

Flume事务机制

概述

Flume的事务机制与可靠性保证的实现，最核心的组件是Channel（通道）。如果没有Channel组件，而紧靠Source与Sink组件是无从谈起的。

文件通道（File Channel）

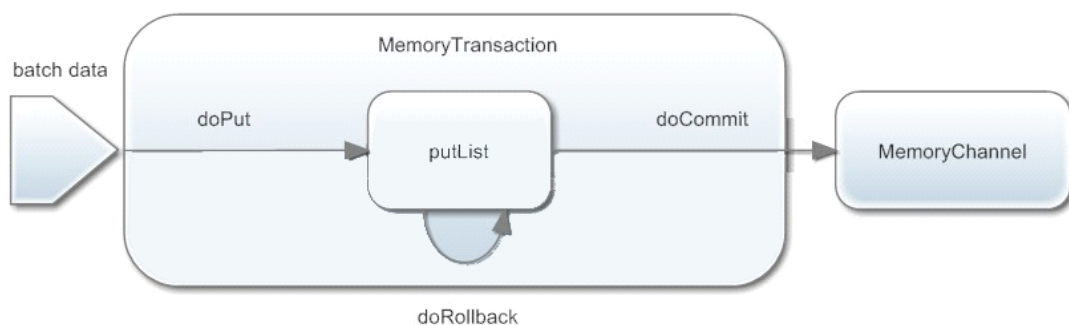
文件通道指的是将事件存储到代理（Agent）本地文件系统中的通道。虽然要比内存通道慢一些，不过它却提供了持久化的存储路径，可以应对大多数情况，它应该用在数据流中不允许出现缺口的场合。

内存通道

File channel虽然提供了持久化，但是其性能较差，吞吐量会受到一定的限制。相反，memory channel则牺牲可靠性换取吞吐量。当然，如果机器断电重启，则无法恢复。在实际应用中，大多数企业都是选择内存通道，因为在通过flume收集海量数据场景下，使用FileChannel所带来的性能下降是很大的甚至是无法忍受的。

Flume内存通道事务机制

编程模型



Put事务流程

Put事务可以分为以下阶段：

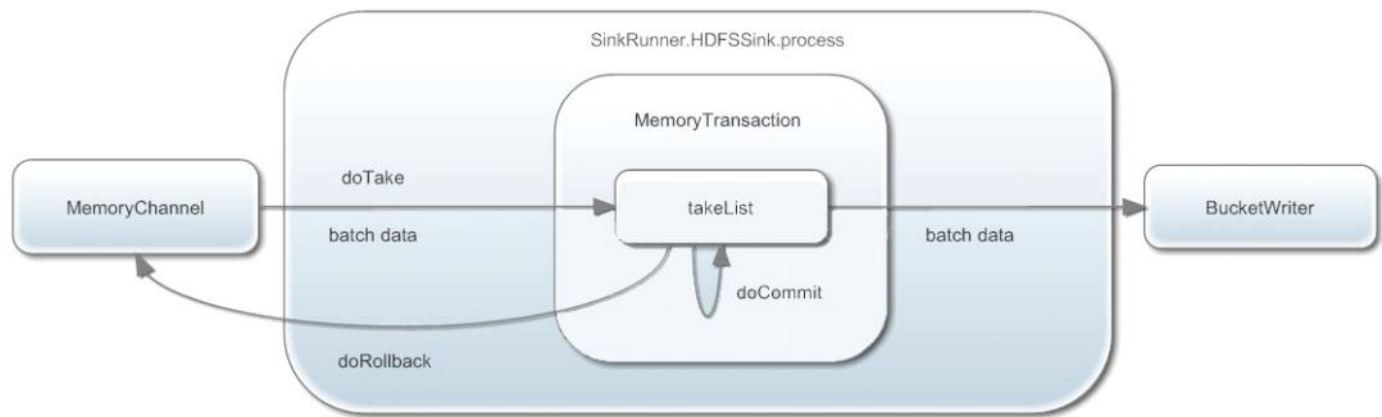
doPut:将批数据先写入临时缓冲区putList（Linkedblockingdequeue）

doCommit:检查channel内存队列是否足够合并。

doRollback:channel内存队列空间不足，回滚，等待内存通道的容量满足合并

putList就是一个临时的缓冲区，数据会先put到putList,最后由commit方法会检查channel是否有足够的缓冲区，有则合并到channel的队列。

Take事务



Take事务分为以下阶段：

doTake:先将数据取到临时缓冲区takeList(linkedBlockingDeque)

将数据发送到下一个节点

doCommit:如果数据全部发送成功，则清除临时缓冲区takeList

doRollback:数据发送过程中如果出现异常，rollback将临时缓冲区takeList中的数据归还给channel内存队列。

Hive介绍

Hadoop开发存在的问题

只能用java语言开发，如果是c语言或其他语言的程序员用Hadoop，存在语言门槛。

需要对Hadoop底层原理，api比较了解才能做开发。

Hive概述

Hive是基于Hadoop的一个**数据仓库工具**。可以将结构化的数据文件映射为一张表，并提供完整的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。其优点是学习成本低，可以通过**类SQL**语句快速实现MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

Hive是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行**数据提取、转化、加载（ETL Extract-Transform-Load）**，这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HiveQL，它允许熟悉 SQL 的用户查询数据。

Hive的Hql

HQL - Hive通过类SQL的语法，来进行分布式的计算。HQL用起来和SQL非常的类似，Hive**在运行的过程中会将HQL转换为MapReduce去执行**，所以Hive其实是基于Hadoop的一种分布式计算框架，**底层仍然是MapReduce**，所以它本质上还是一种离线大数据分析工具。

数据仓库的特征

1. 数据仓库是多个异构数据源所集成的。
2. 数据仓库存储的一般是**历史数据**。
3. 数据库是为**捕获数据**而设计，数据仓库是为**分析数据**而设计。

4. 数据仓库是时变的，数据存储从历史的角度提供信息。即数据仓库中的关键结构都隐式或显式地包含时间元素。

5.数据仓库是弱事务的，因为数据仓库存的是历史数据，一般都读（分析）数据场景。

数据库属于OLTP系统。（Online Transaction Processing）联机**事务**处理系统。涵盖了企业大部分的日常操作，如购物、库存、制造、银行、工资、注册、记账等。

