[原创]Apache Pig的一些基础概念及用法总结（1）

本文可以让刚接触pig的人对一些基础概念有个初步的了解。
本文大概是互联网上第一篇公开发表的且涵盖大量实际例子的Apache Pig中文教程（由Google搜索可知），文中的大量实例都是作者Darran Zhang（website: codelast.com）在工作、学习中总结的经验或解决的问题，并且添加了较为详尽的说明及注解，此外，作者还在不断地添加本文的内容，希望能帮助一部分人。

Apache pig [http://pig.apache.org/] 是用来处理大规模数据的高级查询语言，配合Hadoop使用，可以在处理海量数据时达到事半功倍的效果，比使用Java，C++等语言编写大规模数据处理程序的难度要小N倍，实现同样的效果的代码量也小N倍。Twitter就大量使用pig来处理海量数据——有兴趣的，可以看Twitter工程师写的这个PPT [http://www.slideshare.net/kevinweil/hadoop-pig-and-twitter-nosql-east-2009]。
但是，刚接触pig时，可能会觉得里面的某些概念以及程序实现方法与想像中的很不一样，甚至有些莫名，所以，你需要仔细地研究一下基础概念，这样在写pig程序的时候，才不会觉得非常别扭。

本文基于以下环境：
pig 0.8.1

先给出两个链接：pig参考手册1 [http://pig.apache.org/docs/r0.8.1/piglatin_ref1.html]，pig参考手册2 [http://pig.apache.org/docs/r0.8.1/piglatin_ref2.html]。本文的部分内容来自这两个手册，但涉及到翻译的部分，也是我自己翻译的，因此可能理解与英文有偏差，如果你觉得有疑义，可参考英文内容。

【配置Pig语法高亮】
在正式开始学习Pig之前，你首先要明白，配置好编辑器的Pig语法高亮是很有用的，它可以极大地提高你的工作效率。

如果你在Windows下编写Pig代码，好像还真没有什么轻量级的编辑器插件（例如Notepad++的插件之类的）可以实现对.pig文件的语法高亮显示，我建议你使用Notepad++，在“User Define Language”中自定义Pig语法高亮方案（我这样做之后感觉效果很好）；如果你觉得麻烦，那么你可以直接用Notepad++以SQL的语法高亮来查看Pig代码，这样的话可以高亮Pig中的一部分关键字。
在Linux下，选择就很多了，大分部人使用的是vi，vim，但我是个Emacs控，所以我就先说说如何配置Emacs的Pig语法高亮。此插件是一个很好的选择：https://github.com/cloudera/piglatin-mode
那么，怎么使用这个插件呢？
下载piglatin.el文件，将它放置在任何地方——当然，为了方便，最好是放在你登录用户的根目录下（也就是与.emacs配置文件在同一目录下），然后将其重命名为“.piglatin.el”注意前面是有一个点的，也就是说将这个文件设置成隐藏文件，否则可能会误删了。
然后，在 .emacs 文件中的最后，添加上如下一行：
(load-file "/home/abc/.piglatin.el")
这里假设了你的 .piglatin.el 文件放置的位置是在 /home/abc/ 目录下，也就是说emacs会加载这个文件，实现语法高亮显示。
现在，你再打开一个.pig文件试试看？非常令人赏心悦目的高亮效果就出来了。效果如下图所示：

```
File Edit Options Buffers Tools Help
A = LOAD '1.txt' AS (col1:chararray, col2:int);
B = GROUP A BY col1;
C = FOREACH B {
  D = FILTER A BY col2 == 3;
  E = FILTER A BY col2 == 6;
  GENERATE group, COUNT(D), COUNT(E);
};
DUMP C;
```


其实Emacs也有Windows版的，如果你习惯在Windows下工作，完全可以在Windows下按上面的方法配置一下Pig语法高亮（但是Windows版的Emacs还需要一些额外的配置工作，例如修改注册表等，所以会比在Linux下使用要麻烦一些，具体请看这篇文章 [http://www.codelast.com/?p=4802]）。

[http://www.codelast.com/]

下面开始学习Pig。

(1) 关系（relation）、包（bag）、元组（tuple）、字段（field）、数据（data）的关系

- 一个关系（relation）是一个包（bag），更具体地说，是一个外部的包（outer bag）。
- 一个包（bag）是一个元组（tuple）的集合。在pig中表示数据时，用大括号{}括起来的东西表示一个包——无论是在教程中的实例演示，还是在pig交互模式下的输出，都遵循这样的约定，请牢记这一点，因为不理解的话就会对数据结构的掌握产生偏差。
- 一个元组（tuple）是若干字段（field）的一个有序集（ordered set）。在pig中表示数据时，用小括号()括起来的东西表示一个元组。
- 一个字段是一块数据（data）。

“元组”这个词很抽象，你可以把它想像成关系型数据库表中的一行，它含有一个或多个字段，其中，每一个字段可以是任何数据类型，并且可以有或者没有数据。
“关系”可以比喻成关系型数据库的一张表，而上面说了，“元组”可以比喻成数据表中的一行，那么这里有人要问了，在关系型数据库中，同一张表中的每一行都有固定的字段数，pig中的“关系”与“元组”之间，是否也是这样的情况呢？不是的。“关系”并不要求每一个“元组”都含有相同数量的字段，并且也不会要求各“元组”中在相同位置处的字段具有相同的数据类型（太随意了，是吧？）

[http://www.codelast.com/]

(2) 一个 计算多维度组合下的平均值 的实际例子
为了帮助大家理解pig的一个基本的数据处理流程，我造了一些简单的数据来举个例子——
假设有数据文件：a.txt（各数值之间是以tab分隔的）：

```
[root@localhost pig]$ cat a.txt
a 1 2 3 4.2 9.8
a 3 0 5 3.5 2.1
b 7 9 9 - -
a 7 9 9 2.6 6.2
a 1 2 5 7.7 5.9
a 1 2 3 1.4 0.2
```

问题如下：怎样求出在第2、3、4列的所有组合的情况下，最后两列的平均值分别是多少？
例如，第2、3、4列有一个组合为（1，2，3），即第一行和最后一行数据。对这个维度组合来说，最后两列的平均值分别为：
（4.2+1.4）/2=2.8
（9.8+0.2）/2=5.0
而对于第2、3、4列的其他所有维度组合，都分别只有一行数据，因此最后两列的平均值其实就是它们自身。
特别地，组合（7，9，9）有两行记录：第三、四行，但是第三行数据的最后两列没有值，因此它不应该被用于平均值的计算，也就是说，在计算平均值时，第三行是无效数据。所以（7，9，9）组合的最后两列的平均值为 2.6 和 6.2。
我们现在用pig来算一下，并且输出最终的结果。
先进入本地调试模式（pig -x local），再依次输入如下pig代码：

```
A = LOAD 'a.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double);
B = GROUP A BY (col2, col3, col4);
C = FOREACH B GENERATE group, AVG(A.col5), AVG(A.col6);
DUMP C;
```

pig输出结果如下：

```
((1,2,3),2.8,5.0)
((1,2,5),7.7,5.9)
((3,0,5),3.5,2.1)
((7,9,9),2.6,6.2)
```

这个结果对吗？手工算一下就知道是对的。

[http://www.codelast.com/]

下面，我们依次来看看每一句pig代码分别得到了什么样的数据。
①加载 a.txt 文件，并指定每一列的数据类型分别为 chararray（字符串），int，int，int，double，double。同时，我们还给予了每一列别名，分别为 col1，col2，……，col6。这个别名在后面的数据处理中会用到——如果你不指定别名，那么在后面的处理中，就只能使用索引（$0，$1，……）来标识相应的列了，这样可读性会变差，因此，在列固定的情况下，还是指定别名的好。
将数据加载之后，保存到变量A中，A的数据结构如下：

```
A: {col1: chararray,col2: int,col3: int,col4: int,col5: double,col6: double}
```

可见，A是用大括号括起来的东西。根据本文前面的说法，A是一个包（bag）。
这个时候，A与你想像中的样子应该是一致的，也就是与前面打印出来的 a.txt 文件的内容是一样的，还是一行一行的类似于“二维表”的数据。

[http://www.codelast.com/]

②按照A的第2、3、4列，对A进行分组。pig会找出所有第2、3、4列的组合，并按照升序进行排列，然后将它们与对应的包A整合起来，得到如下的数据结构：

```
B: {group: (col2: int,col3: int,col4: int),A: {col1: chararray,col2: int,col3: int,col4: int,col5: double,col6:
```

