**Spark Others**

|  |  |
| --- | --- |
| **审核人** |  |
| **重要性** | 中 |
| **紧迫性** | 中 |
| **拟制人** | 张包峰 |
| **提交日期** |  |

**作者：张包峰**

(版权所有,翻版必究)

**修改记录**

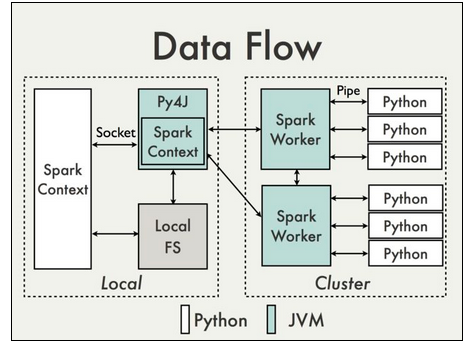
|  |  |  |
| --- | --- | --- |
| **更新时间** | **变更内容** | **变更理由** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

[**作者：张包峰** 1](#_Toc392841527)

[1.1 Pyspark机制 4](#_Toc392841528)

## Pyspark机制

<https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals>



*pyspark data flow*

在python driver端，SparkContext利用Py4J启动一个JVM并产生一个JavaSparkContext。Py4J只使用在driver端，用于本地python与java SparkContext objects的通信。大量数据的传输使用的是另一个机制。

RDD在python下的转换会被映射成java环境下PythonRDD。在远端worker机器上，PythonRDD对象启动一些子进程并通过pipes与这些子进程通信，以此send用户代码和数据。

### 大致流程和做法

java\_gateway.py里启动了py4j.JavaGateWay，并从java里导入了类，



python能通过py4j访问jvm的前提是，jvm开启了GatewayServer，在core工程的deploy工程下，PythonRunner单例里启动了GatewayServer。

py4j的包为$SPARK\_HOME/python/lib/py4j-0.8.1-src.zip

java\_gateway的launch\_gateway()方法在context.py初始化的时候调用。

context.py初始化的时候，完成了SparkContext和其部分主要方法，加入到python环境中，所以大多数的调用都是通过py4j直接调用java的类。java的类主要是指core项目里的java api里的内容。序列化采用的是cPickle库的PickleSerializer等类。

像python下使用spark sql的话，在sql.py里，从jvm里获取了SQLContext/HiveContext类，从而得到spark sql里的关键方法。

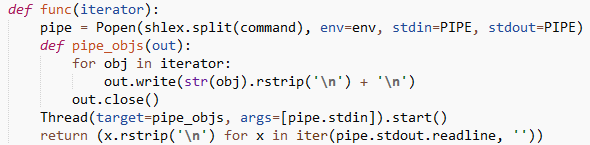
另一方面，worker.py里，worker启动的时候会起一个socket，从socket里，可以获取工作目录名字；可以获取PYTHONPATH下的其他要引入的.zip或.egg文件，将其加到file\_dir里，这里的反序列化使用的是UTF8Deserializer；可以获取广播的变量，这里的反序列化使用的是PickleSerializer。

在daemon.py里，通过分配socket端口，启动POOLSIZE个worker，(里面其他很多细节没有看明白)，使用os.fork的方式创建子进程来启动。

在rdd.py里，声明了rdd的很多action和transformations，有些操作会触发数据在python worker上的传输。

传输大量数据的时候，Py4J很慢，因为socket.readline()很低效。传输的时候，把数据(序列化后)dump成一个文件。后续把这个文件反序列化回来后，可以转成python的类型和结构进行查看和输出(如collect)，也可以调用PythonRDD的asJavaRDD方法(如PipelinedRDD计算时)，在各个worker上启动python进程执行反序列化之后的函数，通过管道与python进程进行通信，最后得到JavaRDD。

管道传输利用的是Popen，这样做标准输入



## StreamSQL

<https://github.com/thunderain-project/StreamSQL>