Spark入门

Introduction to Spark

Spark下载

Lightning-fast unified analytics engine

Download | Libraries | Documentation | Examples | Community | Developers

Download Apache Spark™

1. Choose a Spark release: 2.4.3 (May 07 2019)
2. Choose a package type: Pre-built for Apache Hadoop 2.7 and later
3. Download Spark: spark-2.4.3-bin-hadoop2.7.tgz
4. Verify this release using the 2.4.3 signatures, checksums and project release KEYS.

Note that, Spark is pre-built with Scala 2.11 except version 2.4.2, which is pre-built with Scala 2.12.

Link with Spark

Spark artifacts are hosted in Maven Central. You can add a Maven dependency with the following coordinates:

```
groupId: org.apache.spark
artifactId: spark-core_2.11
version: 2.4.3
```

Spark下载

```
(base) cn40883@CL10082-M spark-2.4.2-bin-hadoop2.7 $ ll
total 160
drwxr-xr-x@  19 cn40883  staff    608 Apr 26 11:36 .
drwxr-xr-x   44 cn40883  staff   1408 May 24 17:30 ..
-rw-r--r--@   1 cn40883  staff  21357 Apr 19 07:31 LICENSE
-rw-r--r--@   1 cn40883  staff  42919 Apr 19 07:31 NOTICE
drwxr-xr-x@   3 cn40883  staff     96 Apr 19 07:31 R
-rw-r--r--@   1 cn40883  staff   3952 Apr 19 07:31 README.md
-rw-r--r--@   1 cn40883  staff    165 Apr 19 07:31 RELEASE
-rw-r--r--    1 cn40883  staff   3026 Apr 26 11:35 als.py
drwxr-xr-x@  29 cn40883  staff    928 Apr 19 07:31 bin
drwxr-xr-x@   9 cn40883  staff    288 Apr 19 07:31 conf
drwxr-xr-x@   5 cn40883  staff    160 Apr 19 07:31 data
drwxr-xr-x@   4 cn40883  staff    128 Apr 19 07:31 examples
drwxr-xr-x@ 229 cn40883  staff   7328 Apr 19 07:31 jars
drwxr-xr-x@   4 cn40883  staff    128 Apr 19 07:31 kubernetes
drwxr-xr-x@  49 cn40883  staff   1568 Apr 19 07:31 licenses
drwxr-xr-x    4 cn40883  staff    128 Apr 26 10:56 py_spark
drwxr-xr-x@  21 cn40883  staff    672 Apr 26 11:18 python
drwxr-xr-x@  24 cn40883  staff    768 Apr 19 07:31 sbin
drwxr-xr-x@   3 cn40883  staff     96 Apr 19 07:31 yarn
```

- README.md 使用说明
- bin 包含了spark交互式运行的一系列软件
- 一系列spark的核心代码

Spark的Python和Scala Shell

- Spark的shell可用来与分布式存储在许多机器的内存中或者磁盘上的数据进行交互，并且处理过程的分发由Spark自动完成

```
(base) cn40883@CL10082-M spark-2.4.2-bin-hadoop2.7 $ ./bin/pyspark
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
19/06/01 10:42:54 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wher
 applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.2
      /_/

Using Python version 3.7.3 (default, Mar 27 2019 16:54:48)
SparkSession available as 'spark'.
>>>
```

Spark的Python和Scala Shell

- 我们通过对弹性式分布式数据集（RDD）的操作来表达我们的计算意图，这些计算会自动的在集群中运行。

- 这样的数据集被称为弹性分布式数据集（resilient distributed database）简称RDD。

- RDD是对分布式数据和计算的基本抽象

- 在详细介绍RDD之前先看几个例子

Spark的Python Shell

```
>>> # 统计README.md行数
...
>>> lines = sc.textFile('./README.md') # 创建一个叫做lines的RDD
>>> lines.count() # 获得RDD的行数
105
>>> lines.first() # 获得RDD的第一个元素，也就是首行
'# Apache Spark'
```