可见，A的第2、3、4列的组合被pig赋予了一个别名：group，这很形象。同时我们也观察到，B的每一行其实就是由一个group和若干个A组成的——注意，是若干个A。这里之所以只显示了一个A，是因为这里表示的是数据结构，而不表示具体数据有多少组。
实际的数据为：

```
((1,2,3),{(a,1,2,3,4.2,9.8),(a,1,2,3,1.4,0.2)})
((1,2,5),{(a,1,2,5,7.7,5.9)})
((3,0,5),{(a,3,0,5,3.5,2.1)})
((7,9,9),{(b,7,9,9,,),(a,7,9,9,2.6,6.2)})
```

可见，与前面所说的一样，组合（1，2，3）对应了两行数据，组合（7，9，9）也对应了两行数据。
这个时候，B的结构就不那么明朗了，可能与你想像中有一点不一样了。

③计算每一种组合下的最后两列的平均值。
根据上面得到的B的数据，你可以把B想像成一行一行的数据（只不过这些行不是对称的），FOREACH 的作用是对 B 的每一行数据进行遍历，然后进行计算。
GENERATE 可以理解为要生成什么样的数据，这里的 group 就是上一步操作中B的第一项数据（即pig为A的第2、3、4列的组合赋予的别名），所以它告诉了我们：在数据集 C 的每一行里，第一项就是B中的group——类似于（1，2，5）这样的东西）。
而 AVG(A.col5) 这样的计算，则是调用了pig的一个求平均值的函数 AVG，用于对 A 的名为 col5 的列求平均值。前文说了，在加载数据到A的时候，我们已经给每一列起了个别名，col5就是倒数第二列。
到这里，可能有人要迷糊了：难道 AVG(A.col5) 不是表示对 A 的col5这一列求平均值吗？也就是说，在遍历B（FOREACH B）的每一行时候，计算结果都是相同的啊！
事实上并不是这样。我们遍历的是B，我们需要注意到，B的数据结构中，每一行数据里，一个group对应的是若干个A，因此，这里的 A.col5，指的是B的每一行中的A，而不是包含全部数据的那个A。拿B的第一行来举例：
((1,2,3),{(a,1,2,3,4.2,9.8),(a,1,2,3,1.4,0.2)})
遍历到B的这一行时，要计算AVG(A.col5)，pig会找到 (a,1,2,3,4.2,9.8) 中的4.2，以及(a,1,2,3,1.4,0.2)中的1.4，加起来除以2，就得到了平均值。
同理，我们也知道了AVG(A.col6)是怎么算出来的。但还有一点要注意的：对(7,9,9)这个组，它对应的数据(b,7,9,9,,)里最后两列是无值的，这是因为我们的数据文件对应位置上不是有效数字，而是两个“-”，pig在加载数据的时候自动将它置为空了，并且计算平均值的时候，也不会把这一组数据考虑在内（相当于忽略这组数据的存在）。
到了这里，我们不难理解，为什么C的数据结构是这样的了：

```
C: {group: (col2: int,col3: int,col4: int),double,double}
```


④DUMP C就是将C中的数据输出到控制台。如果要输出到文件，需要使用：

```
STORE C INTO 'output';
```

这样pig就会在当前目录下新建一个“output”目录（该目录必须事先不存在），并把结果文件放到该目录下。

请想像一下，如果要实现相同的功能，用Java或C++写一个Map-Reduce应用程序需要多少时间？可能仅仅是写一个build.xml或者Makefile，所需的时间就是写这段pig代码的几十倍了！
正因为pig有如此优势，它才得到了广泛应用。

(3) 怎样统计数据行数
在SQL语句中，要统计表中数据的行数，很简单：

```
SELECT COUNT(*) FROM table_name WHERE condition
```

在pig中，也有一个COUNT函数，在pig手册中，对COUNT函数有这样的说明：

> *Computes the number of elements in a bag.*

假设要计算数据文件a.txt的行数：

```
[root@localhost pig]$ cat a.txt
a 1 2 3 4.2 9.8
a 3 0 5 3.5 2.1
b 7 9 9 - -
a 7 9 9 2.6 6.2
a 1 2 5 7.7 5.9
a 1 2 3 1.4 0.2
```

你是否可以这样做呢：

```
A = LOAD 'a.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double);
B = COUNT(*);
DUMP B;
```

答案是：绝对不行。pig会报错。pig手册中写得很明白：

> *Note: You cannot use the tuple designator (*) with COUNT; that is, COUNT(*) will not work.*

那么，这样对某一列计数行不行呢：

```
B = COUNT(A.col2);
```

答案是：仍然不行。pig会报错。
这就与我们想像中的“正确做法”有点不一样了：我为什么不能直接统计一个字段的数目有多少呢？刚接触pig的时候，一定非常疑惑这样明显“不应该出错”的写法为什么行不通。
要统计A中含col2字段的数据有多少行，正确的做法是：

```
A = LOAD 'a.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double);
```

```
B = GROUP A ALL;
C = FOREACH B GENERATE COUNT(A.col2);
DUMP C;
```

输出结果：

```
(6)
```

表明有6行数据。
如此麻烦？没错。这是由pig的数据结构决定的。

[http://www.codelast.com/]

在这个例子中，统计COUNT(A.col2)和COUNT(A)的结果是一样的，但是，如果col2这一列中含有空值：

```
[root@localhost pig]$ cat a.txt
a 1 2 3 4.2 9.8
a   0 5 3.5 2.1
b 7 9 9 - -
a 7 9 9 2.6 6.2
a 1 2 5 7.7 5.9
a 1 2 3 1.4 0.2
```

则以下pig程序及执行结果为：

```
grunt> A = LOAD 'a.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double);
grunt> B = GROUP A ALL;
grunt> C = FOREACH B GENERATE COUNT(A.col2);
grunt> DUMP C;
(5)
```

可见，结果为5行。那是因为你LOAD数据的时候指定了col2的数据类型为int，而a.txt的第二行数据是空的，因此数据加载到A以后，有一个字段就是空的：

```
grunt> DUMP A;
(a,1,2,3,4.2,9.8)
(a,,0,5,3.5,2.1)
(b,7,9,9,,)
(a,7,9,9,2.6,6.2)
(a,1,2,5,7.7,5.9)
(a,1,2,3,1.4,0.2)
```

在COUNT的时候，null的字段不会被计入在内，所以结果是5。

> *The COUNT function follows syntax semantics and ignores nulls. What this means is that a tuple in the bag will not be counted if the first field in this tuple is NULL. If you want to include NULL values in the count computation, use COUNT_STAR.*

[http://www.codelast.com/]

(4) FLATTEN操作符的作用
这个玩意一开始还是挺让我费解的。从字面上看，flatten就是“弄平”的意思，但是在对一个pig的数据结构操作时，flatten到底是“弄平”了什么，又有什么作用呢？
我们还是采用前面的a.txt数据文件来说明：

```
[root@localhost pig]$ cat a.txt
a 1 2 3 4.2 9.8
a 3 0 5 3.5 2.1
b 7 9 9 - -
a 7 9 9 2.6 6.2
a 1 2 5 7.7 5.9
a 1 2 3 1.4 0.2
```

如果我们按照前文的做法，计算多维度组合下的最后两列的平均值，则：

```
grunt> A = LOAD 'a.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double);
grunt> B = GROUP A BY (col2, col3, col4);
grunt> C = FOREACH B GENERATE group, AVG(A.col5), AVG(A.col6);
grunt> DUMP C;
((1,2,3),2.8,5.0)
((1,2,5),7.7,5.9)
((3,0,5),3.5,2.1)
((7,9,9),2.6,6.2)
```

可见，输出结果中，每一行的第一项是一个tuple（元组），我们来试试看 FLATTEN 的作用：

```
grunt> A = LOAD 'a.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double);
grunt> B = GROUP A BY (col2, col3, col4);
grunt> C = FOREACH B GENERATE FLATTEN(group), AVG(A.col5), AVG(A.col6);
grunt> DUMP C;
(1,2,3,2.8,5.0)
(1,2,5,7.7,5.9)
(3,0,5,3.5,2.1)
(7,9,9,2.6,6.2)
```

看到了吗？被 FLATTEN 的group本来是一个元组，现在变成了扁平的结构了。按照pig文档的说法，FLATTEN用于对元组（tuple）和包（bag）“解嵌套”（un-nest）：

> *The FLATTEN operator looks like a UDF syntactically, but it is actually an operator that changes the structure of tuples and bags in a way that a UDF cannot. Flatten un-nests tuples as well as bags. The idea is the same, but the operation and result is different for each type of structure.*
>
> *For tuples, flatten substitutes the fields of a tuple in place of the tuple. For example, consider a relation that has a tuple of the form (a, (b, c)). The expression GENERATE $0, flatten($1), will cause that tuple to become (a, b, c).*

所以我们就看到了上面的结果。
在有的时候，不“解嵌套”的数据结构是不利于观察的，输出这样的数据可能不利于外围数程序的处理（例如，pig将数据输出到磁盘后，我们还需要用其他程序做后续处理，而对一个元组，输出的内容里是含括号的，这就在处理流程上又要多一道去括号的工序），因此，FLATTEN提供了一个让我们在某些情况下可以清楚、方便地分析数据的机会。

