Flink-SQL-UDF(自定义函数)

主要讲三种udf：(SQL里可解释的Function)

- ScalarFunction
- TableFunction
- AggregateFunction

用户自定义函数是非常重要的一个特征，因为他极大地扩展了查询的表达能力.

1. 用户自定义函数在使用之前是必须要注册的。调用TableEnvironment的registerFunction()方法来实现注册。Udf注册成功之后，会被插入TableEnvironment的function catalog，这样table API和sql就能解析他了。

2. 用Runtime集成UDFs，有时候udf需要获取全局runtime信息或者在进行实际工作之前做一些设置和清除工作，比如，打开数据库链接和关闭数据库链接。Udf提供了open()和close()方法，可以被复写。

- Open()方法是在evaluation方法调用前调用一次。Close()是在evaluation方法最后一次调用后调用。
- Open()方法提共一个FunctionContext，FunctionContext包含了udf执行环境的上下文，比如，metric group，分布式缓存文件，全局的job参数。

  通过调用FunctionContext的相关方法，可以获取到相关的信息：

方法	描述
getMetricGroup()	并行子任务的指标组
getCachedFile(name)	分布式缓存文件的本
getJobParameter(name, defaultValue)	给定key全局job参数

3.例子：

```
public class HashCode extends ScalarFunction {
 private int factor = 12;
 public HashCode(int factor) {
     this.factor = factor;
 }
```

```
 public int eval(String s) {
        return s.hashCode() * factor;
 }

 @Override
 public void open(FunctionContext context) throws Exception {
       super.open(context);

    String redisHost = context.getJobParameter("redis.host","localhost");

    int redisPort = Integer.valueOf(context.getJobParameter("redis.port","6
       jedis = new Jedis(redisHost,redisPort);     }
 @Override
 public void close() throws Exception {
       super.close();
       jedis.close();
  }
}


BatchTableEnvironment tableEnv = TableEnvironment.getTableEnvironment(env);
// register the function
tableEnv.registerFunction("hashCode", new HashCode(10));
// use the function in Java Table API
myTable.select("string, string.hashCode(), hashCode(string)");
// use the function in SQL API
tableEnv.sqlQuery("SELECT string, HASHCODE(string) FROM MyTable");
```

一、Scalar Functions 标量函数

标量函数，是指返回一个值的函数。标量函数是实现将0，1，或者多个标量值转化为一个新值。标量函数的行为就是通过evaluation方法来实现的。evaluation方法必须定义为public，命名为eval。evaluation方法的输入参数类型和返回值类型决定着标量函数的输入参数类型和返回值类型。evaluation方法也可以被重载实现多个eval。默认情况下evaluation方法的返回值类型是由flink类型抽取工具决定。对于基础类型及简单的POJOS是足够的，但是更复杂的类型，自定义类型，组合类型，会报错。这种情况下，返回值类型的TypeInformation，需要手动指定，方法是重载ScalarFunction#getResultType()。

```
public class NvlUdf extends ScalarFunction {

/**
* 重载实现多个eval
*/
    public BigDecimal eval(BigDecimal value1, BigDecimal value2) {
        return Optional.ofNullable(value1).orElse(value2);
    }

    public BigDecimal eval(BigDecimal value1, Integer value2) {
```

```
        return Optional.ofNullable(value1).orElse(BigDecimal.valueOf(value2));
    }
                public Long eval(Long value1, Long value2) {
        return Optional.ofNullable(value1).orElse(value2);
    }

    public Long eval(Long value1, Integer value2) {
        return Optional.ofNullable(value1).orElse(value2.longValue());
    }

    public Integer eval(Integer value1, Integer value2) {
        return Optional.ofNullable(value1).orElse(value2);
    }
}
```

二、Table Functions 表函数

与标量函数相似之处是输入可以0，1，或者多个参数，但是不同之处可以输出任意数目的行数。返回的行也可以包含一个或者多个列。**用于联表判断条件。**

在Table API中，表函数在scala语言中使用方法如下：.join(Expression) 或者 .leftOuterJoin(Expression)，在java语言中使用方法如下：.join(String) 或者.leftOuterJoin(String)。

- Join操作算子会使用表函数(操作算子右边的表)产生的所有行进行(cross) join 外部表(操作算子左边的表)的每一行。
- leftOuterJoin操作算子会使用表函数(操作算子右边的表)产生的所有行进行(cross) join 外部表(操作算子左边的表)的每一行，并且在表函数返回一个空表的情况下会保留所有的outer rows。

```
public class Split extends TableFunction<Tuple2<String, Integer>> {

   private String separator = " ";
   public Split(String separator) {
       this.separator = separator;
   }
   public void eval(String str) {
       for (String s : str.split(separator)) {
           // use collect(...) to emit a row
           collect(new Tuple2<String, Integer>(s, s.length()));
       }
   }
}
```

```
BatchTableEnvironment tableEnv = TableEnvironment.getTableEnvironment(env);

Table myTable = ...         // table schema: [a: String]   // Register the func
tableEnv.registerFunction("split", new Split("#"));
// Use the table function in the Java Table API. "as" specifies the field nam
myTable.join("split(a) as (word, length)").select("a, word, length");

myTable.leftOuterJoin("split(a) as (word, length)").select("a, word, length")

// Use the table function in SQL with LATERAL and TABLE keywords.
// CROSS JOIN a table function (equivalent to "join" in Table API).
tableEnv.sqlQuery("SELECT a, word, length FROM MyTable, LATERAL TABLE(split(a
// LEFT JOIN a table function (equivalent to "leftOuterJoin" in Table API).
tableEnv.sqlQuery("SELECT a, word, length FROM MyTable LEFT JOIN LATERAL TABL
```

三、Aggregation Functions 聚合函数

用户自定义聚合函数聚合一张表(一行或者多行，一行有一个或者多个属性)为一个标量的值。

对于每个AggregateFunction，下面三个方法都是比不可少的：

- createAccumulator()
- accumulate()
- getValue()