Flink Queryable State实践

简介

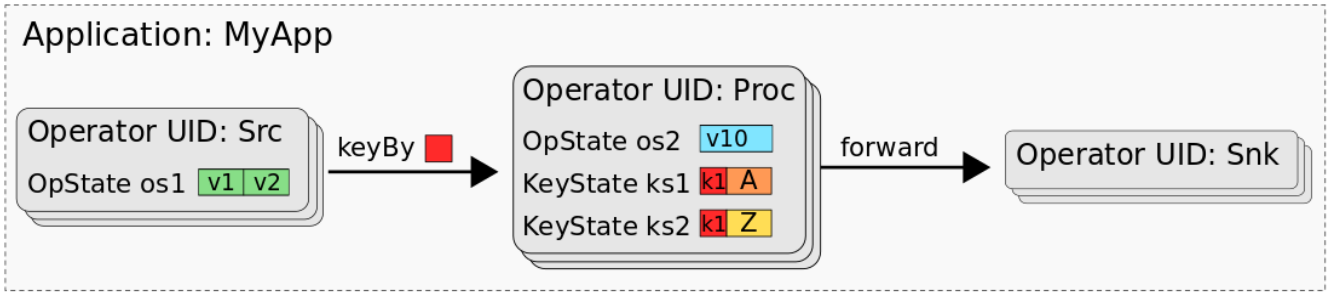
Queryable State API主要作用是将flink管理的键控状态暴露给外部，并允许用户从flink外部查询作业的状态。某些情况下，可查询状态消除了对外部系统（例如键值存储）的分布式操作/事务的需要，这通常是实践中的瓶颈。最后，可查询状态可以为实时作业的调试提供便利。

注意：可查询状态的客户端API当前处于不断发展的状态，并且不保证所提供接口的稳定性。在即将推出的Flink版本中，客户端可能会发生重大的API更改。查询状态对象时，无需任何同步或复制即可从并发线程访问该对象。这是一种设计选择，因为上述任何一种都会导致增加的作业延迟，我们希望避免这种情况。状态后端使用Java堆空间的状态，例如MemoryStateBackend或FsStateBackend在检索值时不能与副本一起使用，而是直接引用存储的值，读取 - 修改 - 写入模式是不安全的，并且可能导致可查询状态服务器由于并发修改而失败。 RocksDBStateBackend可以避免这些问题。

