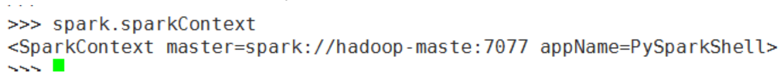
1.**SparkContext**是pyspark的编程入口，作业的提交，任务的分发，应用的注册都会在SparkContext中进行。一个SparkContext实例代表着和Spark的一个连接，只有建立了连接才可以把作业提交到集群中去。实例化了SparkContext之后才能创建RDD和Broadcast广播变量。

2.**Sparkcontext**获取，启动pyspark --master spark://hadoop-maste:7077 之后，可以通过SparkSession获取Sparkcontext对象



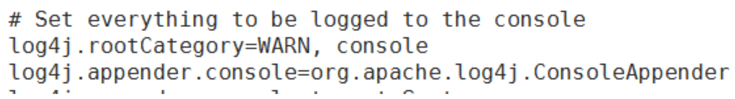
```
>>> spark.sparkContext
<SparkContext master=spark://hadoop-maste:7077 appName=PySparkShell>
>>>
```

从打印的记录来看，SparkContext它连接的Spark集群的master地址是spark://hadoop-maste:7077

另外一种获取SparkContext的方法是，引入pyspark.SparkContext进行创建。新建sparkContext.py文件，内容如下：

```
from pyspark import SparkContext
from pyspark import SparkConf
conf = SparkConf()
conf.set('master','local')
sparkContext = SparkContext(conf=conf)
rdd = sparkContext.parallelize(range(100))
print(rdd.collect())
sparkContext.stop()
```

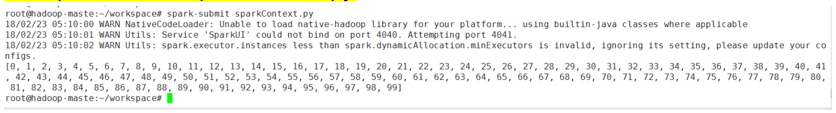
运行之前先将spark目录下的conf配置目录中的log4j.properties配置文件中的日志级别改为如下：



```
# Set everything to be logged to the console
log4j.rootCategory=WARN, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
```

这样后台打印的日志不至于太多印象查看！重启Spark集群

运行spark-submit sparkContext.py



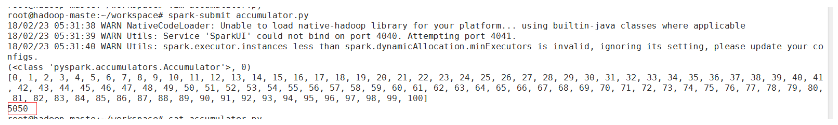
```
root@hadoop-maste:~/workspace# spark-submit sparkContext.py
18/02/23 05:10:00 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 05:10:01 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 05:10:02 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please update your co
nfigs.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41
, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
root@hadoop-maste:~/workspace#
```

上面代码中的Sparkconf对象是Spark里面用来配置参数的对象，接下来我们会详细讲解到。

3.**accumulator**是Sparkcontext上用来创建累加器的方法。创建的累加器可以在各个task中进行累加，并且只能够支持add操作。该方法支持传入累加器的初始值。这里以Accumulator累加器做1到50的加法作为讲解的例子。
新建accumulator.py文件，内容如下：

```
from pyspark import SparkContext,SparkConf
import numpy as np
conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)
acc = context.accumulator(0)
print(type(acc),acc.value)
rdd = context.parallelize(np.arange(101),5)
def acc_add(a):
    acc.add(a)
    return a
rdd2 = rdd.map(acc_add)
print(rdd2.collect())
print(acc.value)
context.stop()
```

使用spark-submit accumulator.py运行



```
root@hadoop-maste:~/workspace# spark-submit accumulator.py
18/02/23 05:31:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 05:31:39 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 05:31:40 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please update your co
nfigs.
(<class 'pyspark.accumulators.Accumulator'>, 0)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41
, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
5050
```

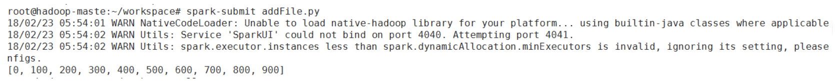
4.addFile方法添加文件，使用SparkFiles.get方法获取文件

这个方法接收一个路径，该方法将会把本地路径下的文件上传到集群中，以供运算过程中各个node节点下载数据，路径可以是本地路径也可是是hdfs路径，或者一个http,https,或者tfp的uri。如果上传的是一个文件夹，则指定recursize参数为True.上传的文件使用SparkFiles.get(filename)的方式进行获取。