(5)关于GROUP操作符
在上文的例子中，已经演示了GROUP操作符会生成什么样的数据。在这里，需要说得更理论一些：

- 用于GROUP的key如果多于一个字段（正如本文前面的例子），则GROUP之后的数据的key是一个元组（tuple），否则它就是与用于GROUP的key相同类型的东西。
- GROUP的结果是一个关系（relation），在这个关系中，每一组包含一个元组（tuple），这个元组包含两个字段：(1)第一个字段被命名为“group”——这一点非常容易与GROUP关键字相混淆，但请区分开来。该字段的类型与用于GROUP的key类型相同。(2)第二个字段是一个包（bag），它的类型与被GROUP的关系的类型相同。

(6)把数据当作“元组”（tuple）来加载
还是假设有如下数据：

```
[root@localhost pig]$ cat a.txt
a 1 2 3 4.2 9.8
a 3 0 5 3.5 2.1
b 7 9 9 - -
a 7 9 9 2.6 6.2
a 1 2 5 7.7 5.9
a 1 2 3 1.4 0.2
```

如果我们按照以下方式来加载数据：

```
A = LOAD 'a.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double);
```

那么得到的A的数据结构为：

```
grunt> DESCRIBE A;
A: {col1: chararray,col2: int,col3: int,col4: int,col5: double,col6: double}
```

如果你要把A当作一个元组（tuple）来加载：

```
A = LOAD 'a.txt' AS (T : tuple (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double));
```

也就是想要得到这样的数据结构：

```
grunt> DESCRIBE A;
A: {T: (col1: chararray,col2: int,col3: int,col4: int,col5: double,col6: double)}
```

那么，上面的方法将得到一个空的A：

```
grunt> DUMP A;
()
()
()
()
()
()
```

那是因为数据文件a.txt的结构不适合于这样加载成元组（tuple）。

如果有数据文件b.txt：

```
[root@localhost pig]$ cat b.txt
(a,1,2,3,4.2,9.8)
(a,3,0,5,3.5,2.1)
(b,7,9,9,-,-)
(a,7,9,9,2.6,6.2)
(a,1,2,5,7.7,5.9)
(a,1,2,3,1.4,0.2)
```

则使用上面所说的加载方法及结果为：

```
grunt> A = LOAD 'b.txt' AS (T : tuple (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double));
grunt> DUMP A;
((a,1,2,3,4.2,9.8))
((a,3,0,5,3.5,2.1))
((b,7,9,9,,))
((a,7,9,9,2.6,6.2))
```

```
((a,1,2,5,7.7,5.9))
((a,1,2,3,1.4,0.2))
```

可见，加载的数据的结构确实被定义成了元组（tuple）。

(7)在多维度组合下，如何计算某个维度组合里的不重复记录的条数
以数据文件 c.txt 为例：

```
[root@localhost pig]$ cat c.txt
a 1 2 3 4.2 9.8 100
a 3 0 5 3.5 2.1 200
b 7 9 9 - - 300
a 7 9 9 2.6 6.2 300
a 1 2 5 7.7 5.9 200
a 1 2 3 1.4 0.2 500
```

问题：如何计算在第2、3、4列的所有维度组合下，最后一列不重复的记录分别有多少条？例如，第2、3、4列有一个维度组合是（1，2，3），在这个维度维度下，最后一列有两种值：100 和 500，因此不重复的记录数为2。同理可求得其他的记录条数。
pig代码及输出结果如下：

```
grunt> A = LOAD 'c.txt' AS (col1:chararray, col2:int, col3:int, col4:int, col5:double, col6:double, col7:int);
grunt> B = GROUP A BY (col2, col3, col4);
grunt> C = FOREACH B {D = DISTINCT A.col7; GENERATE group, COUNT(D);};
grunt> DUMP C;
((1,2,3),2)
((1,2,5),1)
((3,0,5),1)
((7,9,9),1)
```

我们来看看每一步分别生成了什么样的数据：
①LOAD不用说了，就是加载数据；
②GROUP也不用说了，和前文所说的一样。GROUP之后得到了这样的数据：

```
grunt> DUMP B;
((1,2,3),{(a,1,2,3,4.2,9.8,100),(a,1,2,3,1.4,0.2,500)})
((1,2,5),{(a,1,2,5,7.7,5.9,200)})
((3,0,5),{(a,3,0,5,3.5,2.1,200)})
((7,9,9),{(b,7,9,9,,,300),(a,7,9,9,2.6,6.2,300)})
```

其实到这里，我们肉眼就可以看出来最后要求的结果是什么了，当然，必须要由pig代码来完成，要不然怎么应对海量数据？
[http://www.codelast.com/]
③这里的 FOREACH 与前面有点不一样，第一次看到这种写法，肯定会觉得很奇怪。先看一下用于去重的DISTINCT关键字的说明：

> *Removes duplicate tuples in a relation.*

然后再解释一下：FOREACH 是对B的每一行进行遍历，其中，B的每一行里含有一个包（bag），每一个包中含有若干元组（tuple）A，因此，FOREACH 后面的大括号里的操作，其实是对所谓的“内部包”（inner bag）的操作（详情请参看FOREACH的说明 [http://pig.apache.org/docs/r0.8.1/piglatin_ref2.html#FOREACH]），在这里，我们指定了对A的col7这一列进行去重，去重的结果被命名为D，然后再对D计数（COUNT），就得到了我们想要的结果。
④输出结果数据，与前文所述的差不多。
这样就达成了我们的目的。从总体上说，刚接触pig不久的人会觉得这些写法怪怪的，就是扭不过来，但是要坚持，时间长了，连倒影也会让你觉得是正的了。

(8)如何将关系（relation）转换为标量（scalar）
在前文中，我们要统计符合某些条件的数据的条数，使用了COUNT函数来计算，但在COUNT之后，我们得到的还是一个关系（relation），而不是一个标量的数字，如何把一个关系转换为标量，从而可以在后续处理中便于使用呢？
具体请看这个链接 [http://pig.apache.org/docs/r0.8.1/piglatin_ref2.html#Casting+ Relations+ to+ Scalars]。

(9)pig中如何使用shell进行辅助数据处理
pig中可以嵌套使用shell进行辅助处理，下面，就以一个实际的例子来说明。
假设我们在某一步pig处理后，得到了类似于下面 b.txt 中的数据：

```
[root@localhost pig]$ cat b.txt
1 5 98  = 7
34  8 6 3 2
62  0 6 = 65
```

问题：如何将数据中第4列中的“=”符号全部替换为9999？
pig代码及输出结果如下：

```
grunt> A = LOAD 'b.txt' AS (col1:int, col2:int, col3:int, col4:chararray, col5:int);
grunt> B = STREAM A THROUGH `awk '{if($4 == "=") print $1"\t"$2"\t"$3"\t9999\t"$5; else print $0}'`;
grunt> DUMP B;
(1,5,98,9999,7)
(34,8,6,3,2)
(62,0,6,9999,65)
```

我们来看看这段代码是如何做到的：
①加载数据，这个没什么好说的。

②通过“STREAM … THROUGH …”的方式，我们可以调用一个shell语句，用该shell语句对A的每一行数据进行处理。此处的shell逻辑为：当某一行数据的第4列为“=”符号时，将其替换为“9999”；否则就照原样输出这一行。
③输出B，可见结果正确。

（10）向pig脚本中传入参数
假设你的pig脚本输出的文件是通过外部参数指定的，则此参数不能写死，需要传入。在pig中，使用传入的参数如下所示：

```
STORE A INTO '$output_dir';
```

则这个“output_dir”就是个传入的参数。在调用这个pig脚本的shell脚本中，我们可以这样传入参数：

```
pig -param output_dir="/home/my_ourput_dir/" my_pig_script.pig
```

这里传入的参数“output_dir”的值为“/home/my_output_dir/”。
[http://www.codelast.com/]
（11）就算是同样一段pig代码，多次计算所得的结果也有可能是不同的
例如用AVG函数来计算平均值时，同样一段pig代码，多次计算所得的结果中，小数点的最后几位也有可能是不相同的（当然也有可能相同），大概是因为精度的原因吧。不过，一般来说小数点的最后几位已经不重要了。例如我对一个数据集进行处理后，小数点后13位才开始有不同，这样的精度完全足够了。

（12）如何编写及使用自定义函数（UDF）
请看这个链接：《Apache Pig中文教程（进阶）[http://www.codelast.com/?p=4249]》

（13）什么是聚合函数（Aggregate Function）
在pig中，聚合函数就是那些接受一个输入包（bag），返回一个标量（scalar）值的函数。COUNT函数就是一个例子。