数据仓库属于OLAP系统。（Online Analytical Processing）联机**分析**处理系统。

OLTP是面向用户的、用于程序员的事务处理以及客户的查询处理。

OLAP是面向市场的，用于知识工人（经理、主管和数据分析人员）的数据分析。

OLAP通常会集成多个异构数据源的数据，数量巨大。

OLTP系统的访问由于要保证原子性，所以有事务机制和恢复机制。

OLAP系统一般存储的是历史数据，所以大部分都是只读操作，不需要事务。

表 4.1 OLTP 系统与 OLAP 系统的比较

特征	OLTP	OLAP
特性	操作处理	信息处理
面向	事务	分析
用户	办事员、DBA、数据库专业人员	知识工人（如经理、主管、分析人员）
功能	日常操作	长期信息需求、决策支持
DB 设计	基于 E-R，面向应用	星形/雪花、面向主题
数据	当前的、确保最新	历史的、跨时间维护
汇总	原始的、高度详细	汇总的、统一的
视图	详细、一般关系	汇总的、多维的
工作单元	短的、简单事务	复杂查询
访问	读/写	大多为读
关注	数据进入	信息输出
操作	主码上索引/散列	大量扫描
访问记录数量	数十	数百万
用户数	数千	数百
DB 规模	GB 到高达 GB	≥TB
优先	高性能、高可用性	高灵活性、终端用户自治
度量	事务吞吐量	查询吞吐量、响应时间

注：该表部分基于 Chaudhuri 和 Dayal[CD97]。

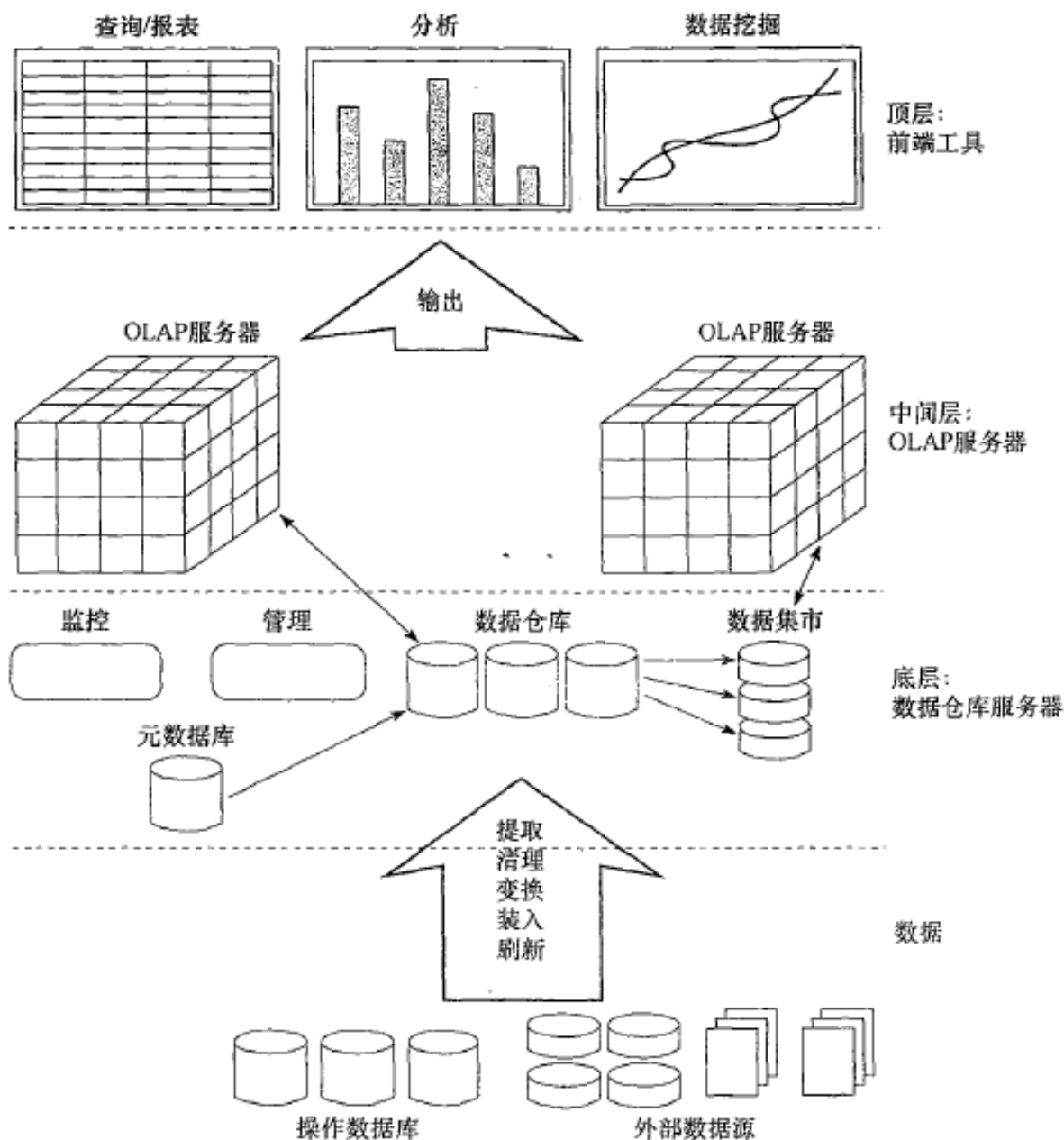


图 4.1 三层数据仓库结构

适用场景

Hive 构建在基于静态批处理的 Hadoop 之上，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。**因此，Hive 并不能够在大规模数据集上实现低延迟快速的查询**，例如，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。因此，Hive 并不适合那些需要低延迟的应用，例如，联机事务处理(OLTP)。Hive 查询操作过程严格遵守 Hadoop MapReduce 的作业执行模型，Hive 将用户的 HiveQL 语句通过解释器转换为 MapReduce 作业提交到 Hadoop 集群上，Hadoop 监控作业执行过程，然后返回作业执行结果给用户。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。**Hive 的最佳使用场合是大数据集的批处理作业，例如，网络日志分析。**