新建addFile.py文件，内容如下：

```
from pyspark import SparkFiles
import os
import numpy as np
from pyspark import SparkContext
from pyspark import SparkConf
tempdir = '/root/workspace/'
path = os.path.join(tempdir,'num_data')
with open(path,'w') as f:
    f.write('100')
conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)
context.addFile(path)
rdd = context.parallelize(np.arange(10))
def fun(iterable):
    with open(SparkFiles.get('num_data')) as f:
        value = int(f.readline())
        return [x*value for x in iterable]
print(rdd.mapPartitions(fun).collect())
context.stop()
```

运行spark-submit addFile.py



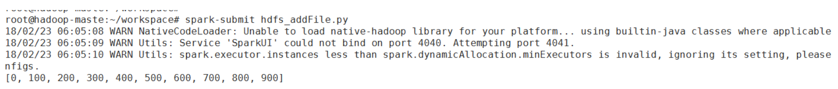
```
root@hadoop-maste:~/workspace# spark-submit addFile.py
18/02/23 05:54:01 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 05:54:02 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 05:54:02 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please
nfigs.
[0, 100, 200, 300, 400, 500, 600, 700, 800, 900]
```

这个例子是使用的本地的文件，接下来我们尝试一下hdfs路径下的文件。

新建hdfs_addFile.py文件，内容如下：

```
from pyspark import SparkFiles
import numpy as np
from pyspark import SparkContext
from pyspark import SparkConf
conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)
path = 'hdfs://hadoop-maste:9000/datas/num_data'
context.addFile(path)
rdd = context.parallelize(np.arange(10))
def fun(iterable):
    with open(SparkFiles.get('num_data')) as f:
        value = int(f.readline())
        return [x*value for x in iterable]
print(rdd.mapPartitions(fun).collect())
context.stop()
```

运行spark-submit hdfs_addFile.py



```
root@hadoop-maste:~/workspace# spark-submit hdfs_addFile.py
18/02/23 06:05:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 06:05:09 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 06:05:10 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please
nfigs.
[0, 100, 200, 300, 400, 500, 600, 700, 800, 900]
```

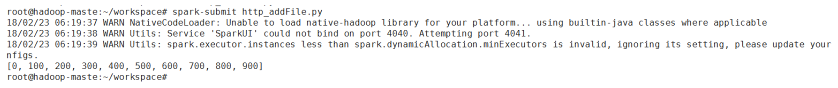
需要注意的是addFile默认识别本地路径，若是hdfs路径，需要指定hdfs://hadoop-maste:9000协议、uri及端口信息。

再来看一下读取网络文件信息。在182.150.37.49这台机器上，我安装了httpd服务器。httpd的配置及安装参考我的这个笔记：
http://note.youdao.com/noteshare?id=4c344eb8c78903d36a3ca9ae4bdc16ed&sub=F68ECF3191DB41EF9C0D9ED0104313A3
进入到安装了httpd服务器的机器的/var/www/html/目录，在这个目录中新建num_data文件，文件内容为100

然后编写http_addFile.py文件，在代码中读取刚才新建的httpd服务器上的num_data文件。
内容如下：

```
from pyspark import SparkFiles
import numpy as np
from pyspark import SparkContext
from pyspark import SparkConf
conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)
path = 'http://192.168.0.6:808/num_data'
context.addFile(path)
rdd = context.parallelize(np.arange(10))
def fun(iterable):
    with open(SparkFiles.get('num_data')) as f:
        value = int(f.readline())
        return [x*value for x in iterable]
print(rdd.mapPartitions(fun).collect())
context.stop()
```

运行spark-submit http_addFile.py

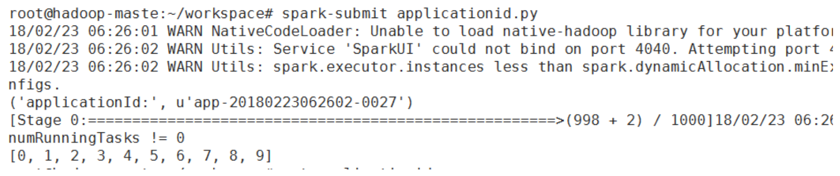


```
root@hadoop-maste:~/workspace# spark-submit http_addFile.py
18/02/23 06:19:37 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 06:19:38 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 06:19:39 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please update your
nfigs.
[0, 100, 200, 300, 400, 500, 600, 700, 800, 900]
root@hadoop-maste:~/workspace#
```