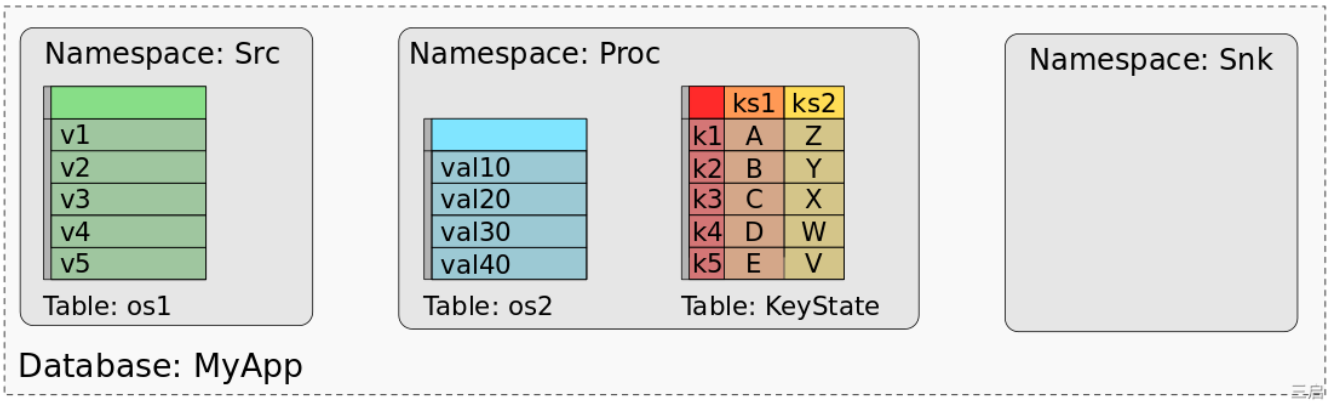
理论背景

有状态的flink job由operator组成，通常是一个或多个source operator，一些实际处理的operator以及一个或多个sink operator。每个operator在任务中并行运行，并且可以使用不同类型的状态。operator可以具有零个，一个或多个“operator状态”，这些状态被组织为以operator任务为范围的列表。如果将operator应用于keyed stream，它还可以具有零个，一个或多个“键控状态”，它们的作用域范围是从每个已处理记录中提取的键。你可以将键控状态视为分布式键-值映射。

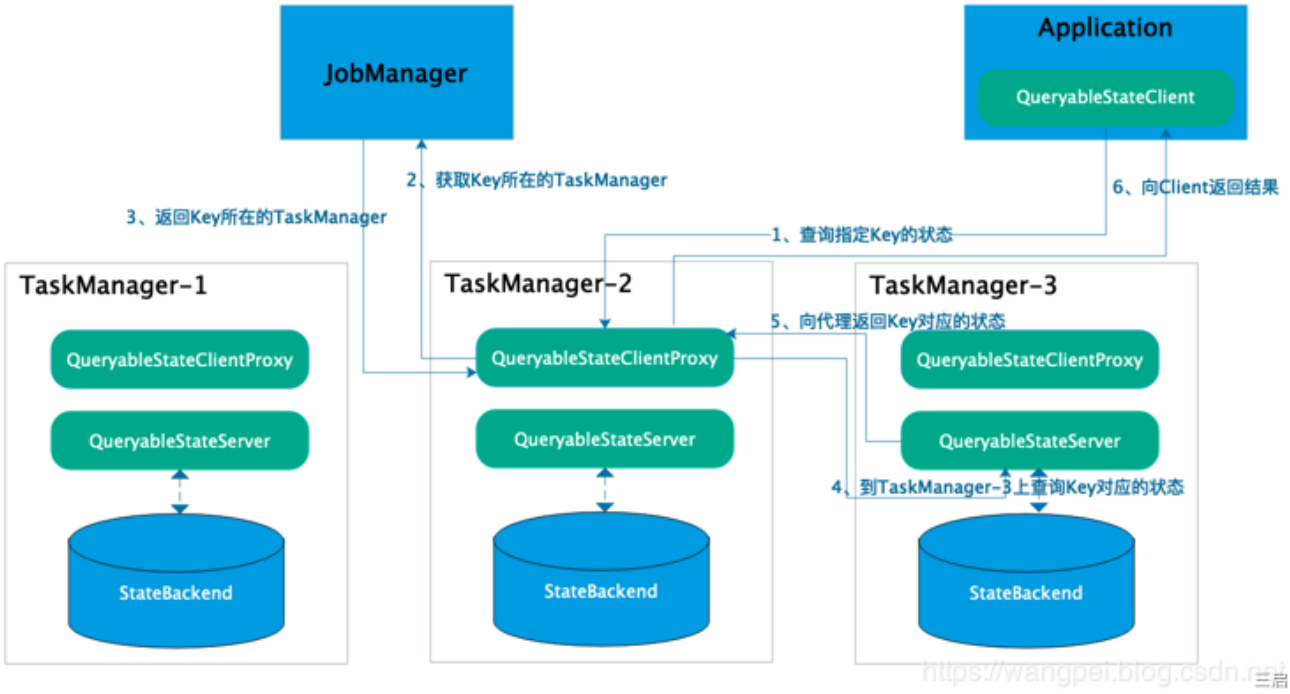
如下图所示，Flink job MyApp，该job由Src、Proc、Snk三个operator组成，Src有一个operator state，Proc有一个operator state和两个keyed state，Snk是无状态的。



