

实验六 Flink 部署与编程 (单机 + 伪分布式)

6.1 实验目的

- 学习 Flink 的部署，简单使用 Scala Shell
- 查看 Flink 的运行日志，体会与其它系统运行过程中查看日志方式的区别
- 通过系统部署理解体系架构，体会流计算系统与批处理系统之间的区别
- 学习编写简单的基于 DataStream API 的 Flink 程序
- 掌握在 IDEA 中调试 Flink 相关程序，以及在单机伪分布式部署方式下提交运行 Flink 程序的方法

6.2 实验任务

- 完成 Flink 的单机集中式部署以及单机伪分布式部署
- 在单机伪分布式部署方式下分别以 Attached 和 Detached 提交方式运行示例程序
- 完成 WordCount 示例程序的编写
- 在单机伪分布式环境下运行 WordCount 示例程序

6.3 实验环境

- 操作系统：Ubuntu 18.04 或 Ubuntu 20.04
- JDK 版本：1.8
- Flink 版本：1.12.1

6.4 实验步骤

注意：本教程以名为 dase-local 的用户为例开展实验，读者在实验过程中需替换为自己的用户名（云主机上的默认用户名为 ubuntu）。

6.4.1 单机集中式部署

(1) 准备工作

- 登录用户 dase-local
- 下载并安装 Flink（云主机镜像中已经下载好了，在/home/ubuntu 目录下，直接解压即可）

```
1 wget http://archive.apache.org/dist/flink/flink-1.12.1  
   /flink-1.12.1-bin-scala_2.11.tgz  
2 tar -zxvf flink-1.12.1-bin-scala_2.11.tgz
```

(2) 使用 Shell 运行 DataStream 程序

- 启动 Scala Shell

```
1 ~/flink-1.12.1/bin/start-scala-shell.sh local # 连接到本地Flink集群
```

- 在 *scala*> 后输入 Scala 代码

```
1 val textstreaming=senv.fromElements("a a b b c")  
2 val countsstreaming=textstreaming.flatMap { _.toLowerCase.split("\\W+") }  
   .map { (_, 1) }.keyBy(0).sum(1)  
3 countsstreaming.print()  
4 senv.execute() // 提交作业
```

运行结果如图6.1所示。

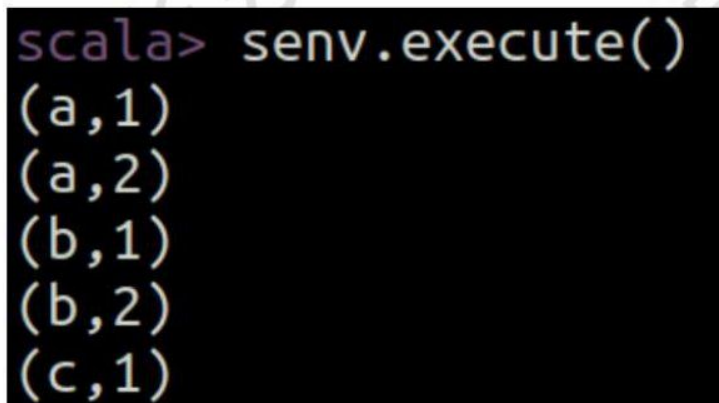


图 6.1 Shell 运行示例程序的结果

- 退出 Scala Shell, 在 *scala*> 后输入: *q*

6.4.2 单机伪分布式部署

(1) 准备工作

- 登录用户 *dase-local*

(2) 修改配置

配置文件位于 *~/flink-1.12.1/conf/* 目录下, 修改其中的 *flink-conf.yaml* 文件。

注意: 配置 *key/value* 时候在 “:” 后面需要有一个空格, 否则配置不会生效。
此外, 不要将注释与配置项写在同一行。

```
1 jobmanager.rpc.address: localhost
2 # 配置JobManager进行RPC通信的地址，使用默认值即可
3
4 jobmanager.rpc.port: 6123
5 # 配置JobManager进行RPC通信的端口，使用默认值即可
6
7 rest.port: 8081
8 # 客户端访问端口与可视化端口，使用默认值即可
9
10 taskmanager.numberOfTaskSlots: 2
11 # 配置TaskManager 提供的任务 slots 数量大小，默认为1
12
13 parallelism.default: 1
14 # 配置程序默认并行计算的个数，使用默认值即可
```

