### 广播变量

（1）广播变量允许编程人员在每台机器上保持1个只读的缓存变量，而不是传送变量的副本给tasks

（2）广播变量创建后，它可以运行在集群中的任何function上，而不需要多次传递给集群节点。另外需要记住，不应该修改广播变量，这样才能确保每个节点获取到的值都是一致的。可以理解为是一个公共的共享变量，我们可以把一个dataset 数据集广播出去，然后不同的task在节点上都能够获取到，这个数据在每个节点上只会存在一份。如果不使用broadcast，则在每个节点中的每个task中都需要拷贝一份dataset数据集，比较浪费内存(也就是一个节点中可能会存在多份dataset数据)。

1. 与DataStreaming 中的Broadcast区别开来，DataStreaming 中的Broadcast是把元素广播给所有的分区，数据会被重复处理，类似于storm中的allGrouping（调用方式 dataStream.broadcast()）

### 语法

(1)：初始化数据 DataSet<Integer> toBroadcast = env.fromElements(1, 2, 3)

(2)：广播数据 withBroadcastSet(toBroadcast, "broadcastSetName");

(3)：获取数据 Collection<Integer> broadcastSet = getRuntimeContext().getBroadcastVariable("broadcastSetName");

注意：

(1)广播出去的变量存在于每个节点的内存中，所以这个数据集不能太大。因为广播出去的数据，会常驻内存，除非程序执行结束

(2)广播变量在初始化广播出去以后不支持修改，这样才能保证每个节点的数据都是一致的。

### 开发

**package** com.zrr;  
  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.api.common.functions.RichMapFunction;  
**import** org.apache.flink.api.java.DataSet;  
**import** org.apache.flink.api.java.ExecutionEnvironment;  
**import** org.apache.flink.api.java.operators.MapOperator;  
**import** org.apache.flink.api.java.tuple.Tuple2;  
**import** org.apache.flink.configuration.Configuration;  
  
**import** java.util.ArrayList;  
**import** java.util.HashMap;  
**import** java.util.List;  
  
**public class** ds5 {  
 **public static void** main(String[] args) **throws** Exception {  
 ExecutionEnvironment env = ExecutionEnvironment.*getExecutionEnvironment*();  
  
  
 List<Tuple2<String, String>> b1 = **new** ArrayList<>();  
 b1.add(**new** Tuple2<>(**"101"**, **"zs"**));  
 b1.add(**new** Tuple2<>(**"102"**, **"ls"**));  
 b1.add(**new** Tuple2<>(**"103"**, **"ww"**));  
  
 DataSet<Tuple2<String, String>> tuple2broadCastData = env.fromCollection(b1);  
  
 DataSet<HashMap<String, String>> sd1 = tuple2broadCastData.map(**new** MapFunction<Tuple2<String, String>, HashMap<String, String>>() {  
 @Override  
 **public** HashMap<String, String> map(Tuple2<String, String> value) **throws** Exception {  
 HashMap<String, String> map = **new** HashMap<>();  
 map.put(value.**f0**, value.**f1**);  
 **return** map;  
 }  
 });  
 List<Tuple2<String, Integer>> l2 = **new** ArrayList<>();  
 l2.add(**new** Tuple2<>(**"101"**, 10000));  
 l2.add(**new** Tuple2<>(**"102"**, 20000));  
 l2.add(**new** Tuple2<>(**"103"**, 30000));  
 DataSet<Tuple2<String, Integer>> sd3 = env.fromCollection(l2);  
  
 DataSet<String> result = sd3.map(**new** RichMapFunction<Tuple2<String, Integer>, String>() {  
 List<HashMap<String, String>> **broadCastMap** = **new** ArrayList<HashMap<String, String>>();  
 HashMap<String, String> **allMap** = **new** HashMap<String, String>();  
  
 @Override  
 **public void** open(Configuration parameters) **throws** Exception {  
 **this**.**broadCastMap** = getRuntimeContext().getBroadcastVariable(**"bm"**);  
 **for** (HashMap<String, String> map : **broadCastMap**) {  
 **allMap**.putAll(map);  
 }  
 }  
  
 @Override  
 **public** String map(Tuple2<String, Integer> t2) **throws** Exception {  
 String name = **allMap**.get(t2.**f0**);  
 **return** name + **","** + t2.**f1**;  
 }  
 }).withBroadcastSet(sd1, **"bm"**);  
  
 result.print();  
 }  
}