

Flink on zeppelin第三弹UDF的使用

用户自定义函数是非常重要的一个特征，因为它极大地扩展了查询的表达能力。不管是在spark sql或者hive中都支持用户自定义UDF的使用,虽然Flink提供了很多内置的UDF可以直接使用,但是一些特定的场景可能需要我们自定义UDF去实现,这篇文章会主要介绍一下在Flink on zeppelin里面使用UDF的3种方法.

1, 开发scala代码并注册UDF

因为目前zeppelin支持Scala, Python, SQL 三种语言,还不支持java.所以可以编写scala或者Python代码来开发UDF.具体的实现和在代码里面是一样的.

```
%flink

class MyScalaUdfDemo extends ScalarFunction {
    def eval(str: String) = str.toUpperCase
}

btenv.createTemporarySystemFunction("MyScalaUdfDemo", new MyScalaUdfDemo)
```

上面实现了一个非常简单的UDF,把字符串转为大写,这里用的是Flink1.11.0新的方法createTemporarySystemFunction ,registerFunction 方法已经被标记为废弃状态了.

2, flink.execution.jars加载指定的包

flink.execution.jars 所有的jar包都会load到flink interpreter的classpath里, 而且会被发送到Task Manager。这个配置主要是用来指定你的flink job所依赖的普通jar包.这种方式用起来很麻烦,因为你需要提前知道你有哪些UDF,还要知道UDF的包名,类名,如果你有多个UDF的话,还需要一个一个注册,使用起来很不方便.

```
package flink.udf

import org.apache.flink.table.functions.ScalarFunction

class MyScalaUdfDemo extends ScalarFunction {
    def eval(str: String) = str.toUpperCase
}
```

然后把项目打包上传到服务器上,这里需要注意的是打包的时候不要别的依赖打进去否则可能会遇到jar包冲突的问题.

在zeppelin里面加载jar包,然后注册.

```
%flink.conf
// 刚才打包上传到服务器的路径
flink.execution.jars /home/jason/bigdata/jar/flink-1.10.0-2020-07-22.jar
```

```
%flink
// 需要传入包名.类名
```

```
btenv.createTemporarySystemFunction("MyScalaUdfDemo1", new flink.udf.MyScalaUdfDei
```

这个感觉比第一种方式还麻烦,如果UDF比较多的情况下,前面两种方法都比较麻烦,需要一个个去注册,不要着急,接着看第三种方式.

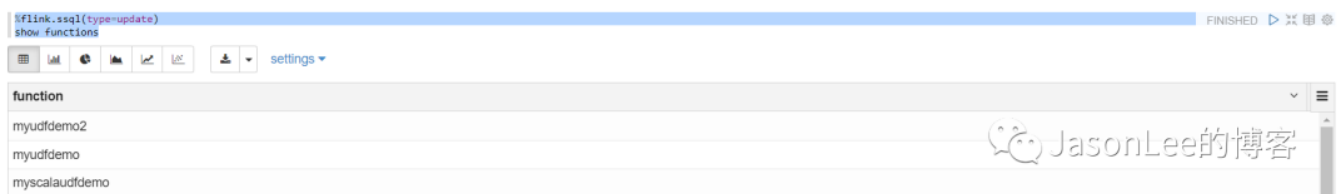
3,flink.udf.jars自动注册所有UDF

```
%flink.conf
flink.udf.jars /home/jason/bigdata/jar/flink-1.10.0-2020-07-22.jar
```

flink.udf.jars 这个配置和flink.execution.jars非常像,不同的地方在于Zeppelin会检测这些jar包中所包含的UDF class,而且会把他们注册到TableEnvironment中。UDF的名字就是这个class name。

上面的三种方法都可以实现注册自定义UDF,推荐使用第三种方法,如果UDF比较多的时候会自动注册,但是打包的时候需要注意jar包冲突问题.注册完了,那有没有成功了,下面来检测一下.

```
%flink.sql(type=update)
show functions
```



我这里用的是第三种方式,在代码里面写了3个UDF,两个java写的2,一个scala写的,虽然在zeppelin里面不支持java语言,但是可以用java编写UDF加载到Flink集群里面使用.

最后再来测试一下这些UDF能不能用呢？我就随便拿一个测试了。

```
%flink.sql
drop table if EXISTS kafka_table;

CREATE TABLE kafka_table (
  name VARCHAR COMMENT '姓名',
  age int COMMENT '年龄',
  city VARCHAR,
  birth VARCHAR,
  ts BIGINT COMMENT '时间戳',
  t as TO_TIMESTAMP(FROM_UNIXTIME(ts/1000,'yyyy-MM-dd HH:mm:ss')),
  proctime as PROCTIME(),
  WATERMARK FOR t AS t - INTERVAL '5' SECOND
)
WITH (
  'connector' = 'kafka', -- 使用 kafka connector
  'topic' = 'jason_flink', -- kafka topic
  'scan.startup.mode' = 'latest-offset', -- 从起始 offset 开始读取
  'properties.bootstrap.servers' = 'master:9092,storm1:9092,storm2:9092', -- broker连接信息
  'properties.group.id' = 'jason_flink_test',
  'scan.startup.mode' = 'latest-offset', -- 读取数据的位置
  'format' = 'json' -- 数据源格式为 json
)


Table has been dropped.
Table has been created.

Took 1 sec. Last updated by anonymous at July 23 2020, 4:19:39 AM.
```




创建一个kafka的表,然后查询一下,看下打印的结果。

```
%flink.sql(type=update)
select myscalaudfdemo(name) from kafka_table;
```



| EXPR\$0 |
|---------|
| JASON |
| JASON |
| JASON |
| JASON |
| JASON |
| SPARK |
| SPARK |
| SPARK |



可以看到字符串都变成大写的了,如果不想这么麻烦的话,直接执行下面的语句也可以测试,非常的方便。

```
%flink.sql(type=update)
select myscalaudfdemo('JasonLee');
```



| EXPR\$0 |
|----------|
| JASONLEE |



你可能会发现最近的几篇文章出现了flink.execution.jar, flink.udf.jar

flink.execution.packages 他们都是用来添加第三方依赖包的,这三个的区别是什么呢？最后再来总结一下：

flink.execution.jar 所有的jar包都会load到flink interpreter的classpath里，而且会被发送到Task Manager。这个配置主要是用来指定你的flink job所依赖的普通jar包。

flink.udf.jar 这个配置和flink.execution.jar非常像，不同的地方在于Zeppelin会检测这些jar包中所包含的UDF class，而且会把他们注册到TableEnvironment中。UDF的名字就是这个class name。

flink.execution.packages 这个配置也类似flink.execution.jar，但它不是用来指定jar包，而是用来指定package的。Zeppelin会下载这个package以及这个package的依

赖，并且放到flink interpreter的classpath上。比如你想使用kafka connector，那么你需要如下配置 `flink.execution.packages`成下面的样子