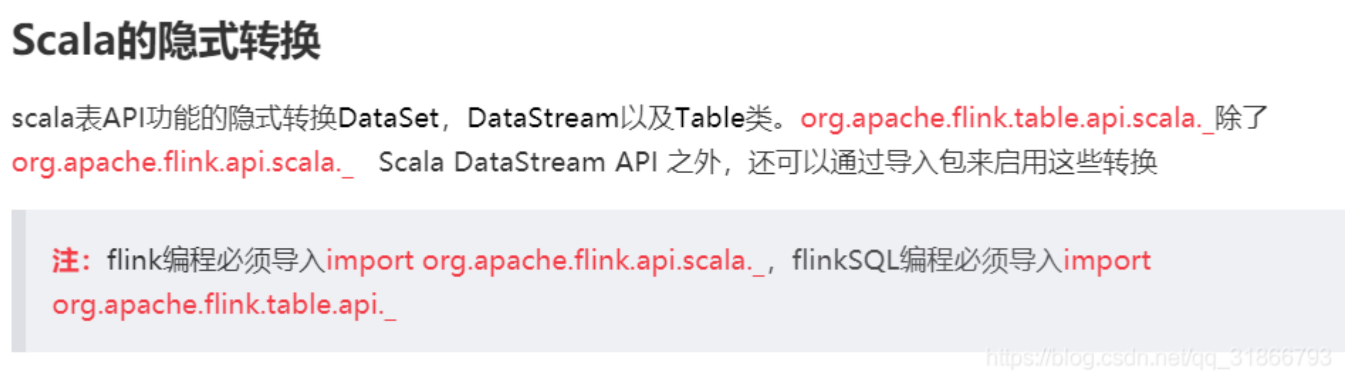
Flink原理实战每日一篇12 ---SQL 自定义函数

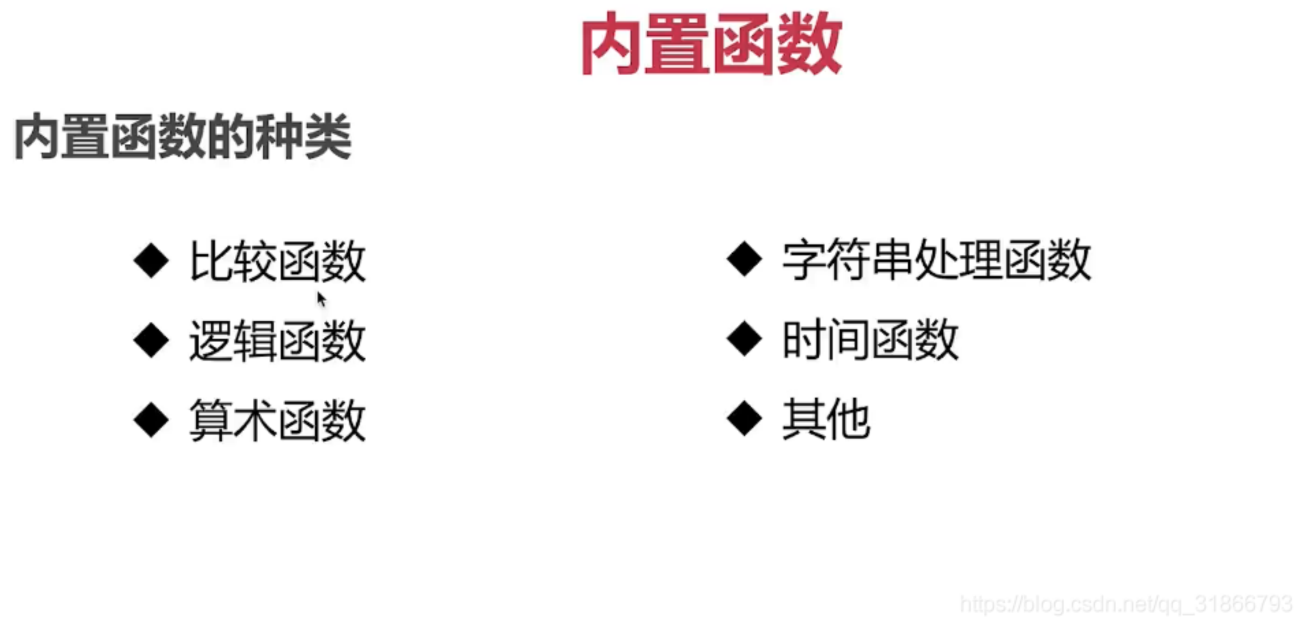


Scala的隐式转换

scala表API功能的隐式转换DataSet，DataStream以及Table类。org.apache.flink.table.api.scala._除了org.apache.flink.api.scala._ Scala DataStream API 之外，还可以通过导入包来启用这些转换