以下列举一些重要的配置值（需要调节时更改，本例中不做更改）：

```
1 jobmanager.heap.mb: 每个JobManager的可用内存量
2
3 taskmanager.heap.mb: 每个TaskManager的可用内存量
4
5 taskmanager.numberOfTaskSlots: 每台机器的可用CPU数量
6
7 parallelism.default: 集群中的CPU总数
8
9 taskmanager.tmp.dirs: 临时目录
10 #内存不够用时，写入到taskmanager.tmp.dirs指定的目录中。如果未显式指定参数，
    Flink会将临时数据写入操作系统的临时目录。
```

(3) 启动服务

- 启动命令

```
1 ~/flink-1.12.1/bin/start-cluster.sh
```

(4) 查看 Flink 服务信息

- 使用 jps 查看进程，验证是否成功启动服务

在单机伪分布式部署方式下，该节点既充当 JobManager 角色，又充当 TaskManager 角色，故该节点上会出现两个进程：一个 JobManager 进程和一个 TaskManager 进程。若同时出现 JobManager 进程和 TaskManager 进程，则

表明配置成功以及启动成功。注意，在 Standalone 模式下，JobManager 的进程名为 StandaloneSessionClusterEntrypoint，如图6.2所示。

```
dase-local@ecnu01:~$ jps
22534 StandaloneSessionClusterEntrypoint
22968 Jps
22808 TaskManagerRunner
dase-local@ecnu01:~$
```

图 6.2 启动 Flink 服务后存在的进程

- 查看 Flink 进程日志

本实验的进程日志记录在 `~/flink-1.12.1/log/` 路径下，后缀为 `.log` 的文件中。

- Client 进程日志: `flink-*-client-*.log`
- Scala Shell 进程日志: `flink-*-scala-shell-local-*.log`
- StandaloneSession 进程日志: `flink-*-standalonesession-*-*.log`
- TaskExecutor 进程日志: `flink-*-taskexecutor-*-*.log`

- 访问 Flink UI 界面

通过访问 `http://localhost:8081`(该端口号为 `flink-conf.yaml` 中的 `rest.port`)，Flink 集群信息，如图6.3所示。

因为之前在 `flink-conf.yaml` 中设置 `taskmanager.numberOfTaskSlots` 的值为 2，故每个 TaskManager 有 2 个 slot。



图 6.3 Flink UI 界面

(5) 运行 Flink DataStream 程序

- 使用 Shell 运行 DataStream 程序
 - 启动 Scala Shell

```
1 ~/flink-1.12.1/bin/start-scala-shell.sh remote localhost 8081 #
   连接本地单机伪分布式集群
```

- 在 `scala>` 后输入 Scala 代码

```

1 val textstreaming=senv.fromElements("a a b b c")
2 val countsstreaming=textstreaming.flatMap { _.toLowerCase.split("\\W+") }
   .map { (_, 1) }.keyBy(0).sum(1)
3 countsstreaming.print()
4 senv.execute() // 提交作业

```

另起一个终端，输入如下命令：

```
1 tail -f ~/flink-1.12.1/log/flink-dase-local-taskexecutor-0-ecnu01.out
```

运行结果如图6.4所示。

```

(a,1)
(a,2)
(b,1)
(b,2)
(c,1)

```

图 6.4 Shell 运行示例程序的结果

- 退出 Scala Shell，在 `scala>` 后输入：`q`
- 通过提交 jar 包运行 DataStream 程序
 - Attached 方式（默认）

* 启动 Netcat 服务

另起一个终端（记此终端为终端 2.1），输入如下命令：

```

1 nc -lk 8888
2 #监听8888端口，之后我们会在此输入数据

```

* 提交 jar 包

另起一个终端（记此终端为终端 2.2），并在该终端中输入相应提交命令来提交 jar 包。

```

1 ~/flink-1.12.1/bin/flink run
   ~/flink-1.12.1/examples/streaming/SocketWindowWordCount.jar
   --port 8888

