Pig是一种探索大规模数据集的脚本语言。它的诱人之处在于它能够用控制台上的五六行Pig Latin代码轻松处理Tb级数据。

Pig提供了更丰富的数据机构，一般都是多值和嵌套的数据结构。
Pig还提供了一套更强大的数据变换操作，包括在MapReduce中被忽视滴“连接”(join)操作。

Pig包括两部分：
用于描述数据流的语言，成为Pig Latin。
用于运行Pig Latin程序的执行环境。当前有两个环境：单JVM中的贝蒂执行环境和Hadoop集群上的分布式执行环境。

一个Pig Latin程序有一系列的“操作”（operation）或者“变换”(transformation)组成。每一个操作或变换对输入进行数据处理，然后产生输出结果。
这些操作基本上描述了一个数据流。Pig执行环境吧数据流翻译成可执行的内容布标，并裕兴它。在Pig内部，这些变换操作被转换成一系列MapReduce作业。

Pig提供了多个命令用于检查和处理程序中的数据结构，因此，它能够很好地支持程序员写查询。
Pig的一个更有用的特性是它支持在输入数据的一个有代表性的自己上试运行。这样一来，用户可以在处理整个数据集前检查程序中是否有错误。

Pig被设计为可扩展的。处理路径中的每个部分，载入，存储，过滤，分组，连接，都是可以指定的。

Pig并不适合所有的数据处理任务。和MapReduce一样，它是为数据批处理而设计的。如果想执行的查询只涉及一个大型数据集中的一小部分数据，
Pig的实现不会很好，因为它要草庙整个数据集或其中的很大一部分。

执行类型（模式）

本地模式（local mode）

Pig运行在单个JVM中，访问本地文件系统。该模式只是用于Pig处理小规模数据集。
执行类型可用-x 或者 -exectype选项设置。

```
% pig -x local
grount>
```

这样就能启功Grunt。Grunt是Pig的外壳程序（shell）。

MapReduece模式(MapReduece mode)

Pig将查询翻译为MapReduce作业，然后再Hadoop集群上执行。集群可以伪分布的，也可以是全分布的。如果要用Pig处理大规模数据集，
应该是用全分布集群上的MapReduce模式。

设置Pig到Hadoop集群的连接：

1.指定Hadoop版本：% export PIG_HADOOP_VERSION=1.8
2.Pig指向集群的namenode和jobtracker,如果一个或者多个Hadoop站点文件已经定义了
fs.default.name和mapred.job.tracker,把Hadoop的配置文件加到Pig的类路径:

```
% exportPIG_CLASSPATH=$HADOOP_INSTALL/conf/,
```

或者可以在Pig的conf目录下pig.properties文件中设置这两个参数。

启动MapReduce的执行模式：

可以通过设置-x选项或者忽略该选项使用Pig。

```
% pig
grunt>
```

运行Pig程序

脚本

运行包含Pig命令的脚本文件。

pig script.pig　运行本地文件script.pig中的命令。

Grunt

Grunt是运行Pig命令的交互式外壳环境(shell)。可以通过run和exec命令运行 Pig脚本。

嵌入式方法

也可以在Java中运行Pig程序。和使用JDBC运行SQL程序很像。

Grunt包含行编辑功能。有一些快捷键。

Ctrl-E 光标移到行末

Ctrl-P 执行的上一条指令

Ctrl-N 执行的下一条指令

Tab　自动补全Pig Latin关键词和函数名。

quit　退出

help　列出命令

Pig Latin编辑器

PigPen 是一个提供了Pig程序开发环境的Eclipse插件。它包含Pig脚本编辑器，
示例生成器，用于在Hadoop集群上运行脚本的按钮。它还提供一个操作图窗口，能够以图形方式显示脚本，将数据流可视化。