Hive的安装配置

实现步骤

- 1.安装JDK
- 2.安装Hadoop
- 3.配置JDK和Hadoop的环境变量
- 4.下载Hive安装包
- 5.解压安装hive
- 6.启动Hadoop的HDFS和Yarn
- 7.启动Hive

进入到bin目录, 指定: sh hive (或者执行: ./hive)

```
Logging initialized using configuration in jar:file:/home/software/apache-hive-1.2.0
-bin/lib/hive-common-1.2.0.jar!/hive-log4j.properties
hive> █
```


Hive基础指令

命令	作用	额外说明
show databases;	查看都有哪些数据库	
create database park;	创建park数据库	创建的数据库，实际是在Hadoop的HDFS文件系统里创建一个目录节点，统一存在： /user/hive/warehouse 目录下 
use park;	进入park数据库	
show tables;	查看当前数据库下所有表	
create table stu (id int,name string);	创建stu表，以及相关的两个字段	hive里，用的是string，不用char和varchar。此外，所创建的表，也是HDFS里的一个目录节点 
insert into stu values(1,'zhang')	向stu表插入数据	<p>1.HDFS不支持数据的修改和删除，但是在2.0版本后支持了数据追加。实际上，insert into 语句执行的是追加操作。</p> <p>2.所以注意：hive支持查询，行级别的插入。不支持行级别的删除和修改。</p> <p>3.hive的操作实际是执行一个job任务，调用的是Hadoop的MR。</p> <pre>Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0 2016-11-15 22:04:34,423 Stage-1 map = 0%, reduce = 0% 2016-11-15 22:05:07,843 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.03 sec MapReduce Total cumulative CPU time: 2 seconds 30 msec Ended Job = job_1478819204253_0001 Stage-4 is selected by condition resolver. Stage-3 is filtered out by condition resolver. Stage-5 is filtered out by condition resolver. Moving data to: hdfs://hadoop01:9000/user/hive/warehouse/park.db/stu/.hive-staging_hive_2016-11-15_22-03-30_182_448915450850878957-1/-ext-10000 Loading data to table park.stu Table park.stu stats: [numFiles=1, numRows=1, totalSize=8, rawDataSize=7] MapReduce Jobs Launched: Stage-Stage-1: Map: 1 Cumulative CPU: 2.49 sec HDFS Read: 3540 HDFS Write: 72 SUCCESS Total MapReduce CPU Time Spent: 2 seconds 490 msec OK Time taken: 118.079 seconds</pre> <p>4.插入完数据之后，发现HDFS stu目录节点下，多了一个文件,文件里存了插入的数据，所以得出结论，hive存储的数据，是通过HDFS的文件来存储的。</p> <div></div>
select * from stu	查看表数据	也可以根据字段来查询，比如select id from stu

drop table stu	删除表	
select * from stu	查询stu表数据	<p>有时会出现这样的情况：</p> <pre>hive> select * from stu; OK NULL NULL NULL NULL NULL NULL Time taken: 0.863 seconds, Fetched: 3 row(s)</pre> <p>原因是：我们从外部导入的文件的数据格式是：</p> <pre>1 jary 2 rose</pre> <p>即第一列和第二列是以空格为分隔符的。</p> <p>但是把数据导入到hive之后，hive并不知道分隔符是什么，所以就不能正确的切分数据。所以显示null。</p> <p>解决办法：在hive创建表的时候，要指定分割符，并且这个分割符要和外部文件里的分割符一致。详见指令8</p>
load data local inpath '/home/software/1.txt' into table stu;	通过加载文件数据到指定的表里	 <p>在执行完这个指令之后，发现hdfs stu目录下多了一个1.txt文件。由此可见，hive的工作原理实际上就是在管理hdfs上的文件，把文件里数据抽象成二维表结构，然后提供hql语句供程序员查询文件数据。</p> <p>可以做这样的实验：不通过load 指令，而通过插件向stu目录下再上传一个文件，看下hive是否能将数据管理到stu表里。</p>
create table stu1(id int,name string) row format delimited fields terminated by ' ';	创建stu1表，并指定分割符 空格。	<p>此时，把外部数据导入hive，就可以正确查出数据了。</p> <pre>hive> select * from stu1; OK 1 rose 2 tom 3 chen Time taken: 0.081 seconds, Fetched: 3 row(s)</pre>
desc stu	查看 stu表结构	
create table stu2 like stu	创建一张stu2表，表结构和stu表结构相同	like只复制表结构，不复制数据
insert overwrite table stu2 select * from stu	把stu表数据插入到stu2表中	<p>insert overwrite 可用于将select 查询出的数据插入到指定的表中或指定的目录下</p> <p>比如：把查询结果存到本地指定的目录下，</p> <p>执行：<code>insert overwrite local directory '/home/stu' row format delimited fields terminated by ' ' select * from stu;</code></p> <p>也可以将查询结果存到HDFS文件系统上</p> <p>执行：<code>insert overwrite directory '/stu' row format delimited fields terminated by ' ' select * from stu;</code></p> <p>也可以将查询结果插入到多张表中</p> <p>执行：<code>from stu insert overwrite table stu1 select * insert overwrite table stu2 select *;</code></p> <p>结果是把stu表的数据插入 stu1和stu2 表。（也可以加where 条件等，比如select * where id>3）</p>
alter table stu	为表stu重命名为	

rename to stu2	stu2	
alter table stu add columns (age int);	为表stu增加一个列 字段age，类型为int	
exit	退出hive	<p>当退出hive后，我们可以尝试做这样的一件事：</p> <p>之前我们是在bin目录执行：sh hive 进入的</p> <p>现在，我们换一个目录来进入hive,比如： sh /bin/hive 来进入</p> <p>当我们查看数据库或查看表时，发现之前建立的park和stu表都没有了。</p> <p>原因：hive可以管理hdfs上的文件，用表的形式来管理文件数据。</p> <p>而表名、表里有哪些字段，字段类型、哪张表存在哪个数据下等这些表信息，称之为hive的元数据信息。</p> <p>知识点：hive的元数据信息不是存在hdfs上的，而是存在hive自带的derby关系型数据库里的。</p> <p>即hive管理的数据是在hdfs上的，hive的元数据信息是存在关系型数据库里的。</p> <p>上述问题出现的原因就是由于derby数据库引起的，这个数据库功能不完善，仅用于测试。</p> <p>derby数据库存储hive元数据的方式：当在bin目录下进入hive时，derby数据会在bin目录下生成一个metastore_db目录，将元数据信息存在这个目录下。</p> <p>当换个目录，比如在home 目录下hive时，derby又会在home目录下生成一个metastore_db目录，存储元数据信息。</p> <p>解决办法：将默认使用的derby数据库换成mysql数据库</p>