MyApp的保存点或检查点由所有状态的数据组成，这些数据的组织方式可以恢复每个任务的状态。我们可以用如下思维模型，将每个任务状态的数据映射到数据集或表中。 实际上，我们可以将保存点视为数据库。 每个运算符（由其UID标识）代表一个名称Namespace。 operator的每个operator state都通过一个列映射到名称空间中的专用表，该列包含所有任务的状态数据。 operator的所有键状态都映射到一个表，该表由用于键的列和用于每个键状态的一列组成。 下图显示了MyApp的保存点如何映射到数据库。



服务架构



可查询状态由以下三个组件组成：

- QueryableStateClient: 客户端。运行在外部系统。提交查询请求并接收最终返回的结果。
- QueryableStateClientProxy: 客户端代理。运行在每个TaskManager上。接收客户端的请求，找到Key对应的TaskManager，然后将请求转发给具体的查询服务，并负责最终向客户端返回结果。
- QueryableStateServer: 查询服务。运行在每个TaskManager上。处理来自客户端代理的请求并返回结果。

激活Queryable State服务

可查询状态在flink发行版中并非是默认开启的，所有需要相应的配置才能启用。

Flink on Yarn/Flink Standalone

1. 添加依赖

```
cp ${FLINK_HOME}/opt/flink-queryable-state-runtime_2.11-1.9.0.jar  ${FLINK_HOME}/lib/
```

2. 启用Queryable State服务

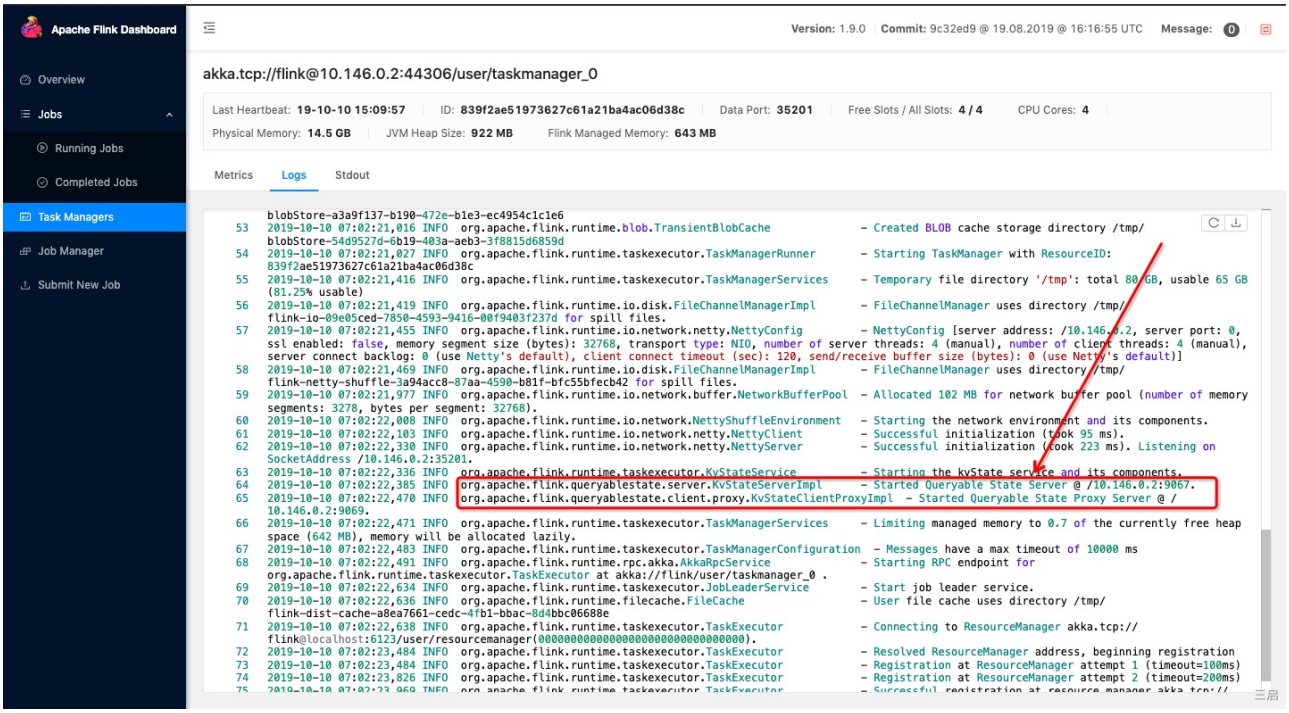
   在`${FLINK_HOME}/conf/flink-conf.yaml`中设置`queryable-state.enable: true`