示例

```
-- max_temp.pig:Finds the maximum temperature by year
    records = LOAD 'input/ncdc/micro-tab/sample.txt'
      AS (year:chararray, temperature:int, quality:int);
    filtered_records = FILTER records BY temperature !  =9999 AND
      (quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality  9);
    grouped_records = GROUP filtered_records BY year;
    max_temp = FOREACH grouped_records GENERATE group,
      MAX(filtered_records.temperature);
    DUMP max_temp;
```

```
grunt> DUMP records;
    (1950,0,1)
    (1950,22,1)
    (1950,-11,1)
    (1949,111,1)
    (1949,78,1)
```

```
grunt> DESCRIBE records;
records: {year: chararray,temperature: int,quality: int}
```

```
grunt> grouped_records =GROUP filtered_records BY year;
grunt> DUMP grouped_records;
(1949,{(1949,111,1),(1949,78,1)})
(1950,{(1950,0,1),(1950,22,1),(1950,-11,1)})
```

```
grunt> DESCRIBE grouped_records;
grouped_records: {group: chararray,filtered_records: {year: chararray,
temperature: int,quality: int}}
```

```
grunt> max_temp =FOREACH grouped_records GENERATE group,
>>  MAX(filtered_records.temperature);
```

```
grunt> DUMP max_temp;
(1949,111)
(1950,22)
```

生成示例

Pig提供了ILLUSTRATE操作，能够生成很好地和简明的数据集

只有当

```
grunt> ILLUSTRATE max_temp;
-------------------------------------------------------------------------------
| records | year: bytearray | temperature: bytearray | quality: bytearray |
-------------------------------------------------------------------------------
|         | 1949            | 9999                   | 1                  |
|         | 1949            | 111                    | 1                  |
|         | 1949            | 78                     | 1                  |
-------------------------------------------------------------------------------
-------------------------------------------------------------------
| records | year: chararray | temperature: int | quality: int |
-------------------------------------------------------------------
|         | 1949            | 9999             | 1            |
|         | 1949            | 111              | 1            |
|         | 1949            | 78               | 1            |
-------------------------------------------------------------------
-------------------------------------------------------------------
| filtered_records | year: chararray | temperature: int | quality: int |
-------------------------------------------------------------------
|                  | 1949            | 111              | 1            |
|                  | 1949            | 78               | 1            |
-------------------------------------------------------------------
-------------------------------------------------------------------------------
| grouped_records | group: chararray | filtered_records: bag({year: chararray, |
                                                 temperature: int,quality: int}) |
-------------------------------------------------------------------------------
|                 | 1949             | {(1949, 111, 1), (1949, 78, 1)}       |
-------------------------------------------------------------------------------
-----------------------------------
| max_temp | group: chararray | int |
-----------------------------------
|          | 1949             | 111 |
-----------------------------------
```

注意，Pig 既使用了部分的原始数据(这对于保持生成数据集的真实性很重要)，也创建了一些新的数据。Pig 注意到查询中 9999 这一值，所以创建了一个包含该值的元组来测试 FILTER 语句。

与数据库比较

看上去Pig Latin和SQL很相似，但是有几个方面不同。

1.Pig Latin是一种数据流编程语言，Pig Latin程序是相对于输入的一步步操作。

而sql是一种描述性编程语言，sql语句是一个约束的集合，这些集合在一起，定义了输出。

2.Pig对它所处理的数据要求则宽松得多，可以在运行是定义模式，而且这是可选的。RDBMS把数据存储在严格定义了模式的表内。

3.Pig支持复杂嵌套数据结构。

4.Pig不支持随机读和几十毫秒级别的查询，也不支持一笑部分数据的随机写。（都是批量的，流式的写操作）。

Pig Latin

结构

一个Pig Latin程序由一组语句构成。一个语句可以理解为一个操作或一个命令。

结束符：通常用分号。一般规则是：在Grunt中交互使用的语句或者命令不需要用分号。

例如：grouped_records = GROUP records BY year;

ls / -- 列出Hadoop文件系统中的文件

注释：单行注释用双减号；多行注释用/* */。

语句

Pig Latin程序执行时，解释器会给每个关系操作间里一个逻辑计划，解释器会吧一个语句的逻辑计划加到目前为止已经解析完的出去的逻辑计划上，然后继续处理下一条语句，在整个程序逻辑计划没有构造完成前，Pig并不处理数据。DUMP是一个诊断工具，它触发语句的执行，会把逻辑计划编译成物理计划，并执行。

关系操

表 11-1. Pig Latin 关系操作

类型	操作	描述
加载与存储	LOAD	将数据从文件系统或其他存储中加载数据，存入关系
	STORE	将一个关系存放到文件系统或其他存储中
	DUMP	将关系打印到控制台
过滤	FILTER	从关系中删除不需要的行
	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变换
	SAMPLE	从关系中随机取样
分组与连接	JOIN	连接两个或多个关系
	COGROUP	在两个或更多关系中对数据进行分组
	GROUP	在一个关系中对数据进行分组
	CROSS	获取两个或更多关系的乘积(叉乘)
排序	ORDER	根据一个或多个字段对某个关系进行排序
	LIMIT	将关系的元组个数限定在一定数量内
合并与分割	UNION	合并两个或多个关系
	SPLIT	把某个关系切分两个或多个关系