> 注：flink编程必须导入import org.apache.flink.api.scala._，flinkSQL编程必须导入import org.apache.flink.table.api._

一，自定义函数需要通过TableEnvironment 进行注册之后才可以使用，函数注册通过tableEnv.registerFuntion()方法完成



内置函数

内置函数的种类

- 比较函数
- 逻辑函数
- 算术函数
- 字符串处理函数
- 时间函数
- 其他

内置函数 官网API地址： https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/table/functions.html

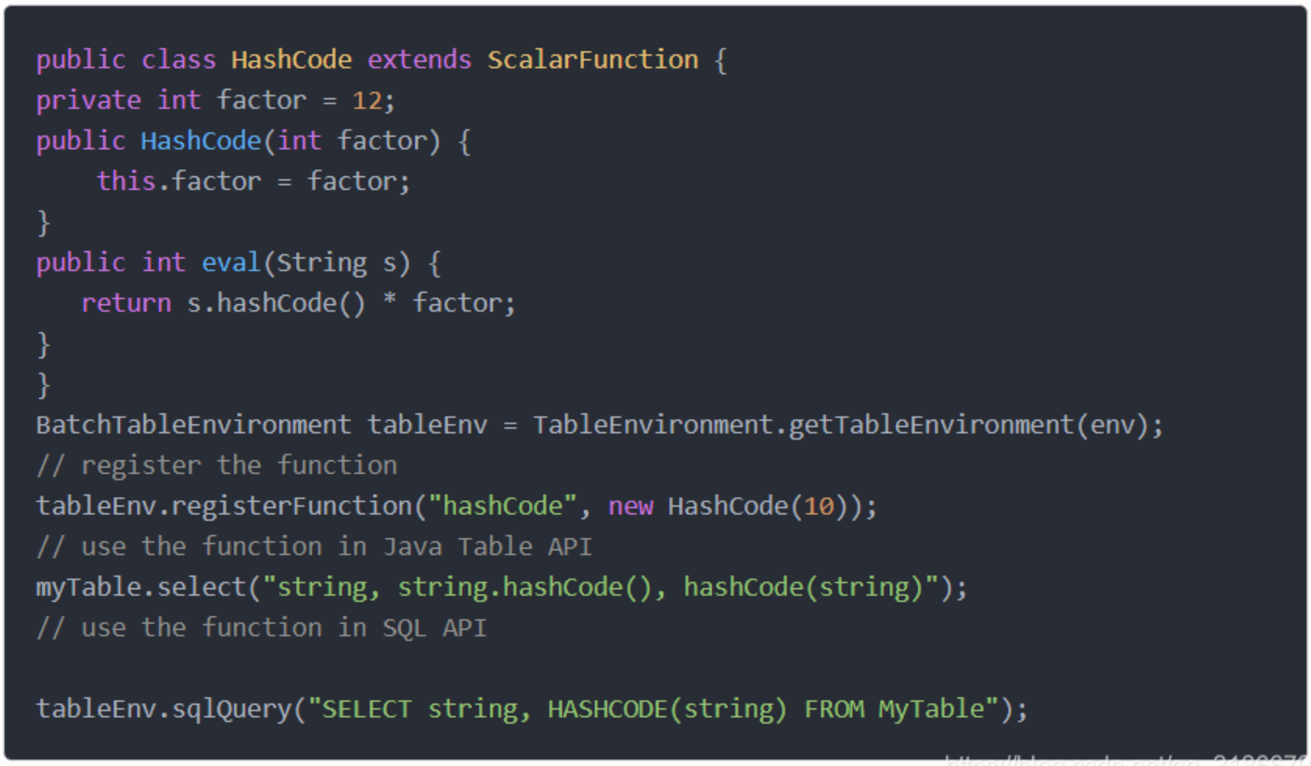
一，自定义函数分为三种

 1，Scalar Function --- 也被称为标量函数， 表示对单个输入或者多个输入字段计算后返回一个确定类型的标量值

  实现一个标量函数需要继承ScalarFunction，并且实现一个或者多个evaluation方法。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。同时evaluation方法支持变参数，例如：eval(String... strs)。