Spark的核心概念

- RDD：是弹性分布式数据集（Resilient Distributed Dataset）的简称，是分布式内存的一个抽象概念，提供了一种高度的共享内存模型；
- DAG：是Directed Acyclic Graph（有向无环图）的简称，反映RDD之间的依赖关系；
- Executor：是运行在工作节点（Worker Node）上的一个进程，负责运行任务，并为应用程序存储数据；
- 应用：用户编写的Spark应用程序；
- 任务：运行在Executor上的工作单元；
- 作业：一个作业包含多个RDD及作用于相应RDD上的各种操作；
- 阶段：是作业的基本调度单位，一个作业会分为多组任务，每组任务被称为“阶段”，或者也被称为“任务集”。

Spark的运行架构

- 每一个spark应用都由一个驱动器程序（driver）发起集群上的各种并行操作。驱动器中包含main函数，并且定义了在集群上的分布式数据集。实际上driver就是spark shell本身。

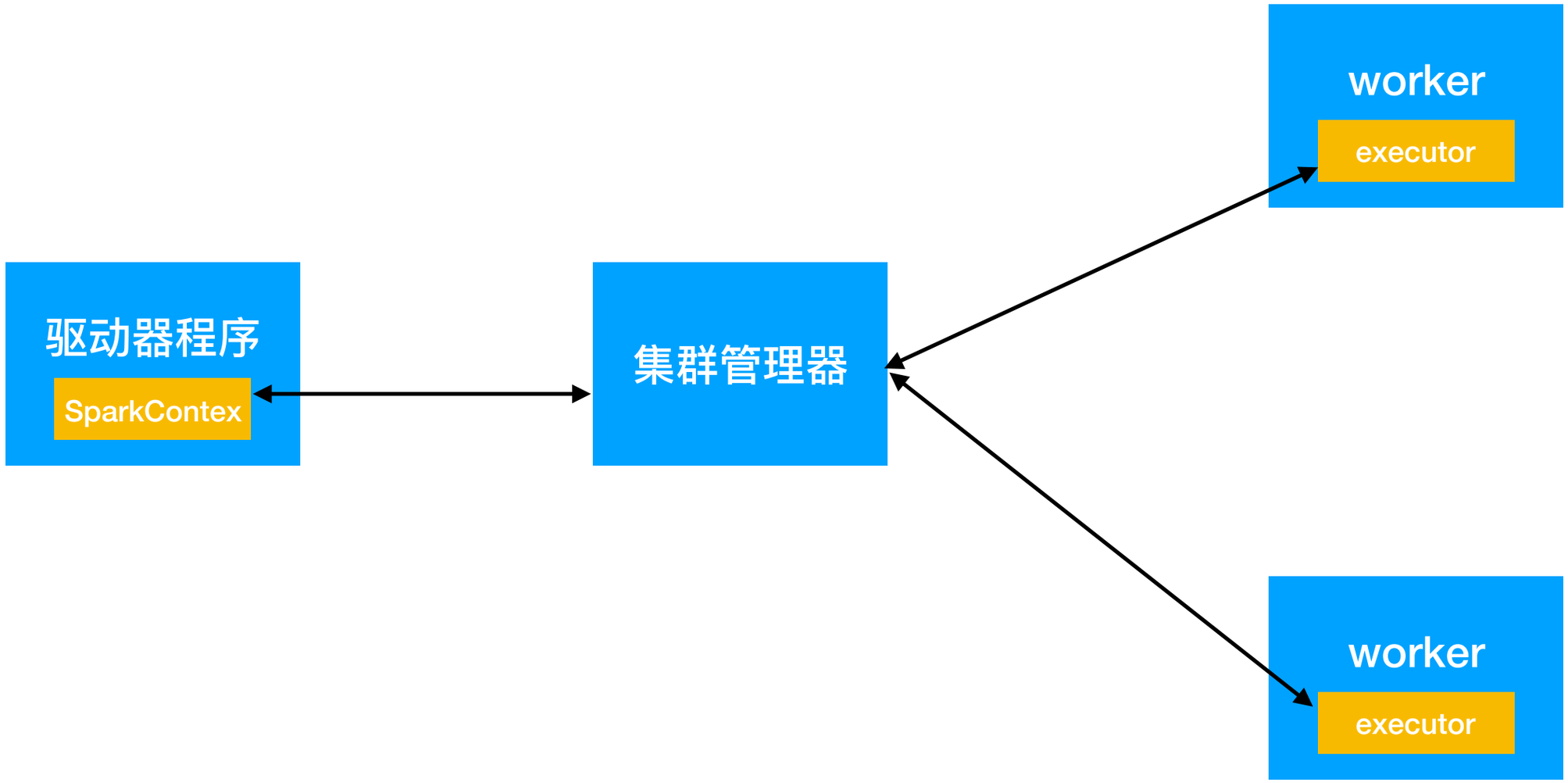
    - 驱动器通过一个SparkContext对象访问Spark。这个对象代表一个对集群的连接。shell启动的时候就会自己创建一个SparkContext，叫做sc，Spark 2.0之后SparkContext封装到SparkSession对象中。