```

在终端 2.1 中输入如图6.5所示的数据。另起一个终端，输入如下命令查看运行结果，如图6.6所示，可以看到这次的程序运行结果也追加到了 `flink-dase-local-taskexecutor-0-ecnu01.out` 文件中。值得注意的是，图6.6所显示的运行结果会随着图6.5中输入速度的不同而有所差异；若实验过程

中 Flink 集群出现过故障，则可能会产生如 `flink-dase-local-taskexecutor-1-ecnu01.out`、`flink-dase-local-taskexecutor-1-ecnu01.out` 之类的文件，同时运行结果也会输出到最新的文件中，因此需要将命令中的 `flink-dase-local-taskexecutor-0-ecnu01.out` 文件替换为对应的最新的文件即可。

```
tail -f ~/flink-1.12.1/log/flink-dase-local-taskexecutor-0-ecnu01.out
```

```
dase-local@ecnu01:~$ nc -lk 8888
hello dase
hello ecnu
```

图 6.5 输入数据

```
(a,1)
(a,2)
(b,1)
(b,2)
(c,1)
hello : 2
ecnu : 1
dase : 1
```

图 6.6 Attached 方式提交示例程序的部分运行结果

```
dase-local@ecnu01:~$ jps
11762 StandaloneSessionClusterEntrypoint
3155 CliFrontend
4762 Jps
12045 TaskManagerRunner
dase-local@ecnu01:~$
```

图 6.7 程序运行中存在的进程

另起一个终端，输入 `jps` 查看进程，如图 6.7 所示。我们可以看到一个 `CliFrontend` 进程。

* 停止 `DataStream` 程序

· 方法 1：终止数据源

在启动 Netcat 服务的窗口使用 `Ctrl + C`，停止 Netcat 服务。应用程序状态会变为 `Finished`。

- 方法 2：在 Flink UI 中停止应用程序

访问 <http://localhost:8081>，选中 Running Jobs 标题下对应的应用程序进入详情页面，点击右上角的 Cancel 停止 DataStream 程序。应用程序状态会变为 Canceled。

- 方法 3：命令行停止应用程序

另起一个终端，使用 list 命令列出正在运行的 Job（其中的 JobID 为一串哈希值），随后根据 JobID 使用 cancel 命令终止应用程序。终止后，应用程序状态会变为 Canceled。

```
1 ~/flink-1.12.1/bin/flink list
2 ~/flink-1.12.1/bin/flink cancel JobID #用查询到的哈希值替换JobID
```

此外，若未结束 Netcat 服务，则在终端 2.1 中使用 Ctrl + C 结束 Netcat 服务。

– Detached 方式

- * 启动 Netcat 服务

在终端为 2.1 中输入如下命令：

```
1 nc -lk 8888
2 #监听8888端口，之后我们会在此输入数据
```

- * 提交 jar 包

在终端为 2.2 中输入如下命令：

```
1 ~/flink-1.12.1/bin/flink run -d
   ~/flink-1.12.1/examples/streaming/SocketWindowWordCount.jar
   --port 8888
```

在终端 2.1 中输入数据，如图 6.8 所示。

在终端 2.2 中输入如下命令查看运行结果。如图 6.9 所示，可以看到这次的程序运行结果也追加到了 `flink-dase-local-taskexecutor-0-ecnu01.out` 文件中。值得注意的是，图 6.9 所显示的运行结果会随着图 6.8 中输入速度的不同而有所差异。

```
1 tail -f ~/flink-1.12.1/log/flink-dase-local-taskexecutor-0-ecnu01.out
   # tail -f 默认显示文件最后10行内容
```