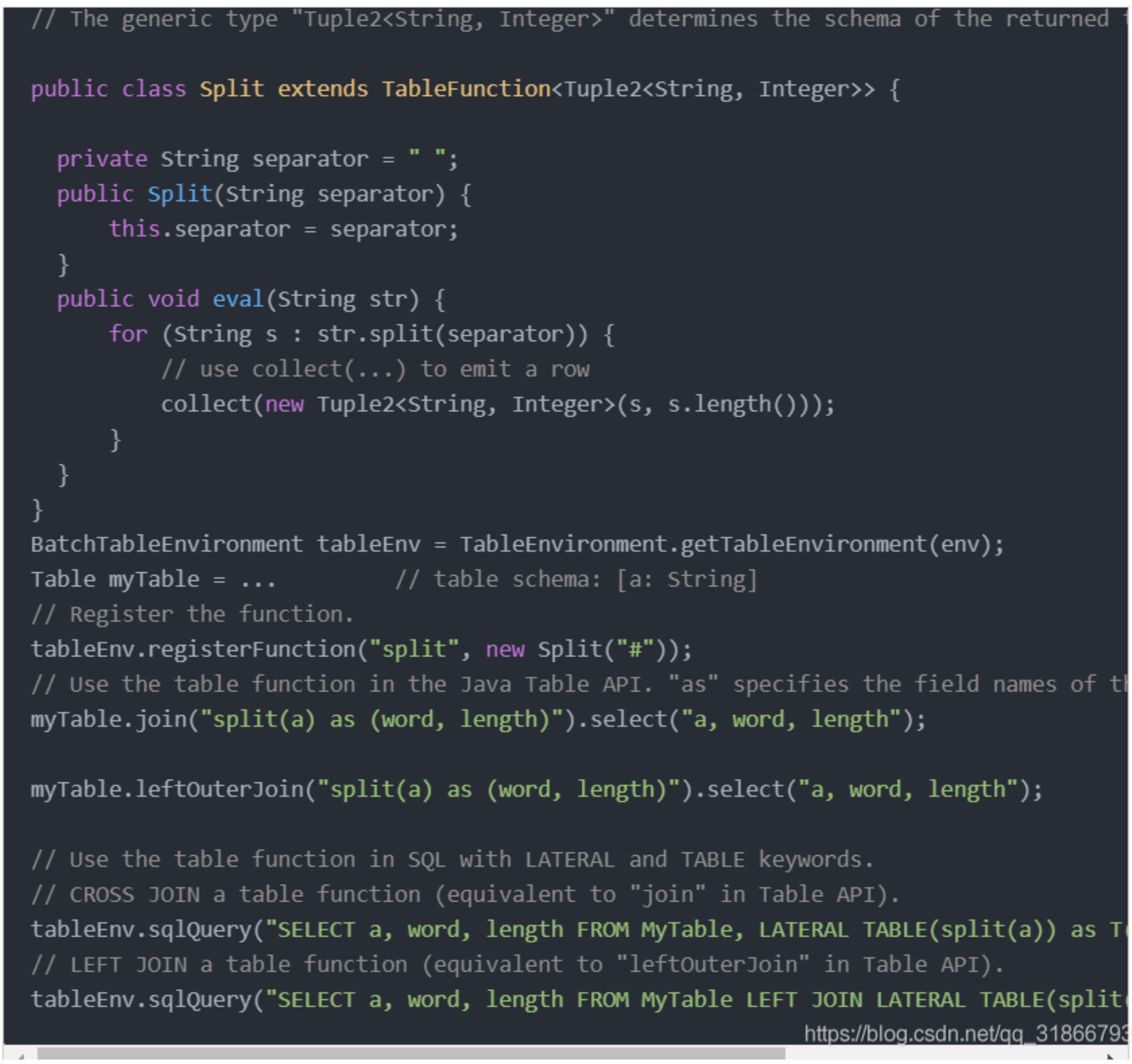
案例：



```
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;
}
}
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");
```

2，Table Function -- -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

为了自定义表函数，需要继承TableFunction，实现一个或者多个evaluation方法。表函数的行为定义在这些evaluation方法内部，函数名为eval并且必须是public。