3. 验证服务启动状态

   查看TaskManager日志，在日志中见到如下内容，则表示可查询状态启用成功

```
Started Queryable State Server @ /x.x.x.x:9067.
 Started Queryable State Proxy Server @ /x.x.x.x:9069
```

启动flink后，可以访问flink web UI查看TaskManager日志，如



Flink on Idea

1. 添加依赖

```
<dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-queryable-state-runtime_2.11</artifactId>
     <version>1.9.0</version>
 </dependency>
```

2. 获取启用Queryable State服务的StreamEnvironment

```
Configuration config = new Configuration();
 config.setInteger(ConfigOptions.key("rest.port").defaultValue(8081),8081);
 config.setBoolean(ConfigConstants.LOCAL_START_WEBSERVER, true);
 //启用Queryable State服务
 config.setBoolean(QueryableStateOptions.ENABLE_QUERYABLE_STATE_PROXY_SERVER, t
```

```
rue);
 StreamExecutionEnvironment env = StreamExecutionEnvironment.createLocalEnviron
mentWithWebUI(config);
 ...
 env.execute("JobName");
```

3. 验证服务启动状态
   flink在Idea环境运行时会启动一个mini cluster。打开INFO日志，查找如下日志：

```
Started Queryable State Server @ /127.0.0.1:9067.
 Started Queryable State Proxy Server @ /127.0.0.1:9069
```

如果有相关日志，则表示该flink job可查询状态启用成功。

使状态可查询

通过以上设置，已在Flink集群上激活了可查询状态服务，除此之外，还需要在代码中暴露具体的可查询状态。有两种方式:

- 将DataStream转换为QueryableStateStream。如将KeyedStream转换QueryableStateStream，即可设定KeyedStream中所有Key的State可查。
- 通过状态描述StateDescriptor的setQueryable(String queryableStateName)方法，可设定某个Key的State可查。

相关依赖

- server端依赖

```
<dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-core</artifactId>
     <version>${flink.version}</version>
 </dependency>
 <dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-streaming-java_2.11</artifactId>
     <version>${flink.version}</version>
 </dependency>
 <dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-runtime-web_2.11</artifactId>
     <version>${flink.version}</version>
 </dependency>
 <dependency>
     <groupId>org.apache.flink</groupId>
```

```
    <artifactId>flink-queryable-state-runtime_2.11</artifactId>
    <version>${flink.version}</version>
 </dependency>
```