    - 通过SparkContext我们就可以创建RDD。例如sc.textFile()，并且对RDD进行了操作，例如count()

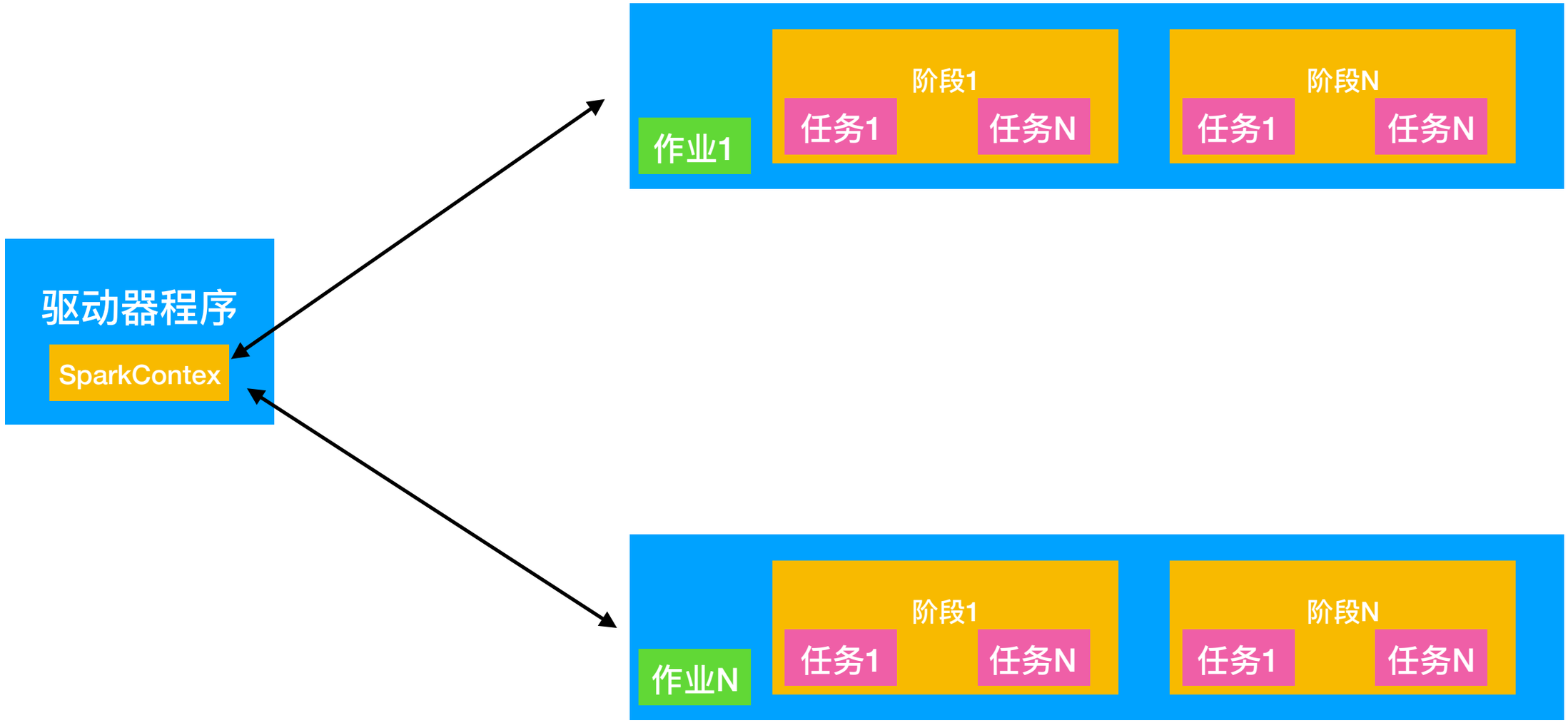
- 需要执行这些操作，驱动器一般要管理多个执行器(executor)，比如统计行数的时候，不同的节点会统计自己部分的行数。