Linux下的Mysql安装

2015年11月17日 11:02

Hive的数据，是存在HDFS里的。此外，hive有哪些数据库，每个数据库有哪些表，这样的信息称之为hive的元数据信息。

元数据信息不存在HDFS里。而是存在关系型数据库里，hive默认用的是derby数据库来存储。即hive工作时，除了要依赖Hadoop，还要依赖关系型数据库。

注意：虽然我们能通过HDFS查看到hive有哪些数据库，有哪些表，以及表里的数据，但是，这不是元数据信息。HDFS最主要的是存储hive的数据信息。

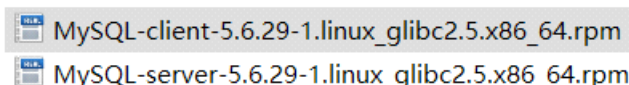
之前遇到的问题是：当退出后，切换到不同的目录来进入hive，发现库和表没有了，是因为，第一次从bin目录进入hive,会在bin目录下创建一个metastore.db目录，在这个目录下，创建一个derby.log文件来存储元数据信息。这个元数据信息是基于bin目录来创建的。而切换到其他目录进入hive时，查询时不是基于bin目录来查询的，所有查不到元数据信息，导致查不到。

这个问题是derby数据库本身的问题，所以，我们不能用derby数据库，此外，用derby数据库，也不支持并发，比如一个人在操作hive，如果此时有其他人想用hive，用不了。

所以我们选择用mysql数据库。目前hive支持derby和mysql两种数据库。

安装步骤

1. 下载mysql安装包



```
MySQL-client-5.6.29-1.linux_glibc2.5.x86_64.rpm  
MySQL-server-5.6.29-1.linux_glibc2.5.x86_64.rpm
```

2.确认当前虚拟机之前是否有安装过mysql

执行：rpm -qa 查看linux安装过的所有rpm包

执行：rpm -qa | grep mysql

如果出现下图，证明已经安装了mysql，需要删除

```
[root@hadoop01 mysql]# rpm -qa |grep mysql
mysql-libs-5.1.71-1.el6.x86_64
```

3.删除mysql

执行：rpm -ev --nodeps mysql-libs-5.1.71-1.el6.x86_64

此时，再执行：rpm -qa | grep mysql 发现没有相关信息了

4.新增mysql用户组，并创建mysql用户

groupadd mysql

useradd -r -g mysql mysql

5.安装mysql server rpm包和client包

执行：rpm -ivh MySQL-server-5.6.29-1.linux_glibc2.5.x86_64.rpm

rpm -ivh MySQL-client-5.6.29-1.linux_glibc2.5.x86_64.rpm

6.安装后，mysql文件所在的目录

Directory	Contents of Directory
/usr/bin	Client programs and scripts
/usr/sbin	The mysqld server

/var/lib/mysql	Log files, databases
/usr/share/info	MySQL manual in Info format
/usr/share/man	Unix manual pages
/usr/include/mysql	Include (header) files
/usr/lib/mysql	Libraries
/usr/share/mysql	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
/usr/share/sql-bench	Benchmarks

7.修改my.cnf,默认在/usr/my.cnf

执行：vim /usr/my.cnf

[client]

default-character-set=utf8

[mysql]

default-character-set=utf8

[mysqld]

character_set_server=utf8

8.将mysqld加入系统服务，并随机启动

执行：cp /usr/share/mysql/mysql.server /etc/init.d/mysqld

说明：/etc/init.d 是linux的一个特殊目录，放在这个目录的命令会随linux开机而启动。

9.启动mysqld

执行：service mysqld start

```
[root@hadoop01 mysql]# cp /usr/share/mysql/mysql.server /etc/init.d/mysqld
[root@hadoop01 mysql]# service mysqld start
Starting MySQL... SUCCESS!
```

10.查看初始生成的密码

执行：vim /root/.mysql_secret 。这个密码随机生成的

```
# The random password set for
: UCgUjqWmTcBNctCJ
```

11.修改初始密码

第一次安装完mysql后，需要指定登录密码

执行：mysqladmin -u root -p password root 此时，提示要输入初始生成的密码，拷贝过来即可

10.进入mysql数据库

执行：mysql -u root -p

输入：root进入

执行：\s

查看mysql数据配置信息

Mysql安装遇到问题的解决

2018年9月4日 11:20

1.执行：rpm -qa | grep Percona

2.执行：ps -aux | grep mysql

查看之前Percona运行的mysql进程，找到进程号，kill 掉

3.删除所有和Percona相关的rpm包

4.安装 mysql-server 和mysql-client

5.配置my.cnf 字符集

6.将mysqld服务随机启动目录

7.启动mysqld服务

Hive的mysql安装配置

2015年11月17日 13:02

实现步骤：

1.删除hdfs中的/user/hive

执行：hadoop fs -rmr /user/hive

2.将mysql驱动包上传到hive安装目录的lib目录下

3.编辑新的配置文件，名字为：hive-site.xml

4.配置相关信息：

```
<configuration>
```

```
<property>
```

```
<name>javax.jdo.option.ConnectionURL</name>
```

```
<value>jdbc:mysql://hadoop01:3306/hive?createDatabaseIfNotExist=true</value>
```

```
</property>
```

```
<property>
```

```
<name>javax.jdo.option.ConnectionDriverName</name>
```

```
<value>com.mysql.jdbc.Driver</value>
```

```
</property>
```

```
<property>
```

```
<name>javax.jdo.option.ConnectionUserName</name>
```

```
<value>root</value>
```

```
</property>
```

```
<property>

<name>javax. jdo. option. ConnectionPassword</name>

<value>root</value>

</property>

</configuration>
```