从上面的三个例子中，可以看出addFile方法的强大之处。借助该方法，在pyspark里任务运行中可以读取几乎任何位置的文件来参与计算！

5.applicationId，用于获取注册到集群的应用的id

```
from pyspark import SparkContext ,SparkConf
import numpy as np
conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)
rdd = context.parallelize(np.arange(10))
print('applicationId:',context.applicationId)
print(rdd.collect())
context.stop()
```



```
root@hadoop-maste:~/workspace# spark-submit applicationid.py
18/02/23 06:26:01 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfo
18/02/23 06:26:02 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 
18/02/23 06:26:02 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minE
nfigs.
('applicationId:', u'app-20180223062602-0027')
[Stage 0:=====================================================>(998 + 2) / 1000]18/02/23 06:2
numRunningTasks != 0
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

6.binaryFiles读取二进制文件。
该方法用于读取二进制文件例如音频、视频、图片，对于每个文件器返回一个tuple，tuple的第一个元素为文件的路径，第二个参数为二进制文件的内容。
我在hdfs的/datas/pics文件目录下上传了两张图片，使用binaryFiles读取/datas/pics目录中的二进制图片数据

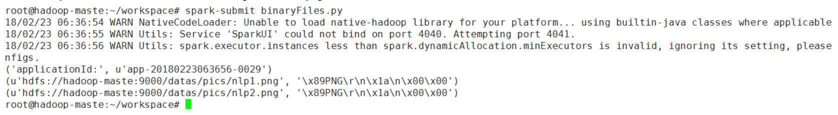
新建binaryFiles.py文件，内容如下：

```
from pyspark import SparkContext ,SparkConf
import numpy as np

conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)

rdd = context.binaryFiles('/datas/pics/')
print('applicationId:',context.applicationId)
result = rdd.collect()
for data in result:
    print(data[0],data[1][:10])
context.stop()
```

运行spark-submit binaryFiles.py



```
root@hadoop-maste:~/workspace# spark-submit binaryFiles.py
18/02/23 06:36:54 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 06:36:55 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 06:36:56 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please
nfigs.
('applicationId:', u'app-20180223063656-0029')
(u'hdfs://hadoop-maste:9000/datas/pics/nlp1.png', '\x89PNG\r\n\x1a\n\x00\x00')
(u'hdfs://hadoop-maste:9000/datas/pics/nlp2.png', '\x89PNG\r\n\x1a\n\x00\x00')
root@hadoop-maste:~/workspace#
```

该方法对于读取二进制文件的数据非常方便，特别是处理图片、音视频的时候。

7.**broadcast广播变量**

SparkContext上的broadcast方法用于创建广播变量，对于大于5M的共享变量，推荐使用广播。广播机制可以最大限度的减少网络IO，从而提升性能。

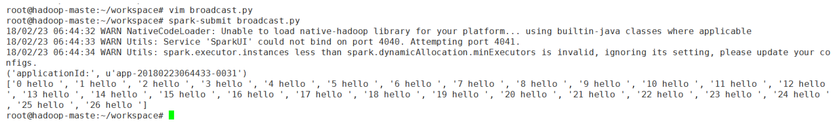
接下来例子中，广播一个 ‘hello’ 字符串，在各个task中接收广播变量，拼接返回。新建broadcast.py文件，内容如下。

```
from pyspark import SparkContext ,SparkConf
import numpy as np

conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)
broad = context.broadcast(' hello ')
rdd = context.parallelize(np.arange(27),3)
print('applicationId:',context.applicationId)
print(rdd.map(lambda x:str(x)+broad.value).collect())

context.stop()
```

运行spark-submit broadcast.py

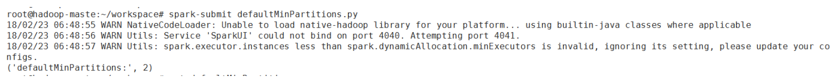


```
root@hadoop-maste:~/workspace# vim broadcast.py
root@hadoop-maste:~/workspace# spark-submit broadcast.py
18/02/23 06:44:32 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 06:44:33 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 06:44:34 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please update your co
nfigs.
('applicationId:', u'app-20180223064433-0031')
['0 hello ', '1 hello ', '2 hello ', '3 hello ', '4 hello ', '5 hello ', '6 hello ', '7 hello ', '8 hello ', '9 hello ', '10 hello ', '11 hello ', '12 hello
', '13 hello ', '14 hello ', '15 hello ', '16 hello ', '17 hello ', '18 hello ', '19 hello ', '20 hello ', '21 hello ', '22 hello ', '23 hello ', '24 hello '
, '25 hello ', '26 hello ']
root@hadoop-maste:~/workspace#
```

从结果来看，分布式运行中的每个任务中都接收到了广播的变量hello.

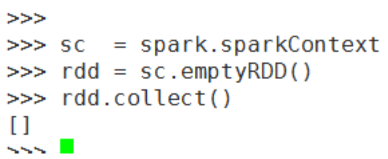
8.defaultMinPartitions 获取默认最小的分区数

```
from pyspark import SparkContext ,SparkConf
import numpy as np
conf = SparkConf()
conf.set('master','spark://hadoop-maste:7077')
context = SparkContext(conf=conf)
print('defaultMinPartitions:',context.defaultMinPartitions)
context.stop()
```



