Flink快速上手之Java API使用

本文将介绍如何通过简单地几步来开始编写你的 Flink Java 程序。

要求

编写你的Flink Java程序唯一的要求是需要安装Maven 3.0.4(或者更高)和Java 7.x(或者更高)

创建Flink Java工程

使用下面其中一个命令来创建Flink Java工程

1、使用Maven archetypes：

```
$ mvn archetype:generate                               ₩
    -DarchetypeGroupId=org.apache.flink            ₩
    -DarchetypeArtifactId=flink-quickstart-java    ₩
    -DarchetypeVersion=1.0.0
```

2、运行quickstart脚本

```
$ curl https://flink.apache.org/q/quickstart.sh | bash
```

上面两个命令（使用Maven创建的时候需要输入artifactId、groupId等相关信息）运行完之后都会产生类似于以下的目录结构：

```
.
└── quickstart
    ├── pom.xml
    └── src
        └── main
            ├── java
            │   └── org
            │       └── myorg
            │           └── quickstart
            │               ├── Job.java
            │               ├── SocketTextStreamWordCount.java
```

```
│           └── WordCount.java
└── resources
    └── log4j.properties

8 directories, 5 files
```

检查工程

您的工作目录中会出现一个新的目录。如果你使用了curl命令来创建Flink Java工程，这个目录的名称是 quickstart。否则，就是你输入的artifactId名字。

这个工程是一个Maven工程, 包含三个类。 Job是一个基本的框架程序，SocketTextStreamWordCount和WordCount是一个简单的单词次数计算示例。需要注意的是，这三个类的main方法都允许你在开发/测试模式下启动Flink。

我们推荐将这个工程导入到你的IDE中，并进行开发和测试。 如果你用的是Eclipse，可以使用m2e插件来导入 Maven 工程。有些Eclipse发行版默认嵌入了这个插件，其他的需要你手动去安装。IntelliJ IDE内置就提供了对Maven工程的支持。

给Mac OS X用户的建议：默认的JVM堆内存对Flink来说太小了，你必须手动增加内存。这里以Eclipse为例，依次选择 Run Configurations -> Arguments， 然后在VM Arguments里写入：-Xmx800m。

编译工程

如果你想要编译你的工程, 进入到工程所在目录，并输入 mvn clean install -Pbuild-jar命令。 你将会找到target/your-artifact-id-1.0-SNAPSHOT.jar文件，它可以在任意的Flink集群上运行。 还有一个fat-jar，名为target/your-artifact-id-1.0-SNAPSHOT-flink-fat-jar.jar，包含了所有添加到 Maven 工程的依赖。

下一步

开始编写我们自己的程序

Quickstart 工程包含了一个 WordCount 的实现，也就是大数据处理系统的 Hello World。WordCount 的目标是计算文本中单词出现的频率。比如： 单词 “the” 或者 “house” 在所有的Wikipedia文本中出现了多少次。

样本输入

big data is big

样本输出

big 2
data 1
is 1

下面的代码就是 Quickstart 工程的 WordCount 实现，它使用两种操作(FlatMap 和 Reduce)处理了一些文本，并且在标准输出中打印了单词的计数结果。

```
public class WordCount {

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

    // set up the execution environment
    final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

    // get input data
    DataSet<String> text = env.fromElements(
        "To be, or not to be,--that is the question:--",
        "Whether 'tis nobler in the mind to suffer",
        "The slings and arrows of outrageous fortune",
        "Or to take arms against a sea of troubles,"
        );

    DataSet<Tuple2<String, Integer>> counts =
        // split up the lines in pairs (2-tuples) containing: (word,1)
        text.flatMap(new LineSplitter())
        // group by the tuple field "0" and sum up tuple field "1"
        .groupBy(0)
        .aggregate(Aggregations.SUM, 1);

    // emit result
    counts.print();
  }
}
```

这些操作是在专门的类中定义的，下面是 LineSplitter 类。

```
public class LineSplitter implements FlatMapFunction<String, Tuple2<String, Integer>> {

  @Override
  public void flatMap(String value, Collector<Tuple2<String, Integer>> out) {
    // normalize and split the line into words
    String[] tokens = value.toLowerCase().split("₩₩W+");

    // emit the pairs
    for (String token : tokens) {
      if (token.length() > 0) {
        out.collect(new Tuple2<String, Integer>(token, 1));
      }
    }
  }
}
```

完整代码参见https://github.com/apache/flink/blob/master//flink-examples/flink-examples-batch/src/main/java/org/apache/flink/examples/java/wordcount/WordCount.java

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接:【】（）