5.进入hive,进入bin目录,执行: sh hive

如果出现:

Access denied for user 'root'@'hadoop01' (using password: YES) 这个错误,指的是当前用户操作mysql数据库的权限不够。

6.进入到mysql数据库,进行权限分配

执行:

```
grant all privileges on *.* to 'root'@'hadoop01' identified by 'root' with grant option;
```

然后执行:

```
flush privileges;
```

7.如果不事先在mysql里创建hive数据库,在进入hive时,mysql会自动创建hive数据库。但是注意,因为我们之前配置过mysql的字符集为utf-8,所以这个自动创建的hive数据库的字符集是utf-8的。

但是hive要求存储元数据的字符集必须是iso8859-1。如果不是的话,hive会在创建表的时候报错(先是卡一会,然后报错)。

解决办法:在mysql数据里,手动创建hive数据库,并指定字符集为iso8859-1;

进入mysql数据库,

```
然后执行:create database hive character set latin1;
```

8.以上步骤都做完后，再次进入mysql的hive数据，发现有如下的表：

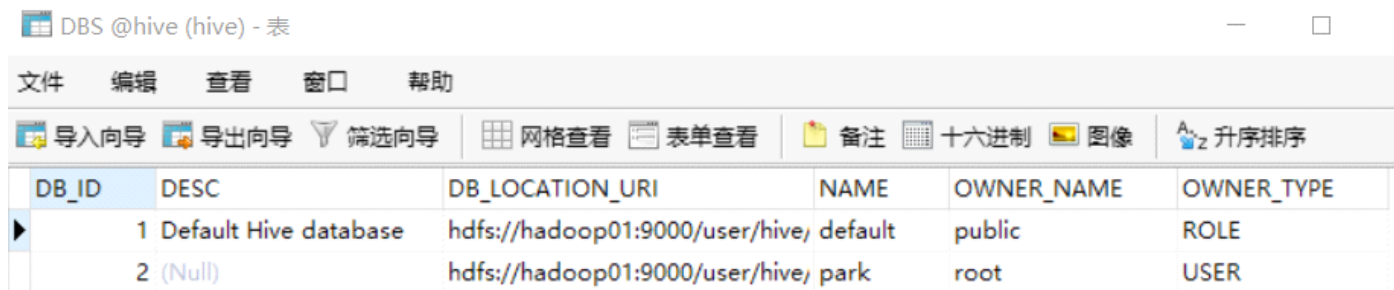
```
mysql> show tables;
+-----+
| Tables_in_hive |
+-----+
| BUCKETING_COLS |
| CDS             |
| COLUMNS_V2     |
| DATABASE_PARAMS |
| DBS             |
| FUNCS          |
| FUNC_RU         |
| GLOBAL_PRIVS    |
| PARTITIONS      |
| PARTITION_KEYS  |
| PARTITION_KEY_VALS |
| PARTITION_PARAMS |
| PART_COL_STATS  |
| ROLES           |
| SDS            |
| SD_PARAMS       |
| SEQUENCE_TABLE  |
| SERDES          |
| SERDE_PARAMS    |
| SKEWED_COL_NAMES |
| SKEWED_COL_VALUE_LOC_MAP |
| SKEWED_STRING_LIST |
| SKEWED_STRING_LIST_VALUES |
| SKEWED_VALUES   |
| SORT_COLS       |
| TABLE_PARAMS   |
| TAB_COL_STATS   |
| TBLS            |
| VERSION         |
+-----+
29 rows in set (0.00 sec)
```

9.可以通过navicat来连接数据库。



10.可以通过DBS、TBLS、COLUMNS_V2这三张表来查看元数据信息。

DBS 存放的数据库的元数据信息



The screenshot shows the 'DBS @hive (hive) - 表' window. The table has 6 columns: DB_ID, DESC, DB_LOCATION_URI, NAME, OWNER_NAME, and OWNER_TYPE. There are two rows of data.

DB_ID	DESC	DB_LOCATION_URI	NAME	OWNER_NAME	OWNER_TYPE
1	Default Hive database	hdfs://hadoop01:9000/user/hive/	default	public	ROLE
2	(Null)	hdfs://hadoop01:9000/user/hive/	park	root	USER

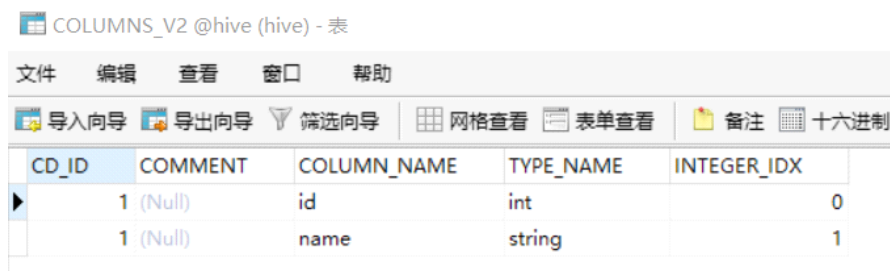
TBLS存放的tables表信息



The screenshot shows the 'TBLS @hive (hive) - 表' window. The table has 11 columns: TBL_ID, CREATE_TIME, DB_ID, LAST_ACCESS_TIME, OWNER, RETENTION, SD_ID, TBL_NAME, TBL_TYPE, VIEW_EXPANDED_TEXT, and VIEW_ORIGINAL_TEXT. There is one row of data.

TBL_ID	CREATE_TIME	DB_ID	LAST_ACCESS_TIME	OWNER	RETENTION	SD_ID	TBL_NAME	TBL_TYPE	VIEW_EXPANDED_TEXT	VIEW_ORIGINAL_TEXT
1	1479296362	2		0 root		0	1 stu	MANAGED_TABLE	(Null)	(Null)

COLUMNS表存放的是列字段信息



The screenshot shows the 'COLUMNS_V2 @hive (hive) - 表' window. The table has 5 columns: CD_ID, COMMENT, COLUMN_NAME, TYPE_NAME, and INTEGER_IDX. There are two rows of data.

