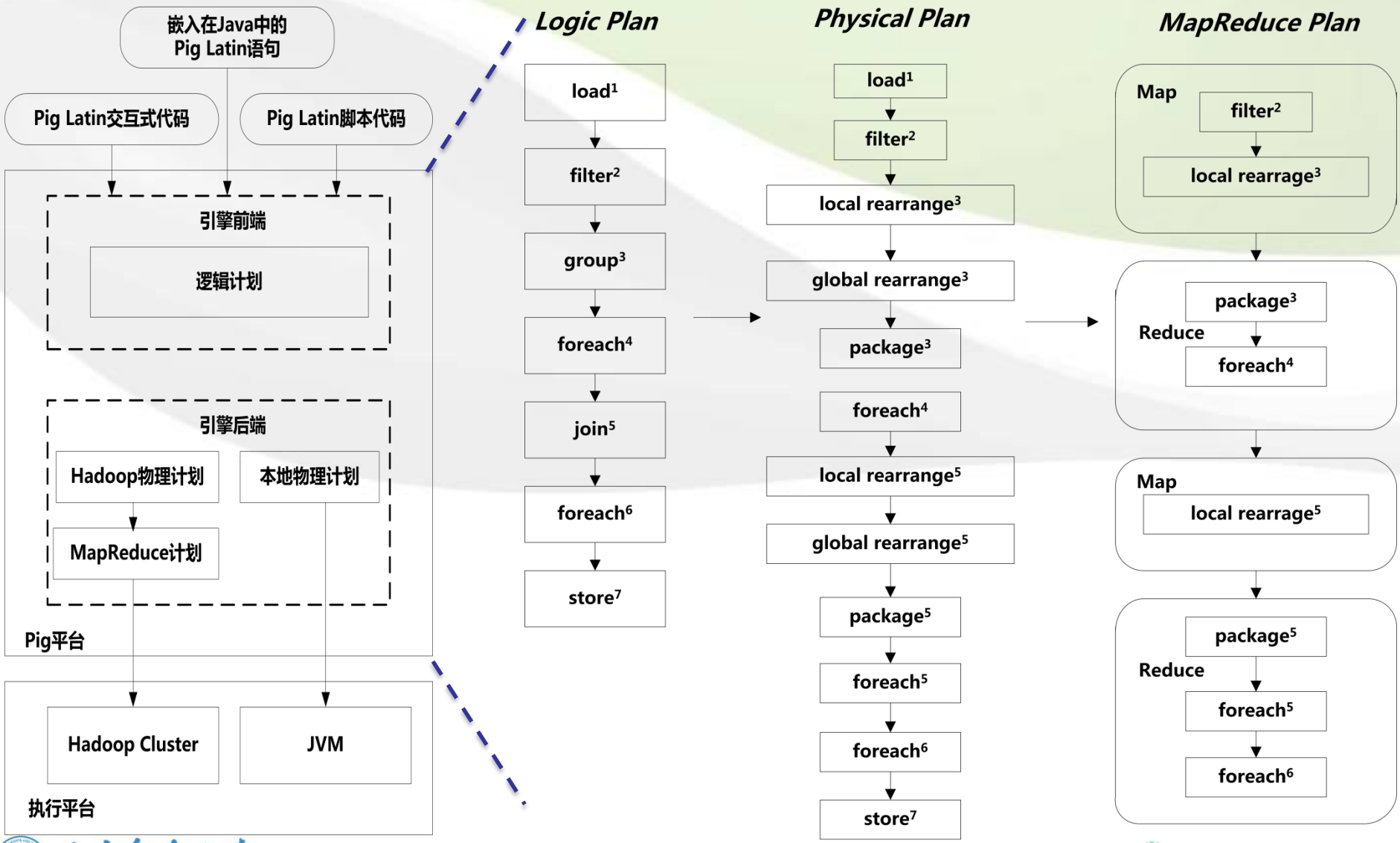
Pig Latin的执行





Pig Latin的执行（cont.）

explain end_records



目录

- Hive
  - 由来
  - 架构
  - HQL及实例
  - 使用UDF扩展Hive
  - Hive使用中的技巧
- Pig
  - 由来
  - 架构
  - Pig Latin及实例
  - 使用UDF扩展Pig
  - Pig使用中的技巧