- client端依赖

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-queryable-state-client-java</artifactId>
    <version>1.9.0</version>
 </dependency>
```

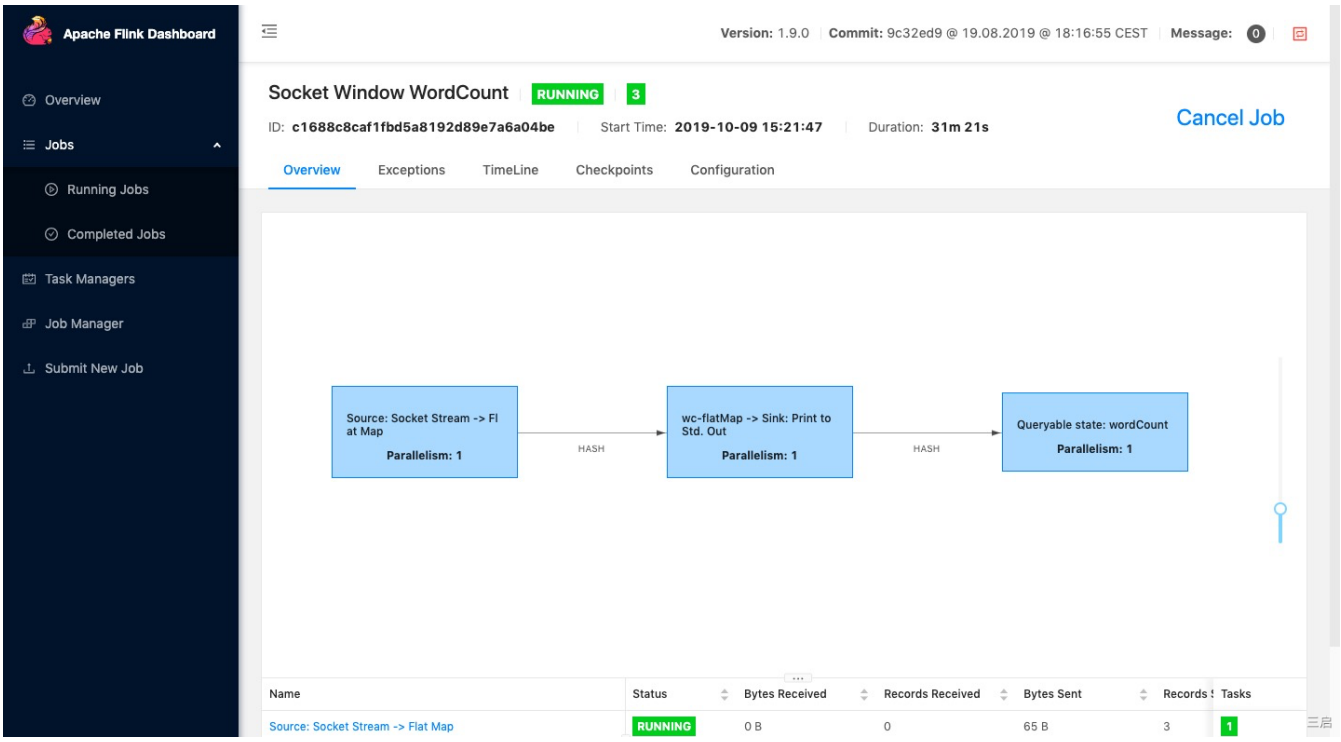
QueryableStateStream

- Server端（Flink job）

```
package com.flink.state.queryable;

 import org.apache.flink.api.common.functions.FlatMapFunction;
 import org.apache.flink.api.java.tuple.Tuple2;
 import org.apache.flink.configuration.ConfigConstants;
 import org.apache.flink.configuration.ConfigOptions;
 import org.apache.flink.configuration.Configuration;
 import org.apache.flink.configuration.QueryableStateOptions;
 import org.apache.flink.streaming.api.CheckpointingMode;
 import org.apache.flink.streaming.api.datastream.DataStream;
 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
 import org.apache.flink.util.Collector;

 /**
  * @author : 恋晨
  * Date   : 2019/10/9 2:40 PM
  * 功能   :  Flink 可查询状态示例job
  *          使用将DataStream转换为QueryableStateStream使得状态可查询
  */
 public class WordCountOnSetStream {
     public static void main(String[] args) throws Exception{

         final String hostname = "localhost";
         final int port = 9000;

         Configuration config = new Configuration();
         config.setInteger(ConfigOptions.key("rest.port").defaultValue(8081),80
81);
         config.setBoolean(ConfigConstants.LOCAL_START_WEBSERVER, true);
```