```
root@hadoop-maste:~/workspace# spark-submit defaultMinPartitions.py
18/02/23 06:48:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/02/23 06:48:56 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/02/23 06:48:57 WARN Utils: spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setting, please update your co
nfigs.
('defaultMinPartitions:', 2)
```

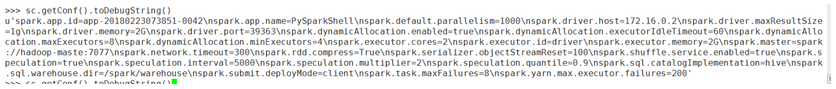
9.emptyRDD创建一个空的RDD ，该RDD没有分区，也没有任何数据



```
>>>
>>> sc  = spark.sparkContext
>>> rdd = sc.emptyRDD()
>>> rdd.collect()
[]
>>>
```

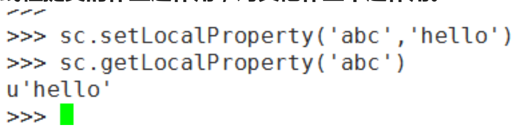
10.getConf()方法返回作业的配置对象信息

sc.getConf().toDebugString()



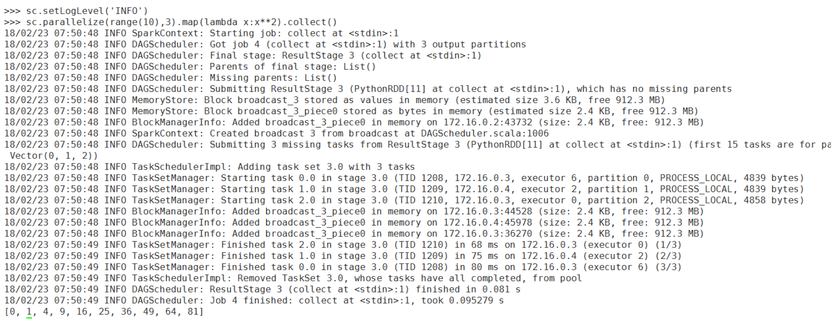
```
>>> sc.getConf().toDebugString()
u'spark.app.id=app-20180223073851-0042\nspark.app.name=PySparkShell\nspark.default.parallelism=1000\nspark.driver.host=172.16.0.2\nspark.driver.maxResultSize
=1g\nspark.driver.memory=2G\nspark.driver.port=39363\nspark.dynamicAllocation.enabled=true\nspark.dynamicAllocation.executorIdleTimeout=60\nspark.dynamicAllo
cation.maxExecutors=8\nspark.dynamicAllocation.minExecutors=4\nspark.executor.cores=2\nspark.executor.id=driver\nspark.executor.memory=2G\nspark.master=spark
://hadoop-maste:7077\nspark.network.timeout=300\nspark.rdd.compress=True\nspark.serializer.objectStreamReset=100\nspark.shuffle.service.enabled=true\nspark.s
peculation=true\nspark.speculation.interval=5000\nspark.speculation.multiplier=2\nspark.speculation.quantile=0.9\nspark.sql.catalogImplementation=hive\nspark
.sql.warehouse.dir=/spark/warehouse\nspark.submit.deployMode=client\nspark.task.maxFailures=8\nspark.yarn.max.executor.failures=200'
>>> sc.getConf().toDebugString()
```

11.getLocalProperty和setLocalProperty获取和设置在本地线程中的属性信息。通过setLocalProperty设置，设置的属性只能对当前线程提交的作业起作用，对其他作业不起作用。

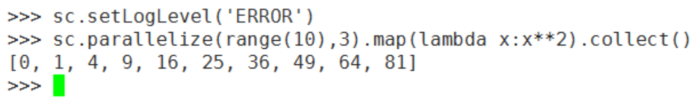


