### StateBackend

默认情况下，state会保存在taskmanager的内存中，checkpoint会存储在JobManager的内存中。

state 的store和checkpoint的位置取决于State Backend的配置（env.setStateBackend(…)）

一共有三种State Backend：MemoryStateBackend、FsStateBackend、RocksDBStateBackend

（1）MemoryStateBackend：state数据保存在java堆内存中，执行checkpoint的时候，会把state的快照数据保存到jobmanager的内存中，基于内存的state backend在生产环境下不建议使用

（2）FsStateBackend：state数据保存在taskmanager的内存中，执行checkpoint的时候，会把state的快照数据保存到配置的文件系统中，可以使用hdfs等分布式文件系统

（3）RocksDBStateBackend：RocksDB跟上面的都略有不同，它会在本地文件系统中维护状态，state会直接写入本地rocksdb中。同时它需要配置一个远端的filesystem uri（一般是HDFS），在做checkpoint的时候，会把本地的数据直接复制到filesystem中。fail over的时候从filesystem中恢复到本地。RocksDB克服了state受内存限制的缺点，同时又能够持久化到远端文件系统中，比较适合在生产中使用

### FsStateBackend

env.setStateBackend(**new** FsStateBackend(**"hdfs://192.168.8.201:9000/flink1"**));

开发：

**package** com.zr3;  
  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.api.common.functions.RichFlatMapFunction;  
**import** org.apache.flink.api.common.state.StateTtlConfig;  
**import** org.apache.flink.api.common.state.ValueState;  
**import** org.apache.flink.api.common.state.ValueStateDescriptor;  
**import** org.apache.flink.api.common.typeinfo.TypeHint;  
**import** org.apache.flink.api.common.typeinfo.TypeInformation;  
**import** org.apache.flink.api.java.tuple.Tuple2;  
**import** org.apache.flink.configuration.Configuration;  
**import** org.apache.flink.runtime.state.filesystem.FsStateBackend;  
**import** org.apache.flink.streaming.api.TimeCharacteristic;  
**import** org.apache.flink.streaming.api.datastream.DataStream;  
**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
  
**import** org.apache.flink.util.Collector;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** hw6 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.enableCheckpointing(2000);  
 env.setStateBackend(**new** FsStateBackend(**"file:///d://ff123"**));  
  
 DataStream<Tuple2<Long,Long>> dd1 =env.socketTextStream(**"192.168.8.66"**,9999).map(**new** MapFunction<String, Tuple2<Long, Long>>() {  
 @Override  
 **public** Tuple2<Long, Long> map(String value) **throws** Exception {  
 **return new** Tuple2<Long,Long>(Long.*parseLong*(value.split(**" "**)[0]),Long.*parseLong*(value.split(**" "**)[1]));  
 }  
 });  
 dd1.keyBy(0).flatMap(**new** avg123()).print();  
 env.execute(**"avg"**);  
  
 }  
 **public static class** avg123 **extends** RichFlatMapFunction<Tuple2<Long,Long>,Tuple2<Long,Long>>{  
 ValueState<Tuple2<Long,Long>> **sum**;  
 @Override  
 **public void** flatMap(Tuple2<Long, Long> value, Collector<Tuple2<Long, Long>> out) **throws** Exception {  
 Tuple2<Long,Long> cSum= **sum**.value();  
 cSum.**f0** +=1;  
 cSum.**f1** += value.**f1**;  
 **sum**.update(cSum);  
 **if**(cSum.**f0**>=2){  
 out.collect(**new** Tuple2<Long,Long>(value.**f0**,cSum.**f1**/cSum.**f0**));  
 *//sum.clear();* }  
  
 }  
 @Override  
 **public void** open(Configuration parameters) **throws** Exception{  
 ValueStateDescriptor<Tuple2<Long,Long>> desc1 = **new** ValueStateDescriptor<Tuple2<Long, Long>>(  
 **"avg"**, TypeInformation.*of*(**new** TypeHint<Tuple2<Long, Long>>() {}),Tuple2.*of*(0L,0L)  
 );  
 **sum** = getRuntimeContext().getState(desc1);  
 }  
  
 }  
}

保存到本地文件系统

env.setStateBackend(new FsStateBackend("file:///D:/flink1"));

### 3.RocksDBStateBackend

Pom.xml

<**dependency**>  
 <**groupId**>org.apache.flink</**groupId**>  
 <**artifactId**>flink-statebackend-rocksdb\_2.11</**artifactId**>  
 <**version**>1.7.2</**version**>  
</**dependency**>

添加配置：

env.setStateBackend(**new** RocksDBStateBackend(**"hdfs://192.168.8.201：9000/flink2"**));