诊断操

表 11-2. Pig Latin 的诊断操作

操作	描述
DESCRIBE	打印关系的模式
EXPLAIN	打印逻辑和物理计划
ILLUSTRATE	使用生成的输入子集显示逻辑计划的试运行结果

UDF语

为了在 Pig 脚本中使用用户自定义函数，Pig Latin 提供了 REGISTER 和 DEFINE 语句(见表 11-3)。

表 11-3. Pig Latin UDF 语句

语句	描述
REGISTER	在 Pig 运行时环境中注册一个 JAR 文件
DEFINE	为 UDF、流式脚本或命令规范新建别名

命令类

表 11-4. PigLatin 命令类型命令描述

类别	命令	描述
HadooFilesystem	cat	打印一个或多个文件的内容
	cd	改变当前目录
	copyFromLocal	把一个本地文件或目录复制到 Hadoop 文件系统
	copyToLocal	将一个文件或目录从 Hadoop 文件系统复制到本地文件系统
	cp	把一个文件或目录复制到另一个目录
	fs	访问 Hadoop 文件系统外壳程序
	ls	打印文件列表信息
	mkdir	创建新目录
	mv	将一个文件或目录移动到另一个目录
	pwd	打印当前工作目录的路径
	rm	删除一个文件或目录
	rmf	强制删除文件或目录(即使文件或目录不存在也不会失败)
HadoopMapReduce 工具	kill	终止某个 MapReduce 作业
	exec	在一个新的 Grunt 外壳程序中以批处理模式运行一个脚本
	help	显示可用的命令和选项
	quit	退出解释器
	run	在当前 Grunt 外壳程序中运行脚本
	set	设置 Pig 选项

表达式

类型	表达式	描述	示例
常数	文字	常量值(参见表 11-6 中)	1.0, 'a'
字段(通过位置指定)	$n	第 n 个字段(以 0 为基数)	$0
字段(通过名字指定)	f	字段名 f	year
投影	c.$n,c.f	在容器 c(关系、包或元组)中的字段(按位置或名称指定)	records.$0, records.year
Map 查找	m#k	在映射 m 中键 k 所对应的值	items'Coat'
类型转换	(t)f	将字段 f 转换为类型 t	(int) year

类别	数据类型	描述	文字示例
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	1.0F
	double	64 位浮点数	1.0
文本	chararray	UTF-16 格式的字符数组	'a'
二进制	Bytearray	字节数组	不支持
复杂类型	tuple	任何类型的字段序列	(1,'pomegranate')
	bag	元组的无序多重集合(允许重复元组)	{(1, 'pomegranate'),(2)}
	map	一组键-值对。键必须是字符数组，值可以是任何类型的数据	['a' 'pomegranate']

类型

类别	数据类型	描述	文字示例
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	1.0F
	double	64 位浮点数	1.0
文本	chararray	UTF-16 格式的字符数组	'a'
二进制	Bytearray	字节数组	不支持
复杂类型	tuple	任何类型的字段序列	(1,'pomegranate')
	bag	元组的无序多重集合(允许重复元组)	{(1, 'pomegranate'),(2)}
	map	一组键-值对。键必须是字符数组，值可以是任何类型的数据	['a' 'pomegranate']

模式

Pig中一个关系可以有一个关联的模式。模式为关系的字段指定名称和类型。
例子

```
grunt> recordes = LOAD 'input/ncdc/micro-tab/sample.txt'
    >> AS (year:int, temperature:int, quality:int);
grunt> DESCRIBE recordes;
records: {year: int, temperature: int,  quality: int}
```

也可以忽略类型的声明,默认的情况下最通用的类型是bytearray。

```
grunt> recordes = LOAD 'input/ncdc/micro-tab/sample.txt'
    >> AS (year, temperature, quality);
grunt> DESCRIBE recordes;
records: {year: bytearray, temperature:bytearray,  quality: bytearray}
```

没有对应模式的关系

```
grunt> recordes = LOAD 'input/ncdc/micro-tab/sample.txt'
grunt> DESCRIBE recordes;
Schema for records unknown.
```