```
        /**启用Queryable State服务*/
        config.setBoolean(QueryableStateOptions.ENABLE_QUERYABLE_STATE_PROXY_S
ERVER, true);
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.crea
teLocalEnvironmentWithWebUI(config);

        env.setParallelism(1);
        env.enableCheckpointing(1000 , CheckpointingMode.AT_LEAST_ONCE);
        env.getCheckpointConfig().setMinPauseBetweenCheckpoints(30000);


        DataStream<String> text = env.socketTextStream(hostname, port, "\n");

        DataStream<Tuple2<String , Long>> windowCounts = text
                .flatMap(new FlatMapFunction<String, Tuple2<String, Long>>() {
                    @Override
                    public void flatMap(String value, Collector<Tuple2<String,
 Long>> out) throws Exception {
                        out.collect(new Tuple2<String,Long>(value ,1L));
                    }
                })
                .keyBy(0)
                .sum(1);

        windowCounts.print();


        windowCounts.keyBy(0).asQueryableState("QueryableState_WordCount");


        env.execute("Socket Window WordCount");
    }


}
```

运行该job，访问flink web UI链接
可以查看该任务的jobID、DAG等等信息。



- Client端

```
package com.flink.state.queryable;

 import org.apache.flink.api.common.JobID;
 import org.apache.flink.api.common.state.ValueState;
 import org.apache.flink.api.common.state.ValueStateDescriptor;
 import org.apache.flink.api.common.typeinfo.BasicTypeInfo;
 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.queryablestate.client.QueryableStateClient;

 import java.util.concurrent.CompletableFuture;

 /**
  * @author : 恋晨
  * Date   : 2019/10/9 4:05 PM
  * 功能   :    状态查询客户端示例
  */
 public class QueryClient {

     public static void main(String[] args) throws Exception{

         final JobID jobID = JobID.fromHexString("5dc4c2765c46664e0c121855baf2d
da8");
         final String hostname = "localhost";
         final int port = 9069;

         QueryableStateClient client = new QueryableStateClient(hostname , port
```

```
);

        ValueStateDescriptor<Tuple2<String, Long>> descriptor =
                new ValueStateDescriptor<>(
                        "QueryableState_WordCount",
                        TypeInformation.of(new TypeHint<Tuple2<String, Long>>(
) {}));
        final String key = "hello";

        CompletableFuture<ValueState<Tuple2<String , Long>>> completableFuture
 =
                client.getKvState(
                        jobID,
                        "QueryableState_WordCount",
                        key,
                        BasicTypeInfo.STRING_TYPE_INFO,
                        descriptor
                );
        System.out.println(completableFuture.get().value());
    }
 }
```

setQueryable

- Server端(flink job)

```
package com.flink.state.queryable;

 import org.apache.flink.api.common.functions.FlatMapFunction;
 import org.apache.flink.api.common.functions.RichFlatMapFunction;
 import org.apache.flink.api.common.state.ValueState;
 import org.apache.flink.api.common.state.ValueStateDescriptor;
 import org.apache.flink.api.common.typeinfo.Types;
 import org.apache.flink.api.java.tuple.Tuple2;
 import org.apache.flink.configuration.ConfigConstants;
 import org.apache.flink.configuration.ConfigOptions;
 import org.apache.flink.configuration.Configuration;
 import org.apache.flink.configuration.QueryableStateOptions;
 import org.apache.flink.streaming.api.CheckpointingMode;
 import org.apache.flink.streaming.api.datastream.DataStream;
 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
 import org.apache.flink.util.Collector;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;

 /**
  * @author : 恋晨
```

```
 * Date   : 2019/10/10 10:49 AM
 * 功能   :  flink 可查询状态示例程序
 *          通过对state setQueryable实现状态可查询
 */
public class WordCountSetQueryable {

    final static Logger log = LoggerFactory.getLogger("WordCountSetQueryable")
;

    public static void main(String[] args) throws Exception{

        final String hostname = "localhost";
        final int port = 9009;