（14）COGROUP做了什么
与GROUP操作符一样，COGROUP也是用来分组的，不同的是，COGROUP可以按多个关系中的字段进行分组。
还是以一个实例来说明，假设有以下两个数据文件：

```
[root@localhost pig]$ cat a.txt
uidk  12  3
hfd 132 99
bbN 463 231
UFD 13  10

[root@localhost pig]$ cat b.txt
908 uidk  888
345 hfd 557
28790 re  00000
```

现在我们用pig做如下操作及得到的结果为：

```
grunt> A = LOAD 'a.txt' AS (acol1:chararray, acol2:int, acol3:int);
grunt> B = LOAD 'b.txt' AS (bcol1:int, bcol2:chararray, bcol3:int);
grunt> C = COGROUP A BY acol1, B BY bcol2;
grunt> DUMP C;
(re,{},{(28790,re,0)})
(UFD,{(UFD,13,10)},{})
(bbN,{(bbN,463,231)},{})
(hfd,{(hfd,132,99)},{(345,hfd,557)})
(uidk,{(uidk,12,3)},{(908,uidk,888)})
```

每一行输出的第一项都是分组的key，第二项和第三项分别都是一个包（bag），其中，第二项是根据前面的key找到的A中的数据包，第三项是根据前面的key找到的B中的数据包。
来看看第一行输出：“re”作为group的key时，其找不到对应的A中的数据，因此第二项就是一个空的包“{}”，“re”这个key在B中找到了对应的数据（28790　re　00000），因此第三项就是包{(28790,re,0)}。
其他输出数据也类似。

（15）安装pig后，运行pig命令时提示“Cannot find hadoop configurations in classpath”等错误的解决办法
pig安装好后，运行pig命令时提示以下错误：

> *ERROR org.apache.pig.Main - ERROR 4010: Cannot find hadoop configurations in classpath (neither hadoop-site.xml nor core-site.xml was found in the classpath).If you plan to use local mode, please put -x local option in command line*

显而易见，提示找不到与hadoop相关的配置文件。所以我们需要把hadoop安装目录下的“conf”子目录添加到系统环境变量PATH中：
修改 /etc/profile 文件，添加：

```
export HADOOP_HOME=/usr/local/hadoop
export PIG_CLASSPATH=$HADOOP_HOME/conf

PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$PIG_CLASSPATH:$PATH
```

然后重新加载 /etc/profile 文件：

```
source /etc/profile
```

[http://www.codelast.com/]

（16）piggybank是什么东西

> *Pig also hosts a UDF repository called piggybank that allows users to share UDFs that they have written.*

说白了就是Apache把大家写的自定义函数放在一块儿，起了个名字，就叫做piggybank。你可以把它理解为一个SVN代码仓库。具体请看这里［https://cwiki.apache.org/confluence/display/PIG/PiggyBank］。

（17）UDF的构造函数会被调用几次
你可能会想在UDF的构造函数中做一些初始化的工作，例如创建一些文件，等等。但是你不能假设UDF的构造函数只被调用一次，因此，如果你要在构造函数中做一些只能做一次的工作，你就要当心了——可能会导致错误。

（18）LOAD数据时，如何一次LOAD多个目录下的数据
例如，我要LOAD两个HDFS目录下的数据：/abc/2010 和 /abc/2011，则我们可以这样写LOAD语句：

```
A = LOAD '/abc/201{0,1}';
```

（19）怎样自己写一个UDF中的加载函数(load function)
请看这个链接：《Apache Pig中文教程（进阶）［http://www.codelast.com/?p=4249］》

（20）重载(overloading)一个UDF
请看这个链接：《Apache Pig中文教程（进阶）［http://www.codelast.com/?p=4249］》。

（21）pig运行不起来，提示“org.apache.hadoop.ipc.Client – Retrying connect to server:
请看这个链接：《Apache Pig中文教程（进阶）［http://www.codelast.com/?p=4249］》

（22）用含有null的字段来GROUP，结果会如何
假设有数据文件 a.txt 内容为：

```
1 2 5
1   3
1 3
6 9 8
```

其中，每两列数据之间是用tab分割的，第二行的第2列、第三行的第3列没有内容（也就是说，加载到Pig里之后，对应的数据会变成null），如果把这些数据按第1、第2列来GROUP的话，第1、2列中含有null的行会被忽略吗？
来做一下试验：

```
A = LOAD 'a.txt' AS (col1:int, col2:int, col3:int);
B = GROUP A BY (col1, col2);
DUMP B;
```

输出结果为：

```
((1,2),{(1,2,5)})
((1,3),{(1,3,)})
((1,),{(1,,3)})
((6,9),{(6,9,8)})
```

从上面的结果（第三行）可见，原数据中第1、2列里含有null的行也被计入在内了，也就是说，GROUP操作是不会忽略null的，这与COUNT有所不同（见本文前面的部分）。

（23）如何统计数据中某些字段的组合有多少种
假设有如下数据：

```
[root@localhost]# cat a.txt
1 3 4 7
1 3 5 4
2 7 0 5
9 8 6 6
```

现在我们要统计第1、2列的不同组合有多少种，对本例来说，组合有三种：

```
1 3
2 7
9 8
```

也就是说我们要的答案是3。
用Pig怎么计算？

[http://www.codelast.com/]

先写出全部的Pig代码：

```
A = LOAD 'a.txt' AS (col1:int, col2:int, col3:int, col4:int);
B = GROUP A BY (col1, col2);
C = GROUP B ALL;
D = FOREACH C GENERATE COUNT(B);
DUMP D;
```

然后再来看看这些代码是如何计算出上面的结果的：
①第一行代码加载数据，没什么好说的。
②第二行代码，得到第1、2列数据的所有组合。B的数据结构为：

```
grunt> DESCRIBE B;
B: {group: (col1: int,col2: int),A: {col1: int,col2: int,col3: int,col4: int}}
```

把B DUMP出来，得到：

```
((1,3),{(1,3,4,7),(1,3,5,4)})
((2,7),{(2,7,0,5)})
((9,8),{(9,8,6,6)})
```

非常明显，(1,3)，(2,7)，(9,8)的所有组合已经被排列出来了，这里得到了若干行数据。下一步我们要做的就是统计这样的数据一共有多少行，也就得到了第1、2列的组合有多少组。
③第三和第四行代码，就实现了统计数据行数的功能。参考本文前面部分的“怎样统计数据行数”一节。就明白这两句代码是什么意思了。
这里需要特别说明的是：
a)为什么倒数第二句代码中是COUNT(B)，而不是COUNT(group)？
我们是对C进行FOREACH，所以要先看看C的数据结构：

```
grunt> DESCRIBE C;
C: {group: chararray,B: {group: (col1: int,col2: int),A: {col1: int,col2: int,col3: int,col4: int}}}
```

可见，你可以把C想像成一个map的结构，key是一个group，value是一个包（bag），它的名字是B，这个包中有N个元素，每一个元素都对应到②中所说的一行。根据②的分析，我们就是要统计B中元素的个数，因此，这里当然就是COUNT(B)了。
b)COUNT函数的作用是统计一个包（bag）中的元素的个数：

> *COUNT*
> *Computes the number of elements in a bag.*

从C的数据结构看，B是一个bag，所以COUNT函数是可以用于它的。
如果你试图把COUNT应用于一个非bag的数据结构上，会发生错误，例如：

```
java.lang.ClassCastException: org.apache.pig.data.BinSedesTuple cannot be cast to org.apache.pig.data.DataBag
```

这是把Tuple传给COUNT函数时发生的错误。

(24) 两个整型数相除，如何转换为浮点型，从而得到正确的结果
这个问题其实很傻，或许不用说你也知道了：假设有int a = 3 和 int b = 2两个数，在大多数编程语言里，a/b得到的是1，想得到正确结果1.5的话，需要转换为float再计算。在Pig中其实和这种情况一样，下面就拿几行数据来做个实验：

```
[root@localhost ~]# cat a.txt
3 2
4 5
```

在Pig中：

```
grunt> A = LOAD 'a.txt' AS (col1:int, col2:int);
grunt> B = FOREACH A GENERATE col1/col2;
grunt> DUMP B;
(1)
(0)
```

可见，不加类型转换的计算结果是取整之后的值。
那么，转换一下试试：

```
grunt> A = LOAD 'a.txt' AS (col1:int, col2:int);
grunt> B = FOREACH A GENERATE (float)(col1/col2);
grunt> DUMP B;
(1.0)
(0.0)
```

这样转换还是不行的，这与大多数编程语言的结果一致——它只是把取整之后的数再转换为浮点数，因此当然是不行的。

正确的做法应该是：

```
grunt> A = LOAD 'a.txt' AS (col1:int, col2:int);
grunt> B = FOREACH A GENERATE (float)col1/col2;
grunt> DUMP B;
(1.5)
(0.8)
```