只能使用位置符号引用没有对已昂模式的关系中的字段。

```
grunt> projected_recordes = FOREACH records Generate $0,$1,$2;
grunt> DESCRIBE projected_recordes ;
projected_recordes : {bytearray,bytearray,bytearray}
```

验证与空值：

Pig在处理损坏数据是会为违例的值产生一个null，空值null被显示成一个空位。

```
grunt> recordes = LOAD 'input/ncdc/micro-tab/sample.txt'
    >> AS (year:bytearray, temperature:int, quality:int);
grunt> DUMP recordes;
(1950,0,1)
(1950,22,1)
(1950,,1)
(1950,111,1)
```

过滤破损记录

```
grunt> corrupt_recordes = FILLER records BY temperature is NULL;
grunt> DUMP recordes;
(1950,,1)
```

模式合并：

针对数据流中的任何关系，可以使用DESCRIBE操作来获取他们的模式。如果要重新定义一个关系的模式，可以使用带AS字句的FOREACH...GENERATE来定义输入关系的一部分或全部字段的模式。

函数

计算函数(Eval function)

计算函数获取一个或多个表达式作为输入，并返回另一个表达式。

过滤函数(Filler function)

过滤函数是一类特殊的计算函数。这类函数返回的是逻辑布尔值。

加载函数(Load function)

加载函数指明如何从外部存储加载数据到一个关系。

存储函数(Store function)

存储函数指明如何把一个关系中的内容存到外部存储。

内置函数：

类别	函数名称	描述
计算	AVG	计算包中项的平均值
	CONCAT	把两个字节数组或字符数组连接成一个
	COUNT	计算一个包中非空值的项的个数
	COUNTSTAR	计算一个包的项的个数，包括空值
	DIFF	计算两个包的差。如果两个参数不是包，那么如果它们相同，则返回一个包含这两个参数的包；否则返回一个空的包
	MAX	计算一个包中项的最大值
	MIN	计算一个包中项的最小值

类别	函数名称	描述
	SIZE	计算一个类型的大小。数值型的大小总是 1；对于字符数组，它返回字符的个数；对于字节数组，它返回字节的个数；对于容器(container，包括元组、包、映射)它返回其中项的个数
	SUM	计算一个包中项的值的总和
	TOKENIZE	对一个字符数组进行标记解析，并把结果词放入一个包
过滤	IsEmpty	判断一个包或映射是否为空
加载/存储	PigStorage	用字段分隔文本格式加载或存储关系。每一行被分为字段后(用一个可设置的分隔符(默认为一个制表符)分隔)，分别对应于元组的各个字段。这是不指定加载/存储方式时的默认存储函数
	BinStorage	从二进制文件加载一个关系或把关系存储到二进制文件中。该函数使用基于 Hadoop Writable 对象的 Pig 内部格式
	BinaryStorage	从二进制文件加载只包含一个类型为 bytearray 的字段的元组到关系，或以这种格式存储一个关系。bytearray 中的字节逐字存放。该函数与 Pig 的流式处理结合使用
	TextLoader	从纯文本格式加载一个关系。每一行对应于一个元组。每个元组只包含一个字段，即该行文本。
	PigDump	用元组的 toString()形式存储关系。每行一个元组。这个函数对设计很有帮助。

用户自定义函数(user-defined function　UDF)

UDF用Java编写。

过滤UDF

例子：

filered_records = FILTER records BY temperature != 9999 AND
(quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);

可简化成

filered_records = FILTER records BY temperature != 9999 AND isGood(quality);

写一个函数

```
package com.hadoopbook.pig;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.pig.FilterFunc;

import org.apache.pig.backend.executionengine.ExecException;
import org.apache.pig.data.DataType;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.logicalLayer.FrontendException;

public class IsGoodQuality extends FilterFunc {

@Override
public Boolean exec(Tuple tuple) throws IOException {
  if (tuple == null || tuple.size() == 0) {
    return false;
  }

    try {
      Object object = tuple.get(0);
      if (object == null) {

        return false;
      }
      int i = (Integer) object;
      return i== 0 || i == 1 || i == 4 || i == 5 || i == 9;
    } catch (ExecException e) {
      throw new IOException(e);
    }
  }

}
```