CD_ID	COMMENT	COLUMN_NAME	TYPE_NAME	INTEGER_IDX
1	(Null)	id	int	0
1	(Null)	name	string	1

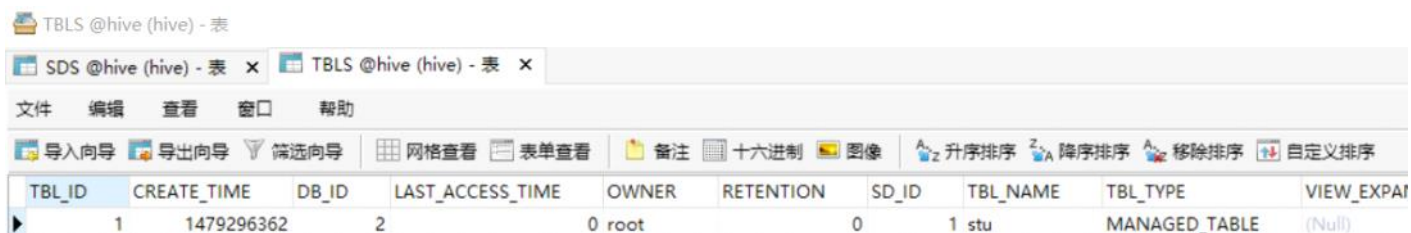
此外，可以通过查看SDS表来查询HDFS里的位置信息

文件 编辑 查看 窗口 帮助					
导入向导 导出向导 筛选向导 网格查看 表单查看 备注 十六进制 图像 升序排序 降序排序 移除排序 刷新					
SD_ID	CD_ID	INPUT_FORMAT	IS_COMPRESSED	IS_STOREDASS	LOCATION
1	1	org.apache.hadoop.mapreduce	0	0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/stu

Hive的内部表和外部表

2015年11月17日 13:02

在查看元数据信息时，有一张TBLS表，
其中有一个字段属性：TBL_TYPE——MANAGED_TABLE
MANAGED_TABLE 表示内部表

A screenshot of a web interface showing the TBLS table in Hive. The table has columns: TBL_ID, CREATE_TIME, DB_ID, LAST_ACCESS_TIME, OWNER, RETENTION, SD_ID, TBL_NAME, TBL_TYPE, and VIEW_EXPANDED. The first row shows TBL_ID 1, CREATE_TIME 1479296362, DB_ID 2, OWNER root, SD_ID 0, TBL_NAME stu, and TBL_TYPE MANAGED_TABLE.

TBL_ID	CREATE_TIME	DB_ID	LAST_ACCESS_TIME	OWNER	RETENTION	SD_ID	TBL_NAME	TBL_TYPE	VIEW_EXPANDED
1	1479296362	2		root		0	stu	MANAGED_TABLE	(Null)

内部表的概念

先在hive里建一张表，然后向这个表插入数据（用insert可以插入数据，也可以通过加载外部文件方式来插入数据），
这样的表称之为hive的内部表。

外部表的概念

HDFS里已经有数据了，比如有一个2.txt文件，里面存储了这样的一些数据：

1 jary
2 rose

然后，通过hive创建一张表stu来管理这个文件数据。则stu这样表称之为外部表。注意，hive外部表管理的是HDFS里的某一个目录下的文件数据。

所以，做这个实验，要先HDFS创建一个目录节点，然后把数据文件上传到这个目录节点下。

创建外部表的命令：

进入hive，执行：

```
create external table stu (id int,name string) row format delimited fields terminated by ' ' location '/目录路径'
```

然后查看TBLS表，

RETENTION	SD_ID	TBL_NAME	TBL_TYPE	VIEW_EXPANDED
	0	1 stu	MANAGED_TABLE	(Memo)
	0	6 teacher	EXTERNAL_TABLE	(Memo)

hive无论是内部表或外部表，当向HDFS对应的目录节点下追加文件时（只要格式符合），
hive都可以把数据管理进来

内部表和外部表的区别

通过hive执行：drop table stu。这是删除表操作。如果stu是一个内部表，则HDFS对应的目录

节点会被删除。

如果stu是一个外部表，HDFS对应的目录节点不会删除

Hive分区表

2015年11月17日 17:06

概念

Hive也支持分区表,对数据进行分区可以提高查询时的效率。

普通表和分区表区别：有大量数据增加的需要建分区表

语法

执行：create table book (id int, name string) partitioned by (category string)

row format delimited fields terminated by '\t';

注：在创建分区表时，partitioned字段可以不在字段列表中。生成的表中自动就会具有该字段。

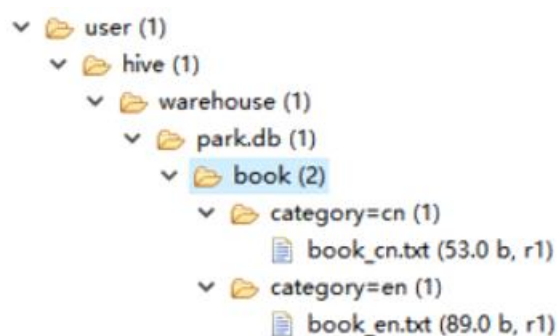
category 是自定义的字段。

分区表加载数据

1) load data local inpath '/home/cn.txt' overwrite into table book partition (category='cn');

2) load data local inpath './book_english.txt' overwrite into table book partition (category='en');