或者这样也行：

```
grunt> A = LOAD 'a.txt' AS (col1:int, col2:int);
grunt> B = FOREACH A GENERATE col1/(float)col2;
grunt> DUMP B;
(1.5)
(0.8)
```

这与我们的通常做法是一致的，因此，你要做除法运算的时候，需要注意这一点。

(25) UNION的一个例子
假设有两个数据文件为：

```
[root@localhost ~]# cat 1.txt
0 3
1 5
0 8

[root@localhost ~]# cat 2.txt
1 6
0 9
```

现在要求出：在第一列相同的情况下，第二列的和分别为多少？
例如，第一列为 1 的时候，第二列有5和6两个值，和为11。同理，第一列为0的时候，第二列的和为 3+ 8+ 9=20。
计算此问题的Pig代码如下：

```
A = LOAD '1.txt' AS (a: int, b: int);
B = LOAD '2.txt' AS (c: int, d: int);
C = UNION A, B;
D = GROUP C BY $0;
E = FOREACH D GENERATE FLATTEN(group), SUM(C.$1);
DUMP E;
```

输出为：

```
(0,20)
(1,11)
```

[http://www.codelast.com/]
我们来看看每一步分别做了什么：
①第1行、第2行代码分别加载数据到关系A、B中，没什么好说的。
②第3行代码，将关系A、B合并起来了。合并后的数据结构为：

```
grunt> DESCRIBE C;
C: {a: int,b: int}
```

其数据为：

```
grunt> DUMP C;
(0,3)
(1,5)
(0,8)
(1,6)
(0,9)
```

③第4行代码按第1列（即$0）进行分组，分组后的数据结构为：

```
grunt> DESCRIBE D;
D: {group: int,C: {a: int,b: int}}
```

其数据为：

```
grunt> DUMP D;
(0,{(0,9),(0,3),(0,8)})
(1,{(1,5),(1,6)})
```

④最后一行代码，遍历D，将D中每一行里的所有bag(即C)的第2列(即$1)进行累加，就得到了我们要的结果。

(26) 错误“ERROR org.apache.pig.tools.grunt.Grunt - ERROR 2042: Error in new logical plan. Try -Dpig.usenewlogicalplan=false.”的可能原因
①Pig的bug，详见此链接 [https://issues.apache.org/jira/browse/PIG-1683] ；
②其他原因。我遇到并解决了一例。具体的代码不便在此陈列，但是基本可以说是由于自己写的Pig代码对复杂数据结构的处理不当导致的，后来我尝试更改了一种实现方式，就绕过了这个问题。关于这点，确实还是要具体问题具体分析的，在这里没有实例的话，无法给大家一个明确的解决问题的指南。

(27) 如何在Pig中使用正则表达式对字符串进行匹配
假设你有如下数据文件：

```
[root@localhost ~]# cat a.txt
1 http://ui.qq.com/abcd.html
5 http://tr.qq.com/743.html
8 http://vid.163.com/trees.php
9 http:auto.qq.com/us.php
```

现在要找出该文件中，第二列符合“*//*.qq.com/*”模式的所有行（此处只有前两行符合条件），怎么做？
Pig代码如下：

```
A = LOAD 'a.txt' AS (col1: int, col2: chararray);
B = FILTER A BY col2 matches '.*//.*\\.qq\\.com/.*';
DUMP B;
```

我们看到，matches关键字对 col2 进行了正则匹配，它使用的是Java格式的正则表达式匹配规则。
. 表示任意字符，* 表示字符出现任意次数；₩. 对 . 进行了转义，表示匹配 . 这个字符；/ 就是表示匹配 / 这个字符。
这里需要注意的是，在引号中，用于转义的字符 ₩ 需要打两个才能表示一个，所以上面的 ₩₩. 就是与正则中的 ₩. 是一样的，即匹配 . 这个字符。所以，如果你要匹配数字的话，应该用这种写法（₩d表示匹配数字，在引号中必须用₩₩d）：

```
B = FILTER A BY (col matches '\\d.*');
```

[http://www.codelast.com/]

最后输出结果为：

```
(1,http://ui.qq.com/abcd.html)
(5,http://tr.qq.com/743.html)
```

可见结果是正确的。

(28) 如何截取一个字符串中的某一段
在处理数据时，如果你想提取出一个日期字符串的年份，例如提取出“2011-10-26”中的“2011”，可以用内置函数 SUBSTRING 来实现：

> *SUBSTRING*
> *Returns a substring from a given string.*
> *Syntax*
> *SUBSTRING(string, startIndex, stopIndex)*

下面举一个例子。假设有数据文件：

```
[root@localhost ~]# cat a.txt
2010-05-06  abc
2008-06-18  uio
2011-10-11  tyr
2010-12-23  fgh
2011-01-05  vbn
```

第一列是日期，现在要找出所有不重复的年份有哪些，可以这样做：

```
A = LOAD 'a.txt' AS (dateStr: chararray, flag: chararray);
B = FOREACH A GENERATE SUBSTRING(dateStr, 0, 4);
C = DISTINCT B;
DUMP C;
```

输出结果为：

```
(2008)
(2010)
(2011)
```

可见达到了我们想要的效果。
上面的代码太简单了，不必多言，唯一需要说明一下的是 SUBSTRING 函数，它的第一个参数是要截取的字符串，第二个参数是起始索引（从0开始），第三个参数是结束索引。

[http://www.codelast.com/]

(29) 如何拼接两个字符串
假设有以下数据文件：

```
[root@localhost ~]# cat 1.txt
abc 123
cde 456
fgh 789
ijk 200
```

现在要把第一列和第二列作为字符串拼接起来，例如第一行会变成“abc123”，那么使用CONCAT这个求值函数（eval function）就可以做到：

```
A = LOAD '1.txt' AS (col1: chararray, col2: int);
B = FOREACH A GENERATE CONCAT(col1, (chararray)col2);
DUMP B;
```

输出结果为：

```
(abc123)
(cde456)
(fgh789)
(ijk200)
```

注意这里故意在加载数据的时候把第二列指定为int类型，这是为了说明数据类型不一致的时候CONCAT会出错（你可以试验一下）：

> *ERROR org.apache.pig.tools.grunt.Grunt - ERROR 1045: Could not infer the matching function for org.apache.pig.builtin.CONCAT as multiple or none of them fit. Please use an explicit cast.*

所以在后面CONCAT的时候，对第二列进行了类型转换。
另外，如果数据文件内容为：

```
[root@localhost ~]# cat 1.txt
5 123
7 456
8 789
0 200
```

那么，如果对两列整数CONCAT：

```
A = LOAD '1.txt' AS (col1: int, col2: int);
B = FOREACH A GENERATE CONCAT(col1, col2);
```

同样也会出错：

> *ERROR org.apache.pig.tools.grunt.Grunt - ERROR 1045: Could not infer the matching function for org.apache.pig.builtin.CONCAT as multiple or none of them fit. Please use an explicit cast.*

要注意这一点。
有人可能会问：要拼接几个字符串的话怎么办？CONCAT 套 CONCAT 就要可以了（有点笨，但管用）：CONCAT(a, CONCAT(b, c))

(30) 如何求两个数据集的重合 & 不同的数据类型JOIN会失败
假设有以下两个数据文件：

```
[root@localhost ~]# cat 1.txt
123
456
789
200
```

以及：

```
[root@localhost ~]# cat 2.txt
200
333
789
```

现在要找出两个文件中，相同的数据有多少行，怎么做？这也就是所谓的求两个数据集的重合。
用关系操作符JOIN，我们可以达到这个目的。在处理海量数据时，经常会有求重合的需求。所以JOIN是Pig中一个极其重要的操作。
在本例中，两个文件中有两个相同的数据行：789以及200，因此，结果应该是2。
我们先来看看正确的代码：

```
A = LOAD '1.txt' AS (a: int);
B = LOAD '2.txt' AS (b: int);
C = JOIN A BY a, B BY b;
D = GROUP C ALL;
E = FOREACH D GENERATE COUNT(C);
DUMP E;
```

解释一下：
①第一、二行是加载数据，不必多言。
②第三行按A的第1列、B的第二列进行“结合”，JOIN之后，a、b两列不相同的数据就被剔除掉了。C的数据结构为：

```
C: {A::a: int,B::b: int}
```

C的数据为：

```
(200,200)
(789,789)
```

③由于我们要统计的是数据行数，所以上面的Pig代码中的第4、5行就进行了计数的运算。
④如果文件 2.txt 多一行数据“200”，结果会是什么？答案是：结果为3行。这个时候C的数据为：

```
(200,200)
(200,200)
(789,789)
```