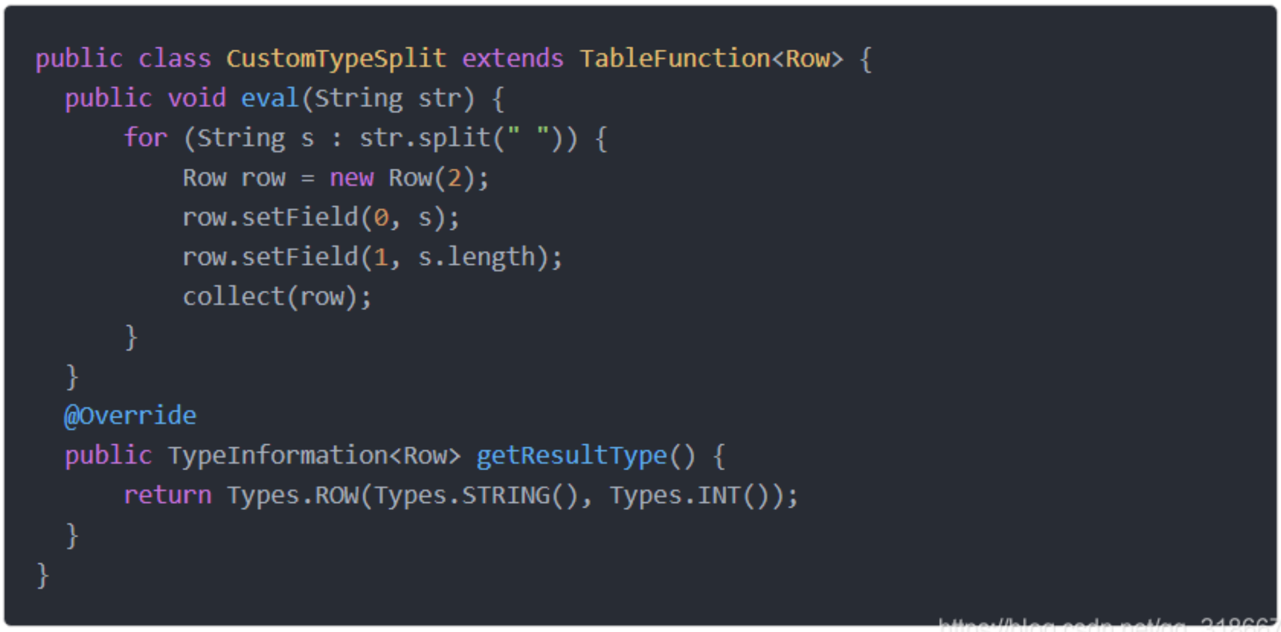


```
// The generic type "Tuple2<String, Integer>" determines the schema of the returned t

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 function.
tableEnv.registerFunction("split", new Split("#"));
// Use the table function in the Java Table API. "as" specifies the field names of th
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)) as T
// LEFT JOIN a table function (equivalent to "leftOuterJoin" in Table API).
tableEnv.sqlQuery("SELECT a, word, length FROM MyTable LEFT JOIN LATERAL TABLE(split
```

https://blog.csdn.net/qq_31866793

指定返回类型：

下面的例子，我们通过复写TableFunction#getResultType()方法使得表返回类型是RowTypeInfo(String, Integer)。



```
public class CustomTypeSplit extends TableFunction<Row> {
  public void eval(String str) {
      for (String s : str.split(" ")) {
          Row row = new Row(2);
          row.setField(0, s);
          row.setField(1, s.length);
          collect(row);
      }
  }
  @Override
  public TypeInformation<Row> getResultType() {
      return Types.ROW(Types.STRING(), Types.INT());
  }
}
```

3，Aggregation Funtion　---　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

为了计算加权平均值，累加器需要存储已累积的所有数据的加权和及计数。在栗子中定义一个WeightedAvgAccum类作为accumulator。尽管，retract(), merge(), 和resetAccumulator()方法在很多聚合类型是不需要的，这里也给出了栗子。

```
/**
* Accumulator for WeightedAvg.
*/
public static class WeightedAvgAccum {
  public long sum = 0;
  public int count = 0;
}
/**
* Weighted Average user-defined aggregate function.
*/
public static class WeightedAvg extends AggregateFunction<Long,
WeightedAvgAccum> {
  @Override
  public WeightedAvgAccum createAccumulator() {
    return new WeightedAvgAccum();
  }
  @Override
  public Long getValue(WeightedAvgAccum acc) {
      if (acc.count == 0) {
          return null;
      } else {
          return acc.sum / acc.count;
      }
  }
  public void accumulate(WeightedAvgAccum acc, long iValue, int iWeight) {
      acc.sum += iValue * iWeight;
      acc.count += iWeight;
  }
  public void retract(WeightedAvgAccum acc, long iValue, int iWeight) {
      acc.sum -= iValue * iWeight;
      acc.count -= iWeight;
  }
  public void merge(WeightedAvgAccum acc, Iterable<WeightedAvgAccum> it) {
      Iterator<WeightedAvgAccum> iter = it.iterator();
      while (iter.hasNext()) {
          WeightedAvgAccum a = iter.next();
          acc.count += a.count;
          acc.sum += a.sum;
      }
  }
  public void resetAccumulator(WeightedAvgAccum acc) {
      acc.count = 0;
      acc.sum = 0L;
  }
}
// register function
StreamTableEnvironment tEnv = ...
tEnv.registerFunction("wAvg", new WeightedAvg());
// use function

tEnv.sqlQuery("SELECT user, wAvg(points, level) AS avgPoints FROM
userScores GROUP BY user");
```