Pig Latin内置函数

类别	函数	说明
数学	ABS	计算表达式的绝对值
	CEIL	计算表达式的上取整值
	FLOOR	计算表达式的下取整值
	RANDOM	产生一个伪随机数
评价	AVG	计算包（bag）中各项的平均值
	COUNT	计算包中非空值的项的个数
	MAX	计算一个包中所有项的最大值
	MIN	计算一个包中所有项的最小值
	SUM	计算一个包中所有项的和
字符串	INDEXOF	查找一个字符在字符串中的位置
	LOWER	将字符串转换为小写字母
	SUBSTRING	从字符串中截取一段构成新的字符串
	TRIM	去除字符串中的空白字符
包/元组	TOBAG	将一个或多个表达式构成一个包
	TOP	从包中取出指定列值排在前N个的元组
	TOTUPLE	将一个或多个表达式构成一个元组
加载/存储	PigStorage	用字段分隔文本格式加载或存储关系
	BinStorage	用二进制文件加载或存储关系
	BinaryStorage	用二进制文件加载或存储值包含bytearray类型字段的元组
	TextLoadr	从纯文本文件中加载关系，每一行为一个单字段元组

Pig Latin用户自定义函数

- 用户自定义函数（User Defined Function，UDF）
  - 支持使用三种语言开发用户自定义函数
    - Java、Python和JavaScript
  - Java编写的自定义函数功能最强大，可以覆盖数据处理的全过程
    - 编写扩展基本处理函数类的自定义类代码
    - 使用UDF操作的register命令注册包含自定义类的JAR包
    - 在代码中使用自定义函数类

Pig Latin用户自定义函数示例

- 需要找出域名中包含s6的记录
- 扩展过滤函数
  - 所有的过滤函数都是FilterFunc的子类
  - 需要扩展FilterFunc子类，并实现类的exec()接口
- 步骤：
  - 编译并打包到JAR文件filterFunc.jar中
  - 在Pig中注册此JAR文件

    grunt> **register filterFunc.jar;**
  - 调用此自定义过滤函数

    grunt> **s6QQLog = filter log by com.example.pig.lss6(host);**

    grunt> **dump s6QQLog;**

    (125.39.127.30,3,s6.photo.store.qq.com,qq,15,0.3)

```
1:  public class IsQQ extends FilterFunc {
2:    public Boolean exec(Tuple tuple) {
3:      if (tuple == null || tuple.size() == 0) {
4:        return false;
5:      }
6:      try{
7:        Object object = tuple.get(0);
8:        if (object==null){
9:          return false;
10:       }
11:       String str = (String) object;
12:       return str.index("s6")>=0?true:false;
13:     }catch(ExecException e){
14:       throw new IOException;
15:     }
16:   }
17: }
```

Hive与Pig的对比

- Hive和Pig的共同点
  - 都提供了一种简单易用的脚本语言
  - 都是基于MapReduce的数据分析脚本运行环境

数据处理三阶段	数据采集	数据准备	数据呈现
功能	从数据源中获取并传递数据到数据处理系统中的过程	“数据工厂”，对数据进行抽取、转换、加载（ETL），将不能直接反映规律的原始数据进行加工和转化，变为具有实际价值的商业数据	“数据仓库”，数据的存放之处，使用者通过数据仓库提供的工具将满足自己需要的数据提取出来并进行呈现
使用者	-	程序员、数据处理专家或研究人员	数据工程师、分析师或决策者
能力要求	-	对快速到达的数据进行流水式处理	对整理后的数据进行检索、组合和统计后有序呈现

特点	Pig	Hive
1	面向关系型的流式数据处理语言，更适合构建数据流	与传统SQL语言非常接近的使用方式
2	支持在处理流程中出现分支并控制分支的发展	对关系型数据模型的定义支持
3	对大数据集的迭代式处理支持较好，可以对不断到达的数据进行增量处理	与传统商业智能分析软件和基于SQL实现分析系统平滑对接

作业

- 本节问题
  - 将access.log（QQ群共享）文件导入到Hive或Pig中，并使用shell进行查询操作
- 要求：
  - 将代码、查询结果截图，发送到 liujun@bupt.edu.cn （2周内）
- 下节课程预告：
  - Hadoop集群管理与面向未来的海量数据处理