```
>>>
>>> sc.setLocalProperty('abc','hello')
>>> sc.getLocalProperty('abc')
u'hello'
>>>
```

12.setLogLevel设置日志级别，通过这个设置将会覆盖任何用户自定义的日志等级设置。取值有：ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE, WARN



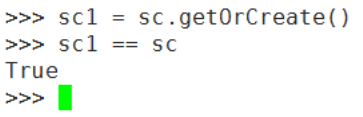
```
>>> sc.setLogLevel('INFO')
>>> sc.parallelize(range(10),3).map(lambda x:x**2).collect()
18/02/23 07:50:48 INFO SparkContext: Starting job: collect at <stdin>:1
18/02/23 07:50:48 INFO DAGScheduler: Got job 4 (collect at <stdin>:1) with 3 output partitions
18/02/23 07:50:48 INFO DAGScheduler: Final stage: ResultStage 3 (collect at <stdin>:1)
18/02/23 07:50:48 INFO DAGScheduler: Parents of final stage: List()
18/02/23 07:50:48 INFO DAGScheduler: Missing parents: List()
18/02/23 07:50:48 INFO DAGScheduler: Submitting ResultStage 3 (PythonRDD[11] at collect at <stdin>:1), which has no missing parents
18/02/23 07:50:48 INFO MemoryStore: Block broadcast_3 stored as values in memory (estimated size 3.6 KB, free 912.3 MB)
18/02/23 07:50:48 INFO MemoryStore: Block broadcast_3_piece0 stored as bytes in memory (estimated size 2.4 KB, free 912.3 MB)
18/02/23 07:50:48 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 172.16.0.2:43732 (size: 2.4 KB, free: 912.3 MB)
18/02/23 07:50:48 INFO SparkContext: Created broadcast 3 from broadcast at DAGScheduler.scala:1006
18/02/23 07:50:48 INFO DAGScheduler: Submitting 3 missing tasks from ResultStage 3 (PythonRDD[11] at collect at <stdin>:1) (first 15 tasks are for pa
 Vector(0, 1, 2))
18/02/23 07:50:48 INFO TaskSchedulerImpl: Adding task set 3.0 with 3 tasks
18/02/23 07:50:48 INFO TaskSetManager: Starting task 0.0 in stage 3.0 (TID 1208, 172.16.0.3, executor 6, partition 0, PROCESS_LOCAL, 4839 bytes)
18/02/23 07:50:48 INFO TaskSetManager: Starting task 1.0 in stage 3.0 (TID 1209, 172.16.0.4, executor 2, partition 1, PROCESS_LOCAL, 4839 bytes)
18/02/23 07:50:48 INFO TaskSetManager: Starting task 2.0 in stage 3.0 (TID 1210, 172.16.0.3, executor 0, partition 2, PROCESS_LOCAL, 4858 bytes)
18/02/23 07:50:48 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 172.16.0.3:44528 (size: 2.4 KB, free: 912.3 MB)
18/02/23 07:50:48 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 172.16.0.4:45978 (size: 2.4 KB, free: 912.3 MB)
18/02/23 07:50:48 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 172.16.0.3:36270 (size: 2.4 KB, free: 912.3 MB)
18/02/23 07:50:49 INFO TaskSetManager: Finished task 2.0 in stage 3.0 (TID 1210) in 68 ms on 172.16.0.3 (executor 0) (1/3)
18/02/23 07:50:49 INFO TaskSetManager: Finished task 1.0 in stage 3.0 (TID 1209) in 75 ms on 172.16.0.4 (executor 2) (2/3)
18/02/23 07:50:49 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 1208) in 80 ms on 172.16.0.3 (executor 6) (3/3)
18/02/23 07:50:49 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
18/02/23 07:50:49 INFO DAGScheduler: ResultStage 3 (collect at <stdin>:1) finished in 0.081 s
18/02/23 07:50:49 INFO DAGScheduler: Job 4 finished: collect at <stdin>:1, took 0.095279 s
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



```
>>> sc.setLogLevel('ERROR')
>>> sc.parallelize(range(10),3).map(lambda x:x**2).collect()
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

通过对比两种不同的日志级别的输出，可以看出不同的日志级别的日志输出量是不同的，可以通过这一点选择合适的日志级别进行调试。

13.getOrCreate得到或者是创建一个SparkContext对象，该方法创建的SparkContext对象为单例对象。该方法可以接受一个Sparkconf对象。

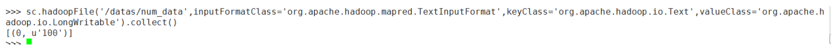


```
>>> sc1 = sc.getOrCreate()
>>> sc1 == sc
True
>>>
```

14.hadoopFile读取‘老’的hadoop接口提供hdfs文件格式.

sc.hadoopFile('/datas/num_data',inputFormatClass='org.apache.hadoop.mapred.TextInputFormat',keyClass='org.apache.hadoo

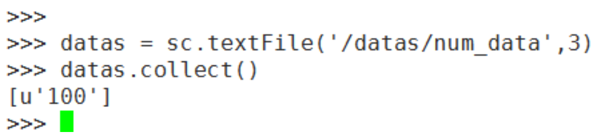
第一个参数为文件路径，第二个参数为输入文件的格式，第三个参数为键的格式，第四个参数为值的格式



```
>>> sc.hadoopFile('/datas/num_data',inputFormatClass='org.apache.hadoop.mapred.TextInputFormat',keyClass='org.apache.hadoop.io.Text',valueClass='org.apache.h
adoop.io.LongWritable').collect()
[(0, u'100')]
>>>
```

读取出来默认会把行号作为键！

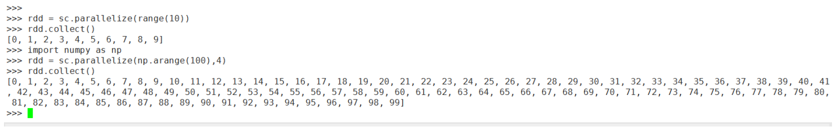
15.textFile和saveAsTextFile读取位于HDFS上的文本文件