- Spark有自己的集群资源管理器也可以使用Yarn或者Mesos。

Spark的运行架构



Spark的运行架构



Spark中各个概念的关系

Spark的运行流程

1. 当一个Spark应用被提交时，由Driver创建一个SparkContext，由SparkContext负责和资源管理器（Cluster Manager）的通信以及进行资源的申请、任务的分配和监控等。SparkContext会向资源管理器注册并申请运行Executor的资源

2. 资源管理器为Executor分配资源，并启动Executor进程，Executor运行情况将随着“心跳”发送到资源管理器上

3. SparkContext根据RDD的依赖关系构建DAG图，DAG图提交给DAG调度器（DAGScheduler）进行解析，将DAG图分解成多个“阶段”（每个阶段都是一个任务集），并且计算出各个阶段之间的依赖关系，然后把一个个“任务集”提交给底层的任务调度器（TaskScheduler）进行处理；Executor向SparkContext申请任务，任务调度器将任务分发给Executor运行，同时，SparkContext将应用程序代码发放给Executor

4. 任务在Executor上运行，把执行结果反馈给任务调度器，然后反馈给DAG调度器，运行完毕后写入数据并释放所有资源。

Spark架构特点

- 每个应用都有自己专属的Executor进程，并且该进程在应用运行期间一直驻留。Executor进程以多线程的方式运行任务，减少了多进程任务频繁的启动开销。

- Spark运行时与资源管理器没有关系，只要能获取executor并且能通讯就可以了

- Executor上有一个BlockManager存储模块，类似于键值存储系统（把内存和磁盘共同作为存储设备），在处理迭代计算任务时，不需要把中间结果写入到HDFS等文件系统，后续有需要时就可以直接读取，提高读写IO性能；

- BlockManager是spark自己的存储系统，RDD-Cache、broadcast 等的实现都是基于BlockManager来实现的，BlockManager也是分布式结构，在driver和所有executor上都会有blockmanager节点，每个节点上存储的block信息都会汇报给driver端的blockManagerMaster作统一管理，BlockManager对外提供get和set数据接口，可将数据存储在memory, disk, off-heap。

- 自动优化将程序向数据靠拢

Spark部署方式

- local模式
    - 在单机上运行，方便调试程序和学习
- 集群模型
    - Spark自带的集群管理器Standalone
    - Yarn
    - Mesos

Spark 实验环境

- AWS EC2 主机
- docker
- Jupyter Notebook

Spark 实验环境

- Spark提供Java、Scala和Python三种编程的语言，我们的课程以Python编程语言进行。如果需要其他编程语言的例子，可以参考spark官网的例子。
- https://spark.apache.org/examples.html

Python API and Java API

RDD API Examples

Word Count

In this example, we use a few transformations to build a dataset of (String, Int) pairs called `counts` and then save it to a file.