有时候UDF需要获取全局运行时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提供了开放（）和关闭（）方法，可以被复写，功能类似数据集和DataStream API的RichFunction方法。

Open(): 是在eval()方法调用前调用一次,Open（）方法提共一个FunctionContext，FunctionContext包含了udf执行环境的上 下文，比如，公制组，分布式缓存文件，全局的工作参数。
Close():是在评估方法最后一次调用后调用.
通过调用FunctionContext的相关方法，可以获取到相关的信息：

getMetricGroup（）并行子任务的指标组;
getCachedFile（名称）分布式缓存文件的本地副本;
getJobParameter（name，defaultValue）给定键全局作业参数;
案例：通过FunctionContext在一个标量函数中获取全局工作的参数。主要是实现获取的Redis的配置，然后获取Redis的链接，实现Redis的的交互的过程。

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
import redis.clients.jedis.Jedis;
public class HashCode extends ScalarFunction {
  private int factor = 12;
  Jedis jedis = null;
  public HashCode() {
      super();
  }
  @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","6379"));
      jedis = new Jedis(redisHost,redisPort);
  }

  @Override
  public void close() throws Exception {
      super.close();
      jedis.close();
  }

  public HashCode(int factor) {
      this.factor = factor;
  }

  public int eval(int s) {
      s = s % 3;
      if(s == 2)
         return Integer.valueOf(jedis.get(String.valueOf(s)));
      else
         return 0;
   }
}

ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
BatchTableEnvironment tableEnv = TableEnvironment.getTableEnvironment(env);
// set job parameter
Map<String,String> hashmap = new HashMap<>();
      hashmap.put("redis.host","localhost");
      hashmap.put("redis.port","6379");
      ParameterTool parameter = ParameterTool.fromMap(hashmap);
      exeEnv.getConfig().setGlobalJobParameters(parameter);
// register the function
tableEnv.registerFunction("hashCode", new HashCode());
// use the function in Java Table API
myTable.select("string, string.hashCode(), hashCode(string)");
// use the function in SQL
tableEnv.sqlQuery("SELECT string, HASHCODE(string) FROM MyTable");
```

最后简单案例：

```
package com.coder.flink.core.table_sql.sql_function

import org.apache.flink.api.scala._
import org.apache.flink.table.api.scala._
import org.apache.flink.table.api.{Table, TableEnvironment}
import org.apache.flink.types.Row
object testScalarDemo {
  def main(args: Array[String]): Unit = {
    val bEnv =ExecutionEnvironment.getExecutionEnvironment
    val  tableEnv = TableEnvironment.getTableEnvironment(bEnv)
    //注册函数
    tableEnv.registerFunction("add", new TestHashCode(10))
    val batch = bEnv.fromElements(("aa","11"))
    tableEnv.registerDataSet("table1",batch)
    tableEnv.registerDataSet("table2",batch,'field222,'field111)
    val table: Table = tableEnv.sqlQuery("SELECT field222, field111 FROM
table2")
//    table.toDataSet[(String,Int)].print()
   val rs =  tableEnv.sqlQuery("SELECT field222, add(field111) FROM
table2").toDataSet[Row]
//    rs.count()
    rs.print()
//    bEnv.execute("aa")

  }
}
```

```
package com.coder.flink.core.table_sql.sql_function;


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

public class TestHashCode extends ScalarFunction {

    private int factor = 12;

    public TestHashCode(int factor) {
        this.factor = factor;
    }

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

    @Override
    public int hashCode() {
        return super.hashCode();
    }
}
```