```
>>>
>>> datas = sc.textFile('/datas/num_data',3)
>>> datas.collect()
[u'100']
>>>
```

这个方法读取位于hdfs上的文本类型的文件最简单

16.**parallelize使用python集合创建RDD，可以使用range函数，当然也可也使用numpy里面的arange方法来创建**

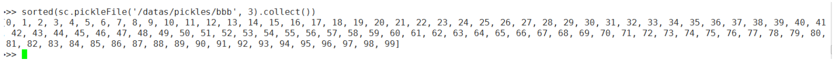


```
>>>
>>> rdd = sc.parallelize(range(10))
>>> rdd.collect()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> import numpy as np
>>> rdd = sc.parallelize(np.arange(100),4)
>>> rdd.collect()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41
, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
>>>
```

17.saveAsPickleFile和pickleFile将RDD保存为python中的pickle压缩文件格式。

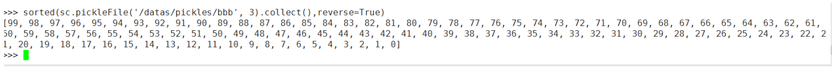
sc.parallelize(range(100),3).saveAsPickleFile('/datas/pickles/bbb', 5)

sorted(sc.pickleFile('/datas/pickles/bbb', 3).collect())



```
>> sorted(sc.pickleFile('/datas/pickles/bbb', 3).collect())
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41
 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
>>
```

sorted方法中使用关键字参数reverse，设置为True



```
>>> sorted(sc.pickleFile('/datas/pickles/bbb', 3).collect(),reverse=True)
[99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61,
50, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 2
1, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>>
```

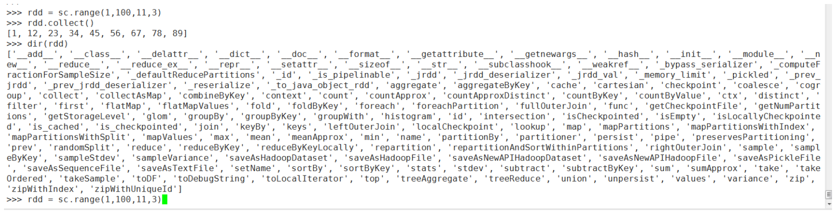
18.range(start, end=None, step=1, numSlices=None)按照提供的起始值和步长，创建RDD

numSlices用于指定分区数

rdd = sc.range(1,100,11,3)

rdd.collect()

dir(rdd)



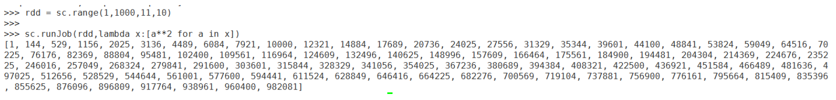
```
>>> rdd = sc.range(1,100,11,3)
>>> rdd.collect()
[1, 12, 23, 34, 45, 56, 67, 78, 89]
>>> dir(rdd)
['__add__', '__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__getnewargs__', '__hash__', '__init__', '__module__', '__n
ew__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_bypass_serializer', '_computeF
ractionForSampleSize', '_defaultReducePartitions', '_id', '_is_pipelinable', '_jrdd', '_jrdd_deserializer', '_jrdd_val', '_memory_limit', '_pickled', '_prev_
jrdd', '_prev_jrdd_deserializer', '_reserialize', '_to_java_object_rdd', 'aggregate', 'aggregateByKey', 'cache', 'cartesian', 'checkpoint', 'coalesce', 'cogr
oup', 'collect', 'collectAsMap', 'combineByKey', 'context', 'count', 'countApprox', 'countApproxDistinct', 'countByKey', 'countByValue', 'ctx', 'distinct', '
filter', 'first', 'flatMap', 'flatMapValues', 'fold', 'foldByKey', 'foreach', 'foreachPartition', 'fullOuterJoin', 'func', 'getCheckpointFile', 'getNumPartit
ions', 'getStorageLevel', 'glom', 'groupBy', 'groupByKey', 'groupWith', 'histogram', 'id', 'intersection', 'isCheckpointed', 'isEmpty', 'isLocallyCheckpointe
d', 'is_cached', 'is_checkpointed', 'join', 'keyBy', 'keys', 'leftOuterJoin', 'localCheckpoint', 'lookup', 'map', 'mapPartitions', 'mapPartitionsWithIndex',
'mapPartitionsWithSplit', 'mapValues', 'max', 'mean', 'meanApprox', 'min', 'name', 'partitionBy', 'partitioner', 'persist', 'pipe', 'preservesPartitioning',
'prev', 'randomSplit', 'reduce', 'reduceByKey', 'reduceByKeyLocally', 'repartition', 'repartitionAndSortWithinPartitions', 'rightOuterJoin', 'sample', 'sampl
eByKey', 'sampleStdev', 'sampleVariance', 'saveAsHadoopDataset', 'saveAsHadoopFile', 'saveAsNewAPIHadoopDataset', 'saveAsNewAPIHadoopFile', 'saveAsPickleFile
', 'saveAsSequenceFile', 'saveAsTextFile', 'setName', 'sortBy', 'sortByKey', 'stats', 'stdev', 'subtract', 'subtractByKey', 'sum', 'sumApprox', 'take', 'take
Ordered', 'takeSample', 'toDF', 'toDebugString', 'toLocalIterator', 'top', 'treeAggregate', 'treeReduce', 'union', 'unpersist', 'values', 'variance', 'zip',
'zipWithIndex', 'zipWithUniqueId']
>>> rdd = sc.range(1,100,11,3)
```

19.runJob(rdd, partitionFunc, partitions=None, allowLocal=False)在给定的分区上运行指定的函数

partitions用于指定分区的编号，是一个列表。若不指定分区默认为所有分区运行partitionFunc函数

rdd = sc.range(1,1000,11,10)

sc.runJob(rdd,lambda x:[a for a in x])

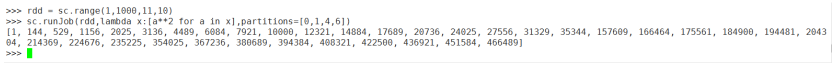