Python | Scala | Java

```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```


RDD API Examples

Word Count

In this example, we use a few transformations to build a dataset of (String, Int) pairs called `counts` and then save it to a file.

Python | Scala | Java

```java
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

大数据中的HelloWorld

- 统计文本中的单词个数
- Spark两种运行方式
  - 使用Spark Shell
  - 使用Java、Scala或者Python编写独立应用
    - Java与Scala需要在Maven中添加spark-core的依赖
    - Python中，编写脚本使用自带的./bin/spark-submit 提交任务

使用Spark Shell

- textFile加载数据文件
- flatMap遍历文件的每一行，遍历的时候调用lambda表达式
- flatMap之后使用map方法，将每个单词的个数赋值为1
- 调用reduceByKey进行统计。对RDD元素按照key进行分组，然后使用给定的函数。这里是对相同的key（单词）的多个value（1）进行累加。

```
>>> words_file = sc.textFile('../data/words')
>>> word_count = words_file.flatMap(lambda line: line.split(' '))\
... .map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)
>>> word_count.collect()
[('srv', 1), ('cs', 1), ('cmu', 3), ('cc', 1), ('sei', 1), ('edu!cis', 1), ('ohio-state', 3), ('edu!zaphod', 1), (
'edu!howland', 1), ('edu!destroyer!newsrelay', 1), ('edu!iscsvax', 1), ('uni', 1), ('edu!sunfish!pwiseman', 1), ('
Newsgroups:', 1), ('os', 1), ('misc', 1), ('Subject:', 1), ('roman', 1), ('01/14In', 1), ('response', 1), ('reques
ts', 1), ('am', 1), ('posting', 1), ('', 26), ('<C65FzK', 1), ('1M2@sunfish', 1), ('From:', 1), ('edu', 3), ('Date
:', 1), ('Tue,', 1), ('1993', 1), ('GMT', 1), ('pwiseman@sunfish', 1), ('Distribution:', 1), ('usa', 1), ('Univers
ity', 1), ('of', 3), ('Dakota', 1), ('978', 1), ('Due', 1), ('size', 2), ('is', 5), ('in', 3), ('14', 3), ('uuenco
```

使用Python脚本

```python
from pyspark import SparkContext

sc = SparkContext( 'local', 'Word Count')
textFile = sc.textFile('../data/words')
word_count = textFile.flatMap(lambda line: line.split(" "))\
                     .map(lambda word: (word,1))\
                     .reduceByKey(lambda a, b : a + b)
word_count.foreach(print)
```

```
drwxrwxrwx 5    500   500 4096 Jun  1 05:41 ./
drwsrwsr-x 1 jovyan users 4096 May 12 16:19 ../
drwxrwxr-x 2    500   500 4096 Jun  1 05:38 data/
drwxr-xr-x 2 jovyan users 4096 May 31 13:21 .ipynb_checkpoints/
-rwxrwxrwx 1    500   500 1182 May 31 13:04 jupyter_memo*
-rwxrwxr-x 1    500   500   99 Jun  1 05:41 run_docker_bash.sh*
-rwxrwxrwx 1    500   500  110 May 31 13:16 run_docker_notebook.sh*
drwxrwxr-x 2    500   500 4096 Jun  1 06:08 scripts/
jovyan@415f748c8c5b:~/work$ cd scripts/
jovyan@415f748c8c5b:~/work/scripts$ ll
total 20
drwxrwxr-x 2 500 500 4096 Jun  1 06:08 ./
drwxrwxrwx 5 500 500 4096 Jun  1 05:41 ../
-rw-r--r-- 1 500 500 2103 Jun  1 05:37 01_spark_overview.ipynb
-rw-rw-r-- 1 500 500  306 Jun  1 06:08 02_word_count.py
-rwxrwxrwx 1 500 500   26 Jun  1 05:39 pyspark.sh*
jovyan@415f748c8c5b:~/work/scripts$ python 02_word_count.py
19/06/01 06:08:35 WARN NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.prope
rties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setL
ogLevel(newLevel).
19/06/01 06:08:36 WARN Utils: Service 'SparkUI' could not bind on port 404
0. Attempting port 4041.
[Stage 0:>                                                          (0 + 1
) / 1]('Path:', 1)
('cantaloupe', 1)
('srv', 1)
('cs', 1)
```