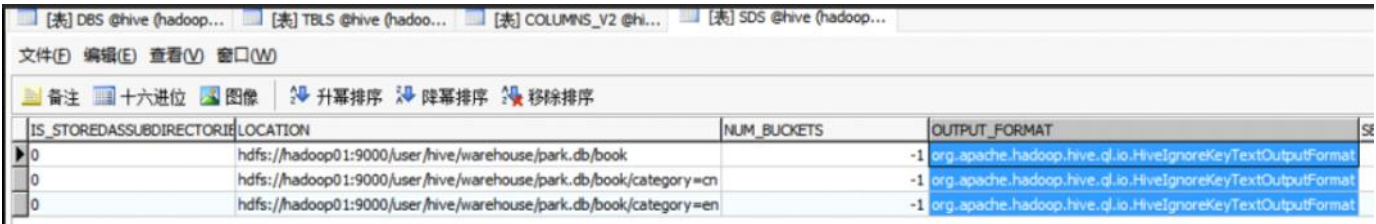
经检查发现分区也是一个目录。



select * from book; 查询book目录下的所有数据

select * from book where category='cn'; 只查询 cn分区的数据

此外，通过查看mysql的SDS表来查询元数据信息



IS_STOREDASSUBDIRECTORIE	LOCATION	NUM_BUCKETS	OUTPUT_FORMAT
0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/book	-1	org.apache.hadoop.hive.gl.io.HiveIgnoreKeyTextOutputFormat
0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/book/category=cn	-1	org.apache.hadoop.hive.gl.io.HiveIgnoreKeyTextOutputFormat
0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/book/category=en	-1	org.apache.hadoop.hive.gl.io.HiveIgnoreKeyTextOutputFormat

通过创建目录来增加分区

如果想先在HDFS的目录下，自己创建一个分区目录，然后在此目录下上传文件，比如：



此时手动创建目录是无法被hive使用的，因为元数据库中并没有记录该分区。

如果需要将自己创建的分区也能被识别，

需要执行：ALTER TABLE book add PARTITION (category = 'jp') location

'/user/hive/warehouse/park.db/book/category=jp';

这行命令的作用是在元数据Dock表里创建对应的元数据信息

INPUT_FORMAT	IS_COMPRESSED	IS_STOREDASSUBDIRECTORIE	LOCATION	NUM_BUCKETS
org.apache.hadoop.mapred.TextInputFormat	0	0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/book	-1
org.apache.hadoop.mapred.TextInputFormat	0	0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/book/category=cn	-1
org.apache.hadoop.mapred.TextInputFormat	0	0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/book/category=en	-1
org.apache.hadoop.mapred.TextInputFormat	0	0	hdfs://hadoop01:9000/user/hive/warehouse/park.db/book/category=jp	-1

分区命令

1.显示分区

```
show partitions iteblog;
```

2.添加分区

```
alter table book add partition (category='jp') location  
'/user/hive/warehouse/test.db/book/category=jp';
```

或者：

```
msck repair table book;
```

3.删除分区

```
alter table book drop partition(category='cn')
```

4.修改分区

```
alter table book partition(category='french') rename to partition (category='hh');
```

Hive 数据类型

2016年1月30日 12:46

常用的基本数据类型

基本数据类型	所占字节
int	
boolean	
float	
double	
string	

复杂数据类型

复杂数据类型	说明
array	array类型是由一系列相同数据类型的元素组成。并且可以通过下标来进行访问。 注意:下标从0开始计
map	map包含key-value 键值对，可以通过key来访问元素
struct	struct 可以包含不同数据类型元素。相当于一个对象结构。可以通过 对象.属性 来访问

一、数组类型 array

案例一



元数据：

100,200,300
200,300,500



建表语句：

```
create external table ex(vals array<int>) row format delimited fields terminated by  
'\t' collection items terminated by ';' location '/ex';
```



查询每行数组的个数，查询语句：

```
select size(vals) from ex;
```

注：hive 内置函数不具备查询某个具体行的数组元素。需要自定义函数来实现，但这样的需求在实际开发里很少，所以不需要在意。

案例二

元数据：

```
100,200,300    tom,jary
200,300,500    rose,jack
```

建表语句：

```
create external table ex1(info1 array<int>,info2 array<string>) row format
delimited fields terminated by '\t' collection items terminated by ',' location
'/ex';
```

结果：

```
hive> select * from ex1;
OK
[100,200,300]    ["tom","jary"]
[200,300,500]    ["rose","jack"]
Time taken: 0.124 seconds, Fetched: 2 row(s)
```

二、map类型

案例一

元数据：

```
tom,23
rose,25
jary,28
```

建表语句：

```
create external table m1 (vals map<string,int>) row format delimited fields
terminated by '\t' map keys terminated by ',' location '/map';
```

查询语句：

```
select vals['tom'] from m1;
```

```
hive> select vals['tom'] from ex;
OK
23
NULL
NULL
```

案例二，要求查询tom这个人都浏览了哪些网站，并且为null的值不显示

源数据（分隔符为空格）：

```
tom 192.168.234.21
rose 192.168.234.21
tom 192.168.234.22
jary 192.168.234.21
tom 192.168.234.24
tom 192.168.234.21
rose 192.168.234.21
tom 192.168.234.22
jary 192.168.234.21
```

```
tom 192.168.234.22
tom 192.168.234.23
```

建表语句

create external table ex (**vals map<string,string>**) row format delimited fields terminated by '/t' map keys terminated by ' ' location '/ex';

注意：map类型，列的分割符必须是\t

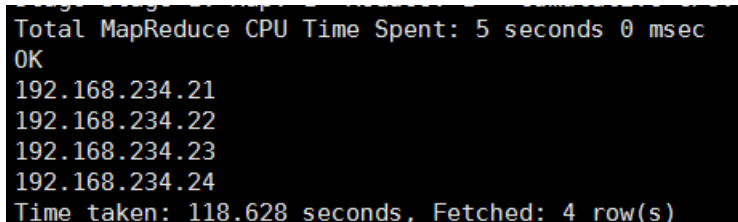
查询语句

```
select vals['tom'] from ex where vals['tom'] is not null;
```

如果想做去重工作，可以调用distinct内置函数

```
select distinct(ip) from (select vals['tom'] ip from ex where vals['tom'] is not null)ex1;
```

```
select distinct(vals['tom']) from m2 where vals['tom'] is not null;
```



```
Total MapReduce CPU Time Spent: 5 seconds 0 msec
OK
192.168.234.21
192.168.234.22
192.168.234.23
192.168.234.24
Time taken: 118.628 seconds, Fetched: 4 row(s)
```