```
>>> rdd = sc.range(1,1000,11,10)
>>>
>>> sc.runJob(rdd,lambda x:[a**2 for a in x])
[1, 144, 529, 1156, 2025, 3136, 4489, 6084, 7921, 10000, 12321, 14884, 17689, 20736, 24025, 27556, 31329, 35344, 39601, 44100, 48841, 53824, 59049, 64516, 70
225, 76176, 82369, 88804, 95481, 102400, 109561, 116964, 124609, 132496, 140625, 148996, 157609, 166464, 175561, 184900, 194481, 204304, 214369, 224676, 2352
25, 246016, 257049, 268324, 279841, 291600, 303601, 315844, 328329, 341056, 354025, 367236, 380689, 394384, 408321, 422500, 436921, 451584, 466489, 481636, 4
97025, 512656, 528529, 544644, 561001, 577600, 594441, 611524, 628849, 646416, 664225, 682276, 700569, 719104, 737881, 756900, 776161, 795664, 815409, 835396
, 855625, 876096, 896809, 917764, 938961, 960400, 982081]
```

指定在0,1,4,6分区上运行a**2函数

rdd = sc.range(1,1000,11,10)

sc.runJob(rdd,lambda x:[a**2 for a in x],partitions=[0,1,4,6])



```
>>> rdd = sc.range(1,1000,11,10)
>>> sc.runJob(rdd,lambda x:[a**2 for a in x],partitions=[0,1,4,6])
[1, 144, 529, 1156, 2025, 3136, 4489, 6084, 7921, 10000, 12321, 14884, 17689, 20736, 24025, 27556, 31329, 35344, 157609, 166464, 175561, 184900, 194481, 2043
04, 214369, 224676, 235225, 354025, 367236, 380689, 394384, 408321, 422500, 436921, 451584, 466489]
>>>
```

20.setCheckpointDir(dirName)设置检查点的目录，检查点用于异常发生时错误的恢复，该目录必须为HDFS目录。

设置检查点目录为/datas/checkpoint/

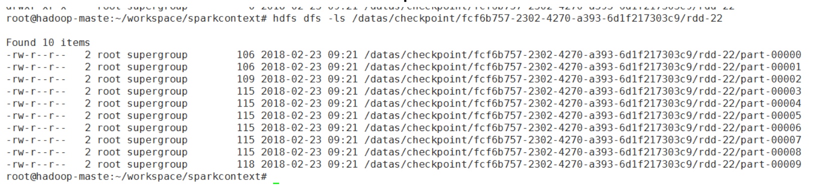
sc.setCheckpointDir('/datas/checkpoint/')

rdd = sc.range(1,1000,11,10)

rdd.checkpoint()

rdd.collect()

运行完成之后，查看hdfs 的/datas/checkpoint目录



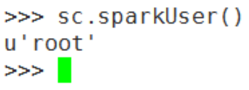
```
root@hadoop-maste:~/workspace/sparkcontext# hdfs dfs -ls /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22

Found 10 items
-rw-r--r--   2 root supergroup        106 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00000
-rw-r--r--   2 root supergroup        106 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00001
-rw-r--r--   2 root supergroup        109 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00002
-rw-r--r--   2 root supergroup        115 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00003
-rw-r--r--   2 root supergroup        115 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00004
-rw-r--r--   2 root supergroup        115 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00005
-rw-r--r--   2 root supergroup        115 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00006
-rw-r--r--   2 root supergroup        115 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00007
-rw-r--r--   2 root supergroup        115 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00008
-rw-r--r--   2 root supergroup        118 2018-02-23 09:21 /datas/checkpoint/fcf6b757-2302-4270-a393-6d1f217303c9/rdd-22/part-00009
root@hadoop-maste:~/workspace/sparkcontext#
```

发现有运行过程中的数据被保存下来，在Spark程序运行过程中若发生异常，将会使用检查点数据来恢复异常。

21.**sparkUser获取运行当前作业的用户名**

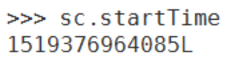
sc.sparkUser()