        Configuration config = new Configuration();
        config.setInteger(ConfigOptions.key("rest.port").defaultValue(8081),80
81);
        config.setBoolean(ConfigConstants.LOCAL_START_WEBSERVER, true);
        /**启用Queryable State服务*/
        config.setBoolean(QueryableStateOptions.ENABLE_QUERYABLE_STATE_PROXY_S
ERVER, true);
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.crea
teLocalEnvironmentWithWebUI(config);

        env.setParallelism(1);
        env.enableCheckpointing(1000 , CheckpointingMode.AT_LEAST_ONCE);
        env.getCheckpointConfig().setMinPauseBetweenCheckpoints(30000);


        DataStream<String> text = env.socketTextStream(hostname, port, "\n");

        DataStream<Tuple2<String , Long>> windowCounts = text
                .flatMap(new FlatMapFunction<String, Tuple2<String, Long>>() {
                    @Override
                    public void flatMap(String value, Collector<Tuple2<String,
 Long>> out) throws Exception {
                        out.collect(new Tuple2<String,Long>(value ,1L));
                    }
                })
                .keyBy(0)
                .flatMap(new RichFlatMapFunction<Tuple2<String, Long>, Tuple2<
String, Long>>() {
                    public  transient ValueState<Tuple2<String,Long>> countSta
te;

                    @Override
                    public void open(Configuration parameters) throws Exceptio
n {
                        ValueStateDescriptor<Tuple2<String,Long>> valueStateDe
```

```
scriptor =
                                    new ValueStateDescriptor<Tuple2<String, Long>>
(
                                            "QueryableState_WordCount_state",
                                            Types.TUPLE(Types.STRING , Types.LONG)
,
                                            new Tuple2<>(null , 0L)
                                    );

                            /**通过ValueStateDescriptor.setQueryable 开放此状态*/
                            valueStateDescriptor.setQueryable("QueryableState_Word
Count");
                            countState = getRuntimeContext().getState(valueStateDe
scriptor);


                        }

                        @Override
                        public void flatMap(Tuple2<String, Long> value, Collector<
Tuple2<String, Long>> out) throws Exception {

                            Tuple2<String,Long> currentState = countState.value();
                            if(currentState.f0 == null){
                                currentState.f0 = value.f0;
                            }

                            currentState.f1 += value.f1;

                            countState.update(currentState);

                            log.info(currentState.toString());

                            out.collect(currentState);

                        }
                    });

            windowCounts.print();


            env.execute("Socket Window WordCount");
        }

    }
```

- Client端

```
package com.flink.state.queryable;

 import org.apache.flink.api.common.JobID;
 import org.apache.flink.api.common.state.ValueState;
 import org.apache.flink.api.common.state.ValueStateDescriptor;
 import org.apache.flink.api.common.typeinfo.BasicTypeInfo;
 import org.apache.flink.api.common.typeinfo.TypeHint;
 import org.apache.flink.api.common.typeinfo.TypeInformation;
 import org.apache.flink.api.common.typeinfo.Types;
 import org.apache.flink.api.java.tuple.Tuple2;
 import org.apache.flink.queryablestate.client.QueryableStateClient;

 import java.util.concurrent.CompletableFuture;

 /**
  * @author : 恋晨
  * Date   : 2019/10/9 4:05 PM
  * 功能   :    状态查询客户端示例
  */
 public class QueryClient {

     public static void main(String[] args) throws Exception{

         final JobID jobID = JobID.fromHexString("722bf03f719a2e81587cc1cc3c684
499");
         final String hostname = "localhost";
         final int port = 9069;

         QueryableStateClient client = new QueryableStateClient(hostname , port
);

         ValueStateDescriptor<Tuple2<String,Long>> valueStateDescriptor =
                 new ValueStateDescriptor<Tuple2<String, Long>>(
                         "QueryableState_WordCount_state",
                         Types.TUPLE(Types.STRING , Types.LONG)
                 );
         final String key = "hello";