所以如果你要去除重复的，还需要用DISTINCE对C处理一下：

```
A = LOAD '1.txt' AS (a: int);
B = LOAD '2.txt' AS (b: int);
C = JOIN A BY a, B BY b;
uniq_C = DISTINCT C;
D = GROUP uniq_C ALL;
E = FOREACH D GENERATE COUNT(uniq_C);
DUMP E;
```

这样得到的结果就是2了。

[http://www.codelast.com/]

尤其需要注意的是，如果JOIN的两列具有不同的数据类型，是会失败的。例如以下代码：

```
A = LOAD '1.txt' AS (a: int);
B = LOAD '2.txt' AS (b: chararray);
C = JOIN A BY a, B BY b;
D = GROUP C ALL;
E = FOREACH D GENERATE COUNT(C);
DUMP E;
```

在语法上是没有错误的，但是一运行就会报错：

ERROR org.apache.pig.tools.grunt.Grunt - ERROR 1107: Cannot merge join keys, incompatible types

这是因为a、b具有不同的类型：int和chararray。

(31) 使用三目运算符“？：”有时候必须加括号
假设有以下数据文件：

```
[root@localhost ~]# cat a.txt
5 8 9
6   0
4 3 1
```

其中，第二行的第二列数据是有缺失的，因此，加载数据之后，它会成为null。顺便废话一句，在处理海量数据时，数据有缺失是经常遇到的现象。
现在，我们如果要把所有缺失的数据填为 -1， 可以使用三目运算符来操作：

```
A = LOAD 'a.txt' AS (col1:int, col2:int, col3:int);
B = FOREACH A GENERATE col1, ((col2 is null)? -1 : col2), col3;
DUMP B;
```

输出结果为：

```
(5,8,9)
(6,-1,0)
(4,3,1)
```

((col2 is null)? -1 : col2) 的含义不用解释你也知道，就是当col2为null的时候将其置为-1，否则就保持原来的值，但是注意，它最外面是用括号括起来的，如果去掉括号，写成 (col2 is null)? -1 : col2，那么就会有语法错误：

ERROR org.apache.pig.tools.grunt.Grunt - ERROR 1000: Error during parsing. Encountered " "is" "is "" at line 1, column 36.
Was expecting one of （后面省略）

错误提示有点不直观。所以，有时候使用三目运算符是必须要使用括号的。

(32) 如何补上缺失的数据
通过前面的文章，我们已经知道了如何按自己的需求补上缺失的数据，那么这里还有一个例子，可以让你多了解一些特殊的情况。
数据文件如下：

```
[root@localhost ~]# cat 1.txt
1 (4,9)
5
8 (3,0)
5 (9,2)
6
```

这些数据的布局比较怪，我们要把它加载成什么样的schema呢？答：第一列为一个int，第二列为一个tuple，此tuple又含两个int。
加载成这样的模式不是为了制造复杂度，而是为了说明后面的问题而设计的。
同时，我们也注意到，第二列数据是有缺失的。
问题：怎样求在第一列数据相同的情况下，第二列数据中的第一个整数的和分别为多少？
例如，第一列为1的数据只有一行（即第一行），因此，第二列的第一个整数的和就是4。
但是对最后一行，也就是第一列为6时，由于其第二列数据缺失，我们希望它输出的结果是0。
先来看看Pig代码：

```
A = LOAD '1.txt' AS (a:int, b:tuple(x:int, y:int));
B = FOREACH A GENERATE a, FLATTEN(b);
C = GROUP B BY a;
D = FOREACH C GENERATE group, SUM(B.x);
DUMP D;
```

结果为：

```
(1,4)
(5,9)
(6,)
(8,3)
```

我们注意到，(5,9) 这一行是由数据文件 1.txt 的第 2、4行计算得到的，其中，第2行数据有缺失，但这并不影响求和计算，因为另一行数据没有缺失。你可以这样想：一个包（bag）中有多个数，当其中一个为null，而其他不为null时，把它们相加会自动忽略null。
然而，第三行 (6,) 是不是太刺眼了？没错，因为数据文件 1.txt 的最后一行缺失了第二列，所以，在 SUM(B.x) 中的 B.x 为null就

会导致计算结果为null，从而什么也输出不了。
这就与我们期望的输出有点不同了。我们希望这种缺失的数据不要空着，而是输出0。该怎么做呢？
[http://www.codelast.com/]
想法1：

```
D = FOREACH C GENERATE group, ((IsEmpty(B.x)) ? 0 : SUM(B.x));
```

输出结果为：

```
(1,4)
(5,9)
(6,)
(8,3)
```

可见行不通。从这个结果我们知道，IsEmpty(B.x) 为false，即B.x不是empty的，所以不能这样做。
想法2：

```
D = FOREACH C GENERATE group, ((B.x is null) ? 0 : SUM(B.x));
```

输出结果还是与上面一样！仍然行不通。这更奇怪了：B.x既非empty，也非null，那么它是什么情况？按照我的理解，当group为6时，它应该是一个非空的包（bag），里面有一个null的东西，所以，这个包不是empty的，它也非null。我不知道这样理解是否正确，但是它看上去就像是这样的。
想法3：

```
D = FOREACH C GENERATE group, SUM(B.x) AS s;
E = FOREACH D GENERATE group, ((s is null) ? -1 : s);
DUMP E;
```

输出结果为：

```
(1,4)
(5,9)
(6,-1)
(8,3)
```

可见达到了我们想要的结果。这与本文前面部分的做法是一致的，即：先得到含null的结果，再把这个结果中的null替换为指定的值。
有人会问：就没有办法在生成数据集D的时候，就直接通过判断语句来实现这个效果吗？据我目前所知是不行的，如果哪位读者知道，不妨告知。

(33) DISTINCT操作用于去重，正因为它要把数据集合到一起，才知道哪些数据是重复的，因此，它会产生reduce过程。同时，在map阶段，它也会利用combiner来先去除一部分重复数据以加快处理速度。

(34) 如何将Pig job的优先级设为HIGH
嫌Pig job运行太慢？只需在Pig脚本的开头加上一句：

```
set job.priority HIGH;
```

即可将Pig job的优先级设为高了。

(35) “Scalars can be only used with projections”错误的原因
这个错误提示比较不直观，光看这句话是不容易发现错误所在的，但是，只要你一Google，可能就找到原因了，例如这个链接[https://issues.apache.org/jira/browse/PIG-1788]里的反馈。
在这里，我也想用一个简单的例子给大家用演示一下产生这个错误的原因之一。
假设有如下数据文件：

```
[root@localhost ~]$ cat 1.txt
a 1
b 8
c 3
c 3
d 6
d 3
c 5
e 7
```

现在要统计：在第1列的每一种组合下，第二列为3和6的数据分别有多少条？
例如，当第1列为 c 时，第二列为3的数据有2条，为6的数据有0条；当第1列为d时，第二列为3的数据有1条，为6的数据有1条。其他的依此类推。
Pig代码如下：

```
A = LOAD '1.txt' AS (col1:chararray, col2:int);
B = GROUP A BY col1;
C = FOREACH B {
  D = FILTER A BY col2 == 3;
  E = FILTER A BY col2 == 6;
  GENERATE group, COUNT(D), COUNT(E);
};
DUMP C;
```

输出结果为：

```
(a,0,0)
(b,0,0)
(c,2,0)
(d,1,1)
(e,0,0)
```

可见结果是正确的。

[http://www.codelast.com/]

那么，如果我在上面的代码中，把“D = FILTER A BY col2 == 3”不小心写成了“D = FILTER B BY col2 == 3”，就肯定会得到“Scalars can be only used with projections”的错误提示。
说白了，还是要时刻注意你每一步生成的数据的结构，眼睛睁大，千万不要用错了relation。

(36)什么是嵌套的FOREACH/内部的FOREACH
嵌套的（nested）FOREACH和内部的（inner）FOREACH是一个意思，正如你在本文第(35)条中所见，一个FOREACH可以对每一条记录施以多种不同的关系操作，然后再GENERATE得到想要的结果，这就是嵌套的/内部的FOREACH。

(37)错误“Could not infer the matching function for org.apache.pig.builtin.CONCAT”的原因之一
如果你遇到这个错误，那么有可能是你在多级CONCAT嵌套的时候，没有写对语句，例如“CONCAT(CONCAT(CONCAT(a, b), c), d)”这样的嵌套，由于括号众多，所以写错了是一点也不奇怪的。我遇这个错误的时候，是由于CONCAT太多，自己多写了一个都没有发现。希望我的提醒能给你一点解决问题的提示。

(38)用Pig加载HBase数据时遇到的错误“ERROR 2999: Unexpected internal error. could not instantiate 'com.twitter.elephantbird.pig.load.HBaseLoader' with arguments XXX”的原因之一
请看这个链接：《Apache Pig中文教程（进阶） [http://www.codelast.com/?p=4249]》