三、struct 类型

元数据：

```
tom 23
rose 22
jary 26
```

建表语句：

create external table ex (vals **struct<name:string,age:int>**)row format delimited collection items terminated by ' ' location '/ex';

查询语句：

```
select vals.age from ex where vals.name='tom';
```

Hive explode

2015年11月19日 11:50

explode 命令可以将行数据，按指定规则切分出多行。

案例一，利用split执行切分规则

有如下数据：

100,200,300

200,300,500

要将上面两行数据根据逗号拆分成多行（每个数字占一行）

实现步骤

1.准备元数据

2.上传HDFS，并创建对应的外部表

执行：create external table ex1 (num string) location '/ex';

注：用explode做行切分，注意表里只有一列，并且行数据是string类型，因为只有字符类型才能做切分。

3.通过explode指令来做行切分

执行：select explode(split(num,',')) from ex1;

```
hive> select explode(split(num,',')) from ex1;
OK
100
200
300
200
300
500
```

Hive常用字符串操作函数

2015年11月21日 14:36

返回类型	函数名	描述
int	length(string A)	返回字符串A的长度 select length(weoirjewo); select length(name) from stu; 此函数在实际工作，可以用于校验手机号，身份证号等信息的合法新
string	reverse(string A)	返回字符串A的反转结果 select reverse('abcd'); select length(name) from stu;
string	concat(string A, string B...)	字符串连接函数 select concat ('a','b'); select concat(id,name) from stu; select concat(id,',',name) from stu; http:// www.baidu.com ?get
string	concat_ws(string SEP, string A, string B...)	带分隔符字符串连接函数：concat_ws select concat_ws('.', 'www', 'baidu', 'com'); //www.baidu.com
string	substr	substr,substring select substr('abcde',2);从第二个截，截到结尾

		<p>select substr('abcde',1,3);从第一个截，截三个长度</p> <p>select substr('wfeww',-2);从尾部截，截两个长度</p> <p>可以用于比如截取身份证后几位操作</p>
string	upper(string a) ucase(string a)	转大写
string	lower(string a) lcase(string a)	转小写
string	trim(string a)	去空格 select trim (' fwoei ');
string	ltrim(string a)	左边去空格函数
string	rtrim(string a)	右边去空格函数
string	regexp_replace (string A, string B, string C)	将字符串A中的符合java正则表达式B的部分替换为C。注意，在有些情况下要使用转义字符，对需要转义的字符，用[]，比如[*]，类似oracle中的regexp_replace函数。
string	regexp_extract (string subject, string pattern, int index)	将字符串subject按照pattern正则表达式的规则拆分，返回index指定的字符 select regexp_extract('foothebar', 'foo(.*)(bar)', 1) ; //the select regexp_extract('foothebar', 'foo(.*)(bar)', 2) ; //bar select regexp_extract('foothebar', 'foo(.*)(bar)', 0) ; //全取 foothebar
string	repeat(string str, int n)	返回重复n次后的str字符串 select repeat('abc',5)
array	split(string str, string pat)	分割字符串函数: split 按照pat字符串分割str，会返回分割后的字符串数组 select split('abtcdef','t') ;

		["ab","cd","ef"]
--	--	------------------

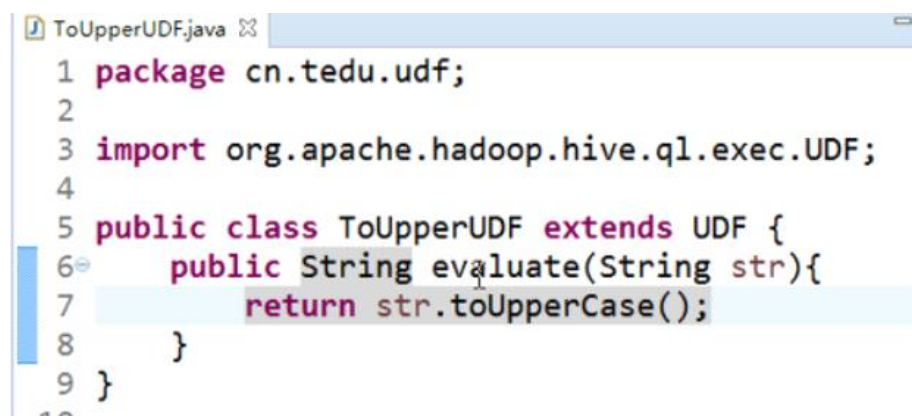
Hive的UDF

2015年11月17日 13:02

如果hive的内置函数不够用，我们也可以自己定义函数来使用，这样的函数称为hive的用户自定义函数，简称UDF。

实现步骤：

- 1.新建java工程，导入hive相关包，导入hive相关的lib。
- 2.创建类继承UDF
- 3.自己编写一个evaluate方法，返回值和参数任意。

A screenshot of an IDE window titled 'ToUpperUDF.java'. The code is as follows:

```
1 package cn.tedu.udf;
2
3 import org.apache.hadoop.hive.ql.exec.UDF;
4
5 public class ToUpperUDF extends UDF {
6     public String evaluate(String str){
7         return str.toUpperCase();
8     }
9 }
```



代码示例：

```
import org.apache.hadoop.hive.ql.exec.UDF;
```

```
public class ToUpper extends UDF{

    public String evaluate(String str){

        return str.toUpperCase();

    }

}
```

4.为了能让mapreduce处理，String要用Text处理。

5.将写好的类打成jar包，上传到linux中

6.在hive命令行下，向hive注册UDF：add jar /xxxx/xxxx.jar

```
hive> add jar /root/work/data/udf.jar
> ;
Added [/root/work/data/udf.jar] to class path
Added resources: [/root/work/data/udf.jar]
```

7.在hive命令行下，为当前udf起一个名字：create temporary function fname as '类的全路径名'；

```
hive> create temporary function mytoup as 'cn.tedu.udf.ToUpper';
```

8.之后就可以在hql中使用该自定义函数了。

```
hive> select mytoup('asdfJKdssdfgUijgafdJKLJKLJLdfsafa');
OK
ASDFJKDSSDFGUIJGAFDJKLJKLJLDFSafa
Time taken: 0.175 seconds, Fetched: 1 row(s)
```