对上述函数定义并打包到一个jar文件。

通过Rigister操作告诉pig这个JAR文件的信息：

```
grunt> REGISTER pig-examples.jar;

grunt> filered_records = FILTER records BY temperature != 9999 AND
>> com.hadoopbook.pig.IsGoodQuality(quality);
```

或者

```
grunt> DEFINE isGood com.hadoopbook.pig.IsGoodQuality();        定义一个别名
grunt> filered_records = FILTER records BY temperature != 9999 AND isGood(quality);
```

计算UDF

加载UDF

数据处理操作

加载和存储数据

```
grunt> STORE A INTO 'out' USING PIGSTROAGE(':');
grunt> =cat out
Joe:cherry:2
Ali:appale:3
Joe:banana:2
Eve:apple:7
```

过滤数据

```
grunt> DUMP A;
(Joe,cherry,2)
(Ali,appale,3)
(Joe,banana,2)
(Eve,apple,7)
grunt> B = FOREACH A GENERTAE $0,$2+1,'Constant';
grunt> DUMP B;
(Joe,3,Constant)
(Ali,4,Constant)
(Joe,3,Constant)
(Eve,8,Constant)
```

STREAM

STREAM操作让你可以用外部程序或脚本对短息中的数据进行变换。

```
grunt> DUMP A;
(Joe,cherry,2)
(Ali,appale,3)
(Joe,banana,2)
(Eve,apple,7)
grunt> C = STREAM A THROUGH 'cut -f 2';
grunt> DUMP C;
(cherry)
(appale)
(banana)
(apple)
```

分组与连接数据

JOIN

```
grunt> DUMP A;          grunt> DUMP B;
(2,Tie)                 (Joe,2)
(4,Coat)                (Hank,4)
(3,Hat)                 (Ali,0)
(1,Scarf)               (Eve,3)
                        (Hank,2)
grunt> C = JOIN A BY $0, B BY $1;
grunt> DUMP C;
(2,Tie,Joe,2)
(2,Tie,Hank,2)
(3,Hat,Eve,3)
(4,Coat,Hank,4)

grunt> C = JOIN A BY $0, LEFT OUTER B BY $1;
grunt> DUMP C;
(1,Scarf,,)
(2,Tie,Joe,2)
(2,Tie,Hank,2)
(3,Hat,Eve,3)
(4,Coat,Hank,4)
```

COGROUP

COGROUP语句和JOIN类型，但是不同点在于，它会创建一组嵌套的输出元组集合。

COGROUP为每个不同的分组健值生成一个元组，每个元组的第一个字段就是那个键值。

其他字段是哥哥恭喜中匹配该键值的元组所组成的“包”(bag).

```
grunt> D = COGROUP A BY $0, B BY $1;
grunt> DUMP D;
(0,{},{(Ali,0)})
(1,{(1,Scarf)},{})
(2,{(2,Tie)},{(Joe,2),(Hank,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})
```

CROSS

Pig Latin包含“叉乘”(cross-product,也叫“笛卡尔积”)的操作。
这一操作吧一个关系中的每个元组和第二个中的所有元组进行连接。
grunt> I = CROSS A, B;
grunt> DUMP I;
(2,Tie,Joe,2)
(2,Tie,Hank,4)
(2,Tie,Ali,0)
(2,Tie,Eve,3)
(2,Tie,Hank,2)
(4,Coat ...
... ...

GROUP

COGROUP用于把两个或多个关系中的数据放到一起，而GROUP语句则对一个关系中的数据进行分组。
grunt> DUMP A;
(Joe,cherry)
(Ali，apple)
(Joe，banana)
(Eve，apple)

grunt>B = GROUP A BY SIZE($1);
grunt> DUMP B;
(5L,{(Ali，apple),(Eve，apple)})
(6L,{(Joe,cherry),(Joe，banana})

对数据进行排序

grunt> DUMP A;
(2,3)
(1,2)
(2,4)

grunt>B = ORDER A BY $0,$1 DESE;
grunt> DUMP B;
(1,2)
(2,4)
(2,3)

组合和切分数据

grunt> DUMP A;
(2,3)
(1,2)
(2,4)

grunt> DUMP B;
(z,x,8)
(w,y,1)

grunt> C = UNION A, B;
grunt> DUMP C;
(z,x,8)
(w,y,1)
(2,3)
(1,2)
(2,4)
因为关系本身是无序的，因此C中元组的顺序是不确定的。