另起一个终端，输入 `jps` 查看进程。此时，不会出现 `CliFrontend` 进程。

(6) 查看 Flink 程序运行信息

- 实时查看应用运行情况

```
dase-local@ecnu01:~$ nc -lk 8888
hello dase
hello ecnu
```

图 6.8 输入数据

```
(b,2)
(c,1)
hello : 1
dase : 1
hello : 1
ecnu : 1
hello : 1
dase : 1
hello : 1
ecnu : 1
```

图 6.9 detached 方式提交示例程序的部分运行结果

在应用运行过程中，访问 <http://localhost:8081>，选中 Running Jobs 标题下对应的应用程序进入详情页面。如图6.10所示，可以从图中看到应用程序的 DAG 图和 Subtasks 信息，还可切换到 Timeline、Exceptions 或 Configuration 等标签页查看相关信息。



图 6.10 Flink 应用程序界面

- 查看 Flink 应用程序日志

在提交一个应用程序后，在 `~flink-1.12.1/log` 下会出现应用程序运行日志（注意日志名称格式为 `flink-*-taskexecutor-*-*.log`，`.out` 结尾的文件为应用程序输出结果）。

- 查看应用历史记录



Job Name	Start Time	Duration	End Time	Tasks	Status
Socket Window WordCount	2019-01-12 12:40:41	1min 36s	2019-01-12 12:42:16	1	Completed
Socket Window WordCount	2019-01-12 12:42:16	28.29m 23s	2019-01-12 12:42:43	1	Completed

图 6.11 Flink 已完成应用界面

在应用运行结束后，访问 <http://localhost:8081>，点击左侧 Dashboard 的中的 Completed Jobs 可查看应用提交历史记录。Web 界面如图6.11所示。

(7) 停止服务

- 停止 DataStream 程序

如有正在运行的 DataStream 程序，参照前述方法停止 DataStream 程序。

- 停止 Flink

```
1 ~/flink-1.12.1/bin/stop-cluster.sh
```

6.4.3 编写 Flink 应用程序

(1) 新建 Maven 项目并添加依赖

- 新建名为“FlinkDemo”的 Maven 项目
- 编辑项目根目录下的 pom.xml 文件，分别在 <build> 和 <dependencies> 标签中添加 Flink 相关的依赖包 `flink-streaming-java_2.11` 和 `flink-core`

```
1 <dependencies>
2 <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-streaming-java -->
3 <dependency>
4   <groupId>org.apache.flink</groupId>
5   <artifactId>flink-streaming-java_2.11</artifactId>
6   <version>1.12.1</version>
7   <scope>compile</scope>
8 </dependency>
9
10 <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-core -->
11 <dependency>
12   <groupId>org.apache.flink</groupId>
13   <artifactId>flink-core</artifactId>
14   <version>1.12.1</version>
15 </dependency>
16 <dependency>
17   <groupId>org.apache.flink</groupId>
18   <artifactId>flink-clients_2.11</artifactId>
```

```

19     <version>1.12.1</version>
20 </dependency>
21 <dependency>
22     <groupId>org.slf4j</groupId>
23     <artifactId>slf4j-simple</artifactId>
24     <version>1.7.25</version>
25 </dependency>
26 </dependencies>

```

修改完成后，在菜单界面选择 View->Tool Windows->Maven，在弹出的界面中点击 Reload All Maven Projects 加载依赖文件，第一次加载此过程可能耗时较长。

(2) 编写程序

- Java 版: 新建 Java 类 *WordCount*

在项目的 *src/main/java* 目录下，选择 New -> Package，输入名称 *cn.edu.ecnu.flink.example.java.wordcount*，之后右键单击新建好的包，选择 New -> Java Class，输入名称 *WordCount*。

```

1 package cn.edu.ecnu.flink.example.java.wordcount;
2
3 import org.apache.flink.api.common.functions.FlatMapFunction;
4 import org.apache.flink.api.common.functions.MapFunction;
5 import org.apache.flink.api.java.tuple.Tuple2;
6 import org.apache.flink.streaming.api.datastream.DataStream;
7 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
8 import org.apache.flink.util.Collector;
9
10 public class WordCount {
11     public static void main(String[] args) throws Exception {
12         run(args);
13     }
14
15     public static void run(String[] args) throws Exception {
16         /* 步骤1: 创建StreamExecutionEnvironment对象 */
17         StreamExecutionEnvironment env =
18             StreamExecutionEnvironment.getExecutionEnvironment();
19
20         /* 步骤2: 按应用逻辑使用操作算子编写DAG，操作算子包括数据源、转换、数据池
21            等 */
22         // 从指定的主机名和端口号接收数据，创建名为lines的DataStream
23         DataStream<String> lines = env.socketTextStream(args[0],
24             Integer.parseInt(args[1]), "\n");