(39)错误“ERROR 1039: In alias XX, incompatible types in EqualTo Operator left hand side:XXX right hand side:XXX”的原因
其实这个错误提示太明显了，就是类型不匹配造成的。上面的XXX可以指代不同的类型。
这说明，前面可能有一个类型为long的字段，后面你却把它当chararray来用了，例如：

```
A = LOAD '1.txt' AS (col1: int, col2: long);
B = FILTER A BY col2 == '123456789';
C = GROUP B ALL;
D = FOREACH C GENERATE COUNT(B);
DUMP D;
```

就会出错：

ERROR 1039: In alias B, incompatible types in EqualTo Operator left hand side:long right hand side:chararray

只要把col2强制类型转换一下（或者一开始就将其类型指定为chararray）就可以解决问题。

[http://www.codelast.com/]

不仅在进行数据比较中，在JOIN时也经常出现数据类型不匹配导致的错误问题。我在实际工作中发现，有的同学写了比较长的Pig代码，出现了这样的错误却不会仔细去看错误提示，而是绞尽脑汁地逐句去检查语法（语法是没有错的），结果费了很大的劲才知道是类型问题，得不偿失，还不如仔细看错误提示想想为什么。

(40)在grunt交互模式下，如何在编辑Pig代码的时候跳到行首和行末/行尾
在grunt模式下，如果你写了一句超长的Pig代码，那么，你想通过HOME/END键跳到行首和行末是做不到的。
按HOME时，Pig会在你的光标处插入一个“1~”：

```
A = LOAD '1.txt' AS (col: int1~);
```

按END时，Pig会在你的光标处插入一个“4~”：

```
A = LOAD '1.txt' AS (col: int4~);
```

正确的做法是：按Ctrl+A 和 Ctrl+E 代替 HOME 和 END，就可以跳到行首和行末了。

(41)不能对同一个关系（relation）进行JOIN
假设有如下文件：

```
[root@localhost ~]# cat 1.txt
1 a
2 e
3 v
4 n
```

我想对第一列这样JOIN：

```
A = LOAD '1.txt' AS (col1: int, col2: chararray);
B = JOIN A BY col1, A BY col1;
```

那么当你试图 DUMP B 的时候，会报如下的错：

ERROR org.apache.pig.tools.grunt.Grunt - ERROR 1108: Duplicate schema alias: A::col1 in "B"

这是因为Pig会弄不清JOIN之后的字段名——两个字段均为A::col1，使得一个关系（relation）中出现了重复的名字，这是不允许的。

[http://www.codelast.com/]

要解决这个问题，只需将数据LOAD两次，并且给它们起不同的名字就可以了：

```
grunt> A = LOAD '1.txt' AS (col1: int, col2: chararray);
grunt> B = LOAD '1.txt' AS (col1: int, col2: chararray);
grunt> C = JOIN A BY col1, B BY col1;
grunt> DESCRIBE C;
C: {A::col1: int,A::col2: chararray,B::col1: int,B::col2: chararray}
grunt> DUMP C;
(1,a,1,a)
(2,e,2,e)
(3,v,3,v)
(4,n,4,n)
```

从上面的 C 的schema，你可以看出来，如果对同一个关系A的第一列进行JOIN，会导致schema中出现相同的字段名，所以当然会出错。

(42) 外部的JOIN(outer JOIN)

初次使用JOIN时，一般人使用的都是所谓的“内部的JOIN”(inner JOIN)，也即类似于 C = JOIN A BY col1, B BY col2 这样的JOIN。Pig也支持“外部的JOIN”(outer JOIN)，下面就举一个例子。

假设有文件：

```
[root@localhost ~]# cat 1.txt
1 a
2 e
3 v
4 n
```

以及：

```
[root@localhost ~]# cat 2.txt
9 a
2 e
3 v
0 n
```

现在来对这两个文件的第一列作一个outer JOIN：

```
grunt> A = LOAD '1.txt' AS (col1: int, col2: chararray);
grunt> B = LOAD '2.txt' AS (col1: int, col2: chararray);
grunt> C = JOIN A BY col1 LEFT OUTER, B BY col1;
grunt> DESCRIBE C;
C: {A::col1: int,A::col2: chararray,B::col1: int,B::col2: chararray}
grunt> DUMP C;
(1,a,,)
(2,e,2,e)
(3,v,3,v)
(4,n,,)
```

在outer JOIN中，“OUTER”关键字是可以省略的。从上面的结果，我们注意到：如果换成一个inner JOIN，则两个输入文件的第一、第二行都不会出现在结果中（因为它们的第一列不相同），而在LEFT OUTER JOIN中，文件1.txt的记录却被保存了下来，所以这就是LEFT OUTER JOIN的特点：对左边的记录来说，即使它与右边的记录不匹配，它也会被包含在输出数据中。

[http://www.codelast.com/]

同理可知RIGHT OUTER JOIN的功能——把上面的 LEFT 换成 RIGHT，结果如下：

```
(,,0,n)
(2,e,2,e)
(3,v,3,v)
(,,9,a)
```

可见，与左边的记录不匹配的右边的记录被保存了下来，而左边的记录没有保存下来（两个逗号表明其为空），这就是RIGHT OUTER JOIN的效果，与我们想像的一样。

有人会问，OUTER JOIN在实际中可以用来做什么？举一个例子：可以用来求“不在某数据集中的那些数据（即：不重合的数据）”。还是以上面的两个数据文件为例，现在我要求出 1.txt 中，第一列不在 2.txt 中的第一列的那些记录，肉眼一看就知道，1和4这两个数字在 2.txt 的第一列里没有出现，而2和3出现了，因此，我们要找的记录就是：

```
1 a
4 n
```

要实现这个效果，Pig代码及结果为：

```
grunt> A = LOAD '1.txt' AS (col1: int, col2: chararray);
grunt> B = LOAD '2.txt' AS (col1: int, col2: chararray);
grunt> C = JOIN A BY col1 LEFT OUTER, B BY col1;
grunt> DESCRIBE C;
C: {A::col1: int,A::col2: chararray,B::col1: int,B::col2: chararray}
grunt> D = FILTER C BY (B::col1 is null);
grunt> E = FOREACH D GENERATE A::col1 AS col1, A::col2 AS col2;
grunt> DUMP E;
(1,a)
(4,n)
```

可见，我们确实找出了“不重合的记录”。在作海量数据分析时，这种功能是极为有用的。
最后来一个总结：
假设有两个数据集（在1.txt和2.txt中），分别都只有1列，则如下代码：

```
A = LOAD '1.txt' AS (col1: chararray);
B = LOAD '2.txt' AS (col1: chararray);
C = JOIN A BY col1 LEFT OUTER, B BY col1;
D = FILTER C BY (B::col1 is null);
E = FOREACH D GENERATE A::col1 AS col1;
DUMP E;
```

计算结果为：在A中，但不在B中的记录。

(43) JOIN的优化
请看这个链接：《Apache Pig中文教程（进阶） [http://www.codelast.com/?p=4249] 》

(44) GROUP时按所有字段分组可以用GROUP ALL吗
假设你有如下数据文件：

```
[root@localhost ~]# cat 3.txt
1 9
2 2
3 3
4 0
1 9
1 9
4 0
```

现在要找出第1、2列的组合中，每一种的个数分别为多少，例如，(1,9)组合有3个，(4,0)组合有两个，依此类推。
显而易见，我们只需要用GROUP就可以轻易完成这个任务。于是写出如下代码：

```
A = LOAD '3.txt' AS (col1: int, col2: int);
B = GROUP A ALL;
C = FOREACH B GENERATE group, COUNT(A);
DUMP C;
```

可惜，结果不是我们想要的：

```
(all,7)
```

为什么呢？我们的本意是按所有列来GROUP，于是使用了GROUP ALL，但是这实际上变成了统计行数，下面的代码就是一段标准的统计数据行数的代码：

```
A = LOAD '3.txt' AS (col1: int, col2: int);
B = GROUP A ALL;
C = FOREACH B GENERATE COUNT(A);
DUMP C;
```

因此，上面的 C = FOREACH B GENERATE group, COUNT(A) 也无非就是多打印了一个group的名字（all）而已——group的名字被设置为“all”，这是Pig帮你做的。

[http://www.codelast.com/]

正确的做法很简单，只需要按所有字段GROUP，就可以了：

```
A = LOAD '3.txt' AS (col1: int, col2: int);
B = GROUP A BY (col1, col2);
C = FOREACH B GENERATE group, COUNT(A);
DUMP C;
```

结果如下：

```
((1,9),3)
((2,2),1)
((3,3),1)
((4,0),2)
```

这与我们前面分析的正确结果是一样的。

(45) 在Pig中使用中文字符串
有读者来信问我，如何在Pig中使用中文作为FILTER的条件？我做了如下测试，结论是可以使用中文。
数据文件 data.txt 内容为（每一列之间以TAB为分隔符）：

```
1 北京市 a
2 上海市 b
3 北京市 c
4 北京市 f
5 天津市 e
```