         CompletableFuture<ValueState<Tuple2<String , Long>>> completableFuture
 =
                 client.getKvState(
                         jobID,
                         "QueryableState_WordCount",
                         key,
                         BasicTypeInfo.STRING_TYPE_INFO,
                         valueStateDescriptor
                 );
         System.out.println(completableFuture.get().value());
     }
```

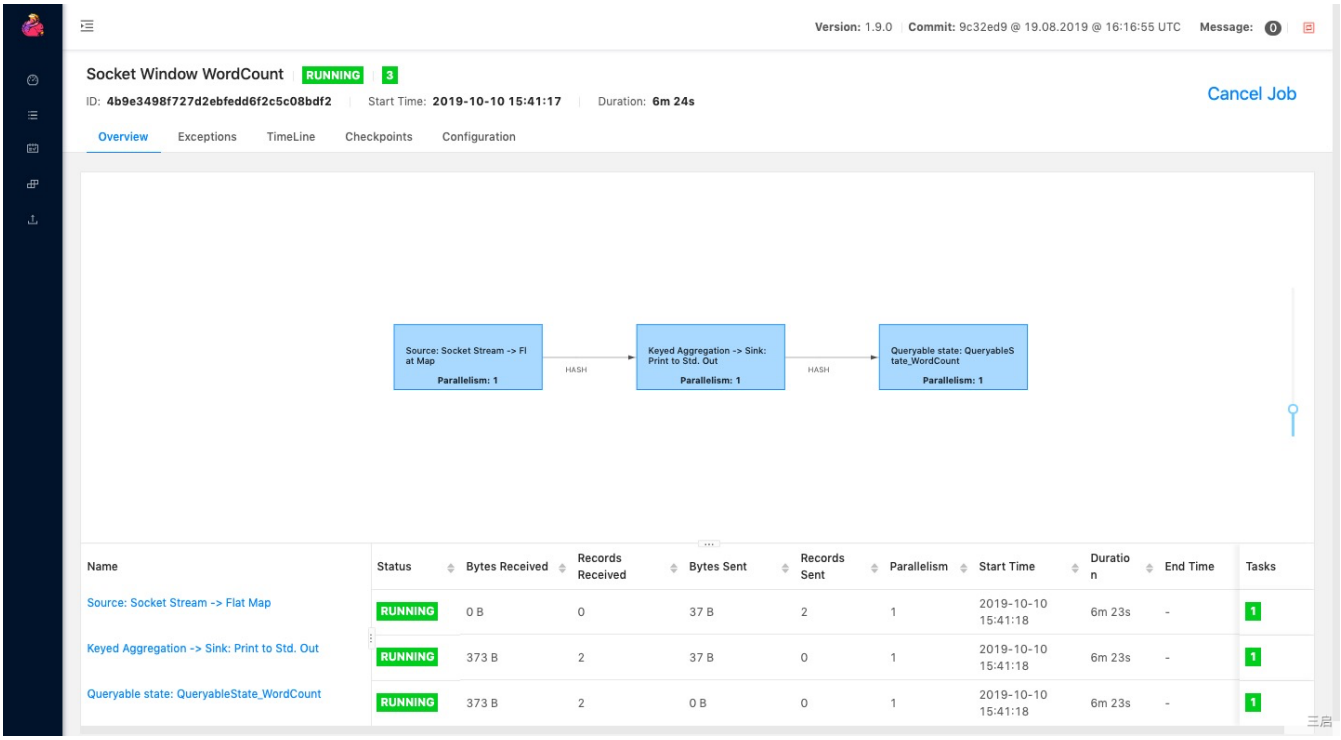
```
    }
```

测试环境test run

1. 将上述程序打包，并上传谷歌云；
2. 运行`nc -l 9000`,发送Words；
3. 提交job

```
bin/flink run QueryableState.jar  -C com.flink.state.queryable.WordCountOnSetSt
ream
```

web UI如下：



4. 使用telnet验证9069端口是否是通的；
5. 本地IDE启动queryable Client查询状态数据；