```

```

22 // 将lines中的每一个文本行按空格分割成单个单词
23 DataStream<String> words =
24     lines.flatMap(
25         new FlatMapFunction<String, String>() {
26             @Override
27             public void flatMap(String value, Collector<String>
28                 out) throws Exception {
29                 for (String word : value.split(" ")) {
30                     out.collect(word);
31                 }
32             }
33         });
34 // 将每个单词的频数设置为1, 即将每个单词映射为[单词, 1]
35 DataStream<Tuple2<String, Integer>> pairs =
36     words.map(
37         new MapFunction<String, Tuple2<String, Integer>>() {
38             @Override
39             public Tuple2<String, Integer> map(String value)
40                 throws Exception {
41                 return new Tuple2<String, Integer>(value, 1);
42             }
43         });
44 // 按单词聚合, 并对相同单词的频数使用sum进行累计
45 DataStream<Tuple2<String, Integer>> counts = pairs.keyBy(0).sum(1);
46 // 输出词频统计结果
47 counts.print();
48
49 /* 步骤3: 触发程序执行 */
50 env.execute("Streaming WordCount");

```

(3) Scala 版: 新建 Scala 类 *WordCountScala* (选做)

在项目的 *src/main* 目录下, 选择 New -> Directory, 输入 *scala*, 接着右键单击目录 *scala*, 选择 Mark Directory as -> Sources Root. 在 *scala* 目录下选择 New->Package, 输入名称 *cn.edu.ecnu.flink.example.scala.wordcount*, 之后右键单击新建好的包, 选择 New -> Scala Class, 输入名称 *WordCount*.

```

1 package cn.edu.ecnu.flink.example.scala.wordcount
2
3 import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
4 import org.apache.flink.streaming.api.scala._

```



```

5 import org.apache.flink.streaming.api.windowing.assigners.{GlobalWindows,
    TumblingEventTimeWindows, TumblingProcessingTimeWindows}
6 import org.apache.flink.streaming.api.windowing.time.Time
7
8 object WordCount {
9     def run(args: Array[String]): Unit = {
10         /* 步骤1: 创建StreamExecutionEnvironment对象 */
11         val env = StreamExecutionEnvironment.getExecutionEnvironment
12
13         /* 步骤2: 按应用逻辑使用操作算子编写DAG, 操作算子包括数据源、转换、数据池等 */
14         // 从指定的主机名和端口号接收数据, 创建名为lines的DataStream
15         val lines = env.socketTextStream(args(0), args(1).toInt)
16         // 将lines中的每一个文本行按空格分割成单个单词
17         val words = lines.flatMap(w => w.split(" "))
18         // 将每个单词的频数设置为1, 即将每个单词映射为[单词, 1]
19         val pairs = words.map(word => (word, 1))
20         // 按单词聚合, 并对相同单词的频数使用sum进行累计
21         val counts = pairs.keyBy(0)
22             .sum(1)
23         // 输出词频统计结果
24         counts.print()
25
26         /* 步骤3: 触发程序执行 */
27         env.execute("Streaming WordCount")
28     }
29
30     def main(args: Array[String]): Unit = {
31         run(args)
32     }
33 }

```

6.4.4 调试 Flink 应用程序

(1) 启动 Netcat 服务

```
1 nc -lk 8888 #在 8888 端口开启Ubuntu自带的Netcat服务, 之后我们会在此输入数据
```