使用Jupyter

使用Jupyter 统计文本中单词出现的个数

1. 导入pyspark包

[16]:
```
from pyspark import SparkContext
```

2. 创建SparkContext ¶

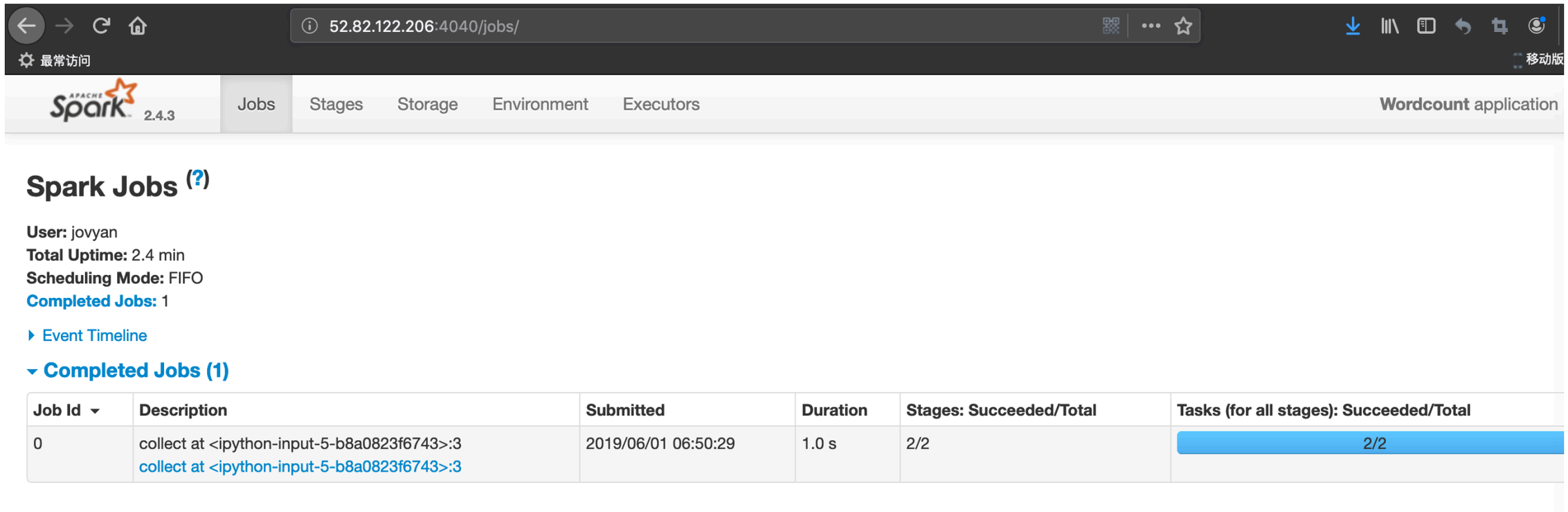
[19]:
```
sc = SparkContext('local', 'Wordcount')
```

[20]:
```
textFile = sc.textFile('../data/words')
word_count = textFile.flatMap(lambda line: line.split(" "))\
                     .map(lambda word: (word,1))\
                     .reduceByKey(lambda a, b : a + b)
```

[21]:
```
# foreach会打印到后台
#word_count.foreach(print)
word_count.collect()
```

[21]:
```
[('Path:', 1),
 ('cantaloupe', 1),
 ('srv', 1),
 ('cs', 1),
 ('cmu', 3),
 ('edu!magnesium', 1),
 ('club', 1),
 ('cc', 1),
```

Spark WebUI


52.82.122.206:4040/jobs/
最常访问
移动版
Spark 2.4.3
Jobs
Stages
Storage
Environment
Executors
Wordcount application
Spark Jobs (?)
User: jovyan
Total Uptime: 2.4 min
Scheduling Mode: FIFO
Completed Jobs: 1
Event Timeline
Completed Jobs (1)
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
0
collect at <ipython-input-5-b8a0823f6743>:3
collect at <ipython-input-5-b8a0823f6743>:3
2019/06/01 06:50:29
1.0 s
2/2
2/2