```
>>> sc.sparkUser()
u'root'
>>> 
```

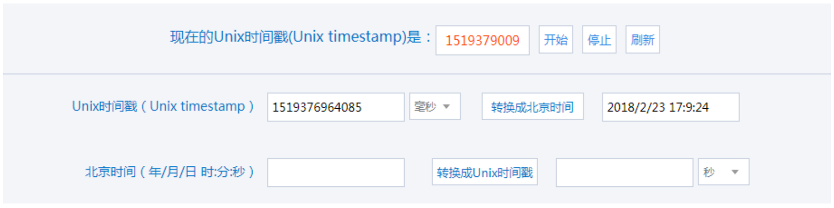
22.startTime返回作业启动的时间

sc.startTime



```
>>> sc.startTime
1519376964085L
```

它返回的是Long类型的毫秒时间值，可借助在线时间转换工具查看具体时间



现在的Unix时间戳(Unix timestamp)是： 1519379009 开始 停止 刷新

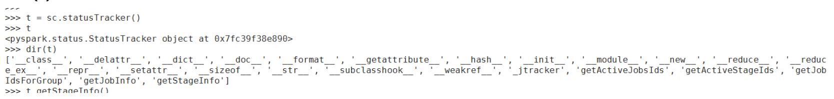
Unix时间戳（Unix timestamp） 1519376964085 毫秒 转换成北京时间 2018/2/23 17:9:24

北京时间（年/月/日 时:分:秒） 转换成Unix时间戳 秒

23.statusTracker()方法用于获取StatusTracker对象，通过该对象可以获取活动的Jobs的id，活动的stage 的id。job的信息，stage的信息。可以使用这个对象来实时监控作业运行的中间状态数据。

t = sc.statusTracker()

dir(t)

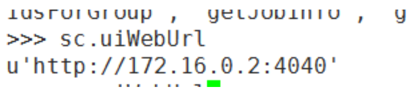


```
>>> t = sc.statusTracker()
>>> t
<pyspark.status.StatusTracker object at 0x7fc39f38e890>
>>> dir(t)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduc
e_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_jtracker', 'getActiveJobsIds', 'getActiveStageIds', 'getJob
IdsForGroup', 'getJobInfo', 'getStageInfo']
>>> t.getStageInfo()
```

24.stop()方法用于停止SparkContext和cluster的连接。一般在书写程序最后一行都要加上这句话，确保作业运行完成之后连接和cluster集群断开。

25.uiWebUrl返回web的url

sc.uiWebUrl



```
IdsForGroup', 'getJobInfo', 'g
>>> sc.uiWebUrl
u'http://172.16.0.2:4040'
```

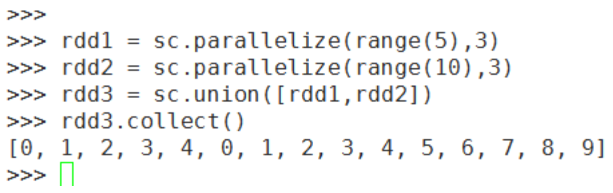
26.union(rdds)用合并多个rdd为一个rdd

rdd1 = sc.parallelize(range(5),4)

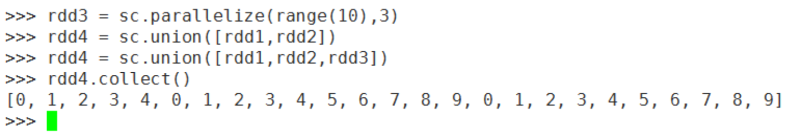
rdd2 = sc.parallelize(range(10),3)

rdd3 = sc.union([rdd1,rdd2])

rdd3.collect()



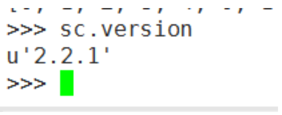
```
>>>
>>> rdd1 = sc.parallelize(range(5),3)
>>> rdd2 = sc.parallelize(range(10),3)
>>> rdd3 = sc.union([rdd1,rdd2])
>>> rdd3.collect()
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 
```



```
>>> rdd3 = sc.parallelize(range(10),3)
>>> rdd4 = sc.union([rdd1,rdd2])
>>> rdd4 = sc.union([rdd1,rdd2,rdd3])
>>> rdd4.collect()
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 
```

27.version获取版本号

sc.version



```
>>> sc.version
u'2.2.1'
>>> 
```