(2) 配置运行环境

在菜单栏点击 Run -> Edit Configuration, 在弹出的界面中点击 + 号选择 Application, 新建 Application 配置, Name 为 WordCountFlink (Scala 版则为 WordCountScala), 配置界面如图6.12所示。

- 配置 Main Class 为 `cn.edu.ecnu.flink.example.java.wordcount.WordCount` (Scala 版则配置为 `cn.edu.ecnu.flink.example.scala.wordcount.WordCount`)
- 配置 Program arguments 为 IP 或者主机名 端口号
如 `localhost 8888`

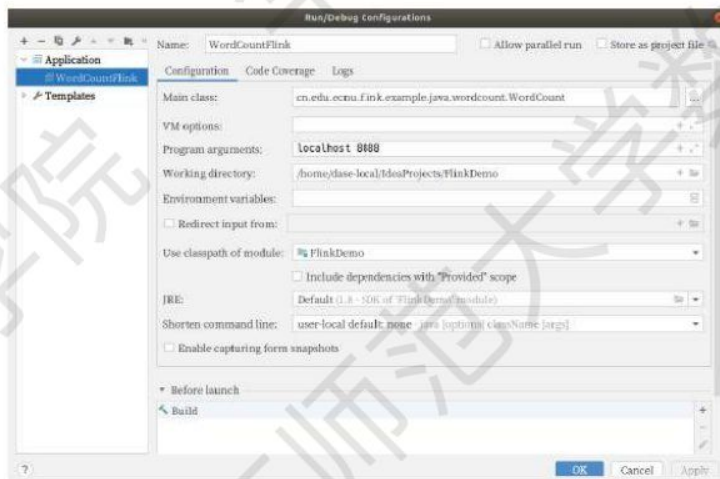


图 6.12 运行应用程序配置界面

(3) 调试程序

- 在菜单栏点击 Run->Run 'WordCountFlink'(Scala 版为 Run 'WordCountScala')
- 在启动 Netcat 服务的窗口中输入数据，如图6.13所示

```
dase-local@ecnu01:~$ nc -lk 8888
hello dase
hello ecnu
hello flink
```

图 6.13 输入数据

- 在 IDEA 中查看处理结果，如图6.14所示

```
4> (dase,1)
2> (hello,1)
1> (ecnu,1)
2> (hello,2)
2> (hello,3)
4> (flink,1)
```

图 6.14 在 IDEA 中的运行结果


```
1 ~/flink-1.12.1/bin/flink run -c  
  cn.edu.ecnu.flink.example.java.wordcount.WordCount  
  ~/flink-1.12.1/myApp/FlinkWordCount.jar localhost 8888
```

- 在启动 Netcat 服务的终端中输入数据，如图6.16所示。

```
dase-local@ecnu01:~$ nc -lk 8888  
hello flink  
hello dase  
hello ecnu
```

图 6.16 为程序提供输入数据

- 另起一个终端查看输出结果，如图6.17所示

```
1 tail -f ~/flink-1.12.1/log/flink-dase-local-taskexecutor-0-ecnu01.out
```

```
(hello,1)  
(flink,1)  
(hello,2)  
(dase,1)  
(hello,3)  
(ecnu,1)
```

图 6.17 程序的输出结果

- 停止应用程序和服务

- 终止应用程序

在启动 Netcat 服务的窗口使用 Ctrl + C，停止 Netcat 服务。应用程序状态会变为 Finished。

- 停止 Flink 服务

```
1 ~/flink-1.12.1/bin/stop-cluster.sh
```

6.5 思考题

- 1 在部署 Flink 时，具体是通过 flink.yaml 配置文件中的什么参数来指定每个 TaskManager 中的 slots 数量的？

2 一个 Flink 流计算应用程序何时停止、如何停止？请通过与 Spark 批处理应用程序进行对比来阐述。

3 如何查看一个 Flink 流计算应用程序启动了多少个任务，以及每个任务的并行度是多少？请结合 Web UI 来说明。

4 一个 Flink 应用程序中的算子会发生合并吗？哪些算子会合并？并结合上述编程示例进行说明。