Pig脚本文件 test.pig 内容为：

```
A = LOAD 'data.txt' AS (col1: int, col2: chararray, col3: chararray);
B = FILTER A BY (col2 == '北京市');
DUMP B;
```

首先，我这两个文件的编码都是UTF-8(无BOM)，在Linux命令行下，我直接以本地模式执行Pig脚本 test.pig：

```
pig -x local test.pig
```

得到的输出结果为：

```
(1,北京市,a)
(3,北京市,c)
(4,北京市,f)
```

可见结果是正确的。

[http://www.codelast.com/]

但是，如果我在grunt交互模式下，把 test.pig 的内容粘贴进去执行，是得不到任何输出结果的：

```
grunt> A = LOAD 'data.txt' AS (col1: int, col2: chararray, col3: chararray);
grunt> B = FILTER A BY (col2 == '北京市');
grunt> DUMP B;
```

具体原因我不清楚，但是至少有一点是肯定的：可以使用中文作为FILTER的条件，只要不在交互模式下执行你的Pig脚本即可。

（46）如何统计 tuple 中的 field 数，bag 中的 tuple 数，map 中的 key/value 组数
一句话：用Pig内建的 SIZE 函数：

> *Computes the number of elements based on any Pig data type.*

具体可看这个 [http://pig.apache.org/docs/r0.8.1/piglatin_ref2.html#SIZE] 链接。

（47）一个字符串为null，与它为空不一定等价
在某些情况下，要获取“不为空”的字符串，仅仅用 is not null 来判断是不够的，还应该加上 SIZE(field_name) > 0 的条件：

```
B = FILTER A BY (field_name is not null AND (SIZE(field_name) > 0));
```

注意，这只是在某些情况下需要这样做，在一般情况下，仅用 is not null 来过滤就可以了。我并没有总结出特殊情况是哪些情况，我只能说我遇到了一例，所以才有了这一个结论。

（48）Pig中的各operator（操作符），哪些会触发reduce过程
①GROUP：由于GROUP操作会将所有具有相同key的记录收集到一起，所以数据如果正在map中处理的话，就会触发shuffle→reduce的过程。
②ORDER：由于需要将所有相等的记录收集到一起（才能排序），所以ORDER会触发reduce过程。同时，除了你写的那个Pig job之外，Pig还会添加一个额外的M-R job到你的数据流程中，因为Pig需要对你的数据集做采样，以确定数据的分布情况，从而解决数据分布严重不均的情况下job效率过于低下的问题。
③DISTINCT：由于需要将记录收集到一起，才能确定它们是不是重复的，因此DISTINCT会触发reduce过程。当然，DISTINCT也会利用combiner在map阶段就把重复的记录移除。
④JOIN：JOIN用于求重合，由于求重合的时候，需要将具有相同key的记录收集到一起，因此，JOIN会触发reduce过程。
⑤LIMIT：由于需要将记录收集到一起，才能统计出它返回的条数，因此，LIMIT会触发reduce过程。
⑥COGROUP：与GROUP类似（参看本文前面的部分），因此它会触发reduce过程。
⑦CROSS：计算两个或多个关系的叉积。

（49）如何统计一个字符串中包含的指定字符数
这可以不算是个Pig的问题了，你可以把它认为是一个shell的问题。从本文前面部分我们已经知道，Pig中可以用 STREAM … THROUGH 来调用shell进行辅助数据处理，所以在这我们也能这样干。
假设有文本文件：

```
[root@localhost ~]$ cat 1.txt
123 abcdef:243789174
456 DFJKSDFJ:3646:555558888
789 yKDSF:00000%0999:2343324:11111:33333
```

现在要统计：每一行中，第二列里所包含的冒号（“:”）分别为多少？代码如下：

```
A = LOAD '1.txt' AS (col1: chararray, col2: chararray);
B = STREAM A THROUGH `awk -F":" '{print NF-1}'` AS (colon_count: int);
DUMP B;
```

结果为：

```
(1)
(2)
(4)
```

[http://www.codelast.com/]

（50）UDF是区分大小写的
因为UDF是由Java类来实现的，所以区分大小写，就这么简单。

- 上一篇: [原创]使用C++（通过Thrift）访问/操作/读写Hbase
- 下一篇: [原创]一些未归类的命令、操作方法或问题总结（3）

发表在Linux，原创
Tags: apache pig Cannot find hadoop configurations in classpath Could not infer the matching function for org.apache.pig.builtin.CONCAT could not instantiate 'com.twitter.elephantbird.pig.load.HBaseLoader' Error in new logical plan getSchema incompatible types in EqualTo Operator left hand side inner bag inner foreach inner

JOIN JOIN优化 load function localhost/127.0.0.1:9000. Already tried nested foreach org.apache.hadoop.ipc.Client - Retrying connect to server outer JOIN pig tutorial in Chinese PigStorage pig中文基础概念 pig中文教程 pig使用 Pig加载HBase数据 pig拼接字符串 pig无法启动 Pig正则匹配 pig语法高亮 Scalars can be only used with projections Try -Dpig.usenewlogicalplan=false UDF user-defined function 中文 中文教程 元组 内部的FOREACH 内部的JOIN 基础 外部的JOIN 嵌套的FOREACH 手册 概念 聚合函数 自定义函数 重载UDF

若要跟踪这篇文章的任何更新，你可以使用 RSS 2.0 Feed. 你可以直接转到文章底部进行评论，Pinging目前已关闭

6 条评论

添加你的评论

learnhard 2012 年 03 月 22 日 15:21

to tewilove：
问题1：还是那个回答，我没有load过HBase的数据，所以无法回答你。

问题2：我的方法比较笨，两步走——如果你的每一行都是“((a,1),(b,1),(c,3))”这样的字符串（最前端包含两个左括号，最右端包含两个右括号），那么，我是先把每一行拆成N行类似于“c,3”这样的字符串，最后得到类似于这样的N行：
a,1
b,2
a,1
b,1
c,3
c,2
STORE到磁盘上之后，再用写第二个Pig脚本来统计。由于被拆成了N行最简单的形式，所以统计起来就非常容易了。
拆成N行我是用了正则替换的方法：
A = LOAD ‘data.txt’ AS (col: chararray);
A1 = FOREACH A GENERATE REPLACE(col, ‘₩₩(₩₩(‘, ‘₩₩(‘) AS col;
A2 = FOREACH A1 GENERATE REPLACE(col, ‘₩₩)₩₩)’, ‘₩₩)’) AS col;

B = FOREACH A2 GENERATE REPLACE(col, ‘₩₩),₩₩(‘, ‘₩₩)₩n₩₩(‘) AS col;
C = FOREACH B GENERATE REPLACE(col, ‘₩₩(‘, ”) AS col;
D = FOREACH C GENERATE REPLACE(col, ‘₩₩)’, ”) AS col;
STORE D INTO ‘/home/abc/temp’;

第二个job：
A = LOAD ‘/home/abc/temp’ AS (col1: chararray, col2: int) USING PigStorage(‘,’);
B = GROUP A BY col1;
C = FOREACH B GENERATE FLATTEN(group), COUNT(A);
DUMP C;

回复

tewilove 2012 年 03 月 22 日 18:41

谢谢，我试试看。第一个的错误原因是分隔符是空格，逗号会算进变量名。第二个简化点说就是怎么把一个chararray按规则展开成bag，似乎可以写UDF看看。

回复

tewilove 2012 年 03 月 21 日 13:51

数据格式为((a1,b1),(a2,b2),…)，数量不确定，位于一个HBase的column内，这样的schema怎么定义呢。

回复

learnhard 2012 年 03 月 21 日 14:22

找一个HBase的loader试试吧，我没有用Pig从HBase中加载过数据，所以在这里无法具体回答你

回复

tewilove 2012 年 03 月 22 日 10:33

两个小问题：
1.
A = LOAD ‘test_logs’ using org.apache.pig.backend.hadoop.hbase.HBaseStorage(‘tag:key, ‘) as (o:chararray);
正常。
A = LOAD ‘test_logs’ using org.apache.pig.backend.hadoop.hbase.HBaseStorage(‘tag:key, response:code’) as (o:chararray, c:int);
缺失key这一列。
(,200)
(,304)
A = LOAD ‘test_logs’ using org.apache.pig.backend.hadoop.hbase.HBaseStorage(‘tag:key, response:code, tag:eclipsed_time’) as (o:chararray, c:int, e:double);
于是没有返回。

2. 数据看起来是这样的，但实际上每一行都是chararray：
((a,1),(b,2))
((a,1),(b,1),(c,3))
我想找出a，b，c的个数，最好能按后面的数字group一下。

初学，不好意思。

回复

回复