Scio:Apache Beam和Google Cloud Dataflow的Scala API

我们都知道，目前 Apache Beam 仅仅提供了 Java 和 Python 两种语言的 API，尚不支持 Scala 相关的 API。基于此全球最大的流音乐服务商 Spotify 开发了 Scio ，其为 Apache Beam 和 Google Cloud Dataflow 提供了Scala API，使得我们可以直接使用 Scala 来编写 Beam 应用程序。Scio 开发受 Apache Spark 和 Scalding 的启发，目前最新版本是 Scio 0.3.0，0.3.0版本之前依赖于 Google Cloud Dataflow SDK，0.3.0及未来版本会直接依赖于 Apache Beam。Scio 目前使用 Apache License, Version 2.0 许可证发布，源代码在 https://github.com/spotify/scio。

主要功能

- Scala API 与 Spark 和 Scalding 的核心 API 非常类似
- 统一 batch 和 streaming 编程模型
- 与 Google Cloud 产品集成，包括：云存储，BigQuery，Pub/Sub，Datastore，Bigtable
- 支持 HDFS、JDBC、TensorFlow TFRecords、Cassandra 以及 Elasticsearch I/O
- 使用 Scio REPL 支持交互模式
- 可以与Algebird和Breeze整合
- 分布式缓存
- Pipeline orchestration with Scala Futures

使用

前面说了 Scio 开发受 Apache Spark 和 Scalding 的启发，所以如果我们使用 Scio API 来编写一个 WordCount 程序看起来和使用 Spark 来编写很类似。首先我们需要引入相关依赖：

```
libraryDependencies ++= Seq(
  "com.spotify" % "scio-core_2.11" % "0.3.4",
  "com.spotify" % "scio-test_2.11" % "0.3.4" % "test"
)
```

或

```
libraryDependencies ++= Seq(
  "com.spotify" %% "scio-core" % "0.3.4",
  "com.spotify" %% "scio-test" % "0.3.4" % "test"
)
```

或

```
<dependency>
   <groupId>com.spotify</groupId>
   <artifactId>scio-core_2.11</artifactId>
   <version>0.3.4</version>
</dependency>

<dependency>
   <groupId>com.spotify</groupId>
   <artifactId>scio-test_2.11</artifactId>
   <version>0.3.4</version>
   <scope>test</scope>
</dependency>
```

然后我们的 Scio API 版的 WordCount 可以这样来编写：

```
package com.iteblog

import com.spotify.scio._
import com.spotify.scio.accumulators._
import com.spotify.scio.examples.common.ExampleData

object WordCount {
  def main(cmdlineArgs: Array[String]): Unit = {
    val (sc, args) = ContextAndArgs(cmdlineArgs)

    val input = args.getOrElse("input", ExampleData.KING_LEAR)
    val output = args("output")

    // initialize accumulators
    val max = sc.maxAccumulator[Int]("maxLineLength")
    val min = sc.minAccumulator[Int]("minLineLength")
    val sumNonEmpty = sc.sumAccumulator[Long]("nonEmptyLines")
    val sumEmpty = sc.sumAccumulator[Long]("emptyLines")

    sc.textFile(input)
      .map(_.trim)
      .accumulateBy(max, min)(_.length)
      .accumulateCountFilter(sumNonEmpty, sumEmpty)(_.nonEmpty)
      .flatMap(_.split("[^a-zA-Z']+").filter(_.nonEmpty))
      .countByValue
      .map(t => t._1 + ": " + t._2)
```

```
      .saveAsTextFile(output)

    val result = sc.close().waitUntilFinish()

    // scalastyle:off regex
    // retrieve accumulator values
    println("Max: " + result.accumulatorTotalValue(max))
    println("Min: " + result.accumulatorTotalValue(min))
    println("Sum non-empty: " + result.accumulatorTotalValue(sumNonEmpty))
    println("Sum empty: " + result.accumulatorTotalValue(sumEmpty))
    // scalastyle:on regex
  }
}
```

编写玩之后，我们可以

```
iteblog@www.iteblog.com scio $ sbt
[info] ...
> project scio-examples
[info] ...
> runMain com.iteblog.WordCount --input=<FILE PATTERN> --output=<DIRECTORY>
--project=[PROJECT] --runner=DataflowRunner --zone=[ZONE]
```

注意：和我们之前见到的不一样，--input 参数匹配的文件必须写到文件那层，也就是需要使用gs//bucket/path/part-*.txt而不是gs://bucket/path；
如果 --runner 没指定，默认的是 DirectRunner 。

更多关于 Scio 的使用，请参见官方文档 https://github.com/spotify/scio/wiki。