RDD编程

RDD Programing

RDD基础

RDD基础

- 弹性分布式数据集是Spark对数据的核心抽象
- 在Spark中对数据的所有操作不外乎是创建RDD、转换RDD以及调用RDD进行求值
- RDD会自动将RDD中的数据分发到集群上，并将操作并行化执行

RDD基础

- RDD是一个不可变的分布式对象集合
- 每个RDD都被分成多个分区，这些分区运行在集群的不同节点上。
- RDD可以包含Java、Scala和Python中任意类型的对象

RDD创建方式

- 方式1
  - 在driver中分发driver程序中的对象集合（比如list或set）
- 方式2
  - 读取一个外部数据集

RDD创建方式

- 使用SparkContext的parallelize()方法，把程序中一个已有的集合转换为RDD

- 除了开发运行、测试和学习阶段外很少使用。因为这种方式需要把所有数据放到一台机器的内存中

RDD创建方式

使用SparkContext的parallelize()方法将程序中存在的集合转换为RDD

[2]:
```
lines = sc.parallelize(['pandas', 'i like pandas'])
```

[8]:
```
type(lines)
```

[8]:
```
pyspark.rdd.RDD
```

[9]:
```
lines.collect()
```

[9]:
```
['pandas', 'i like pandas']
```

RDD创建方式

- 更加常用的方式是从外部数据读取

使用sc.textFile()将外部文件转换为RDD

[10]:
```
jupyter_log = sc.textFile('../data/jupyter.log')
```

[11]:
```
jupyter_log.count()
```

[11]: 98

RDD操作

- 转换操作（Transformation）
  - 基于现有的数据集创建一个新的数据集
- 行动操作（Action）
  - 在数据集上进行运算，返回计算值

RDD转换操作

- 转换操作都会产生不同的RDD
- 转换得到的RDD是惰性求值。
- 整个转换过程只是记录了转换的轨迹，并不会发生真正的计算，只有遇到行动操作时，才会发生真正的计算，

转换操作是惰性求值，只有遇到行动操作之后才会计算

[12]:
```
# 文件不存在，运行后并不会报错
jupyter_log = sc.textFile('../data/jupyter.log_')
```

[13]:
```
# 执行行动操作后才会报错
jupyter_log.count()
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-13-0c03b5e0b389> in <module>
      1 # 执行行动操作后才会报错
----> 2 jupyter_log.count()
```

RDD转换操作

转换操作例子，取log中包含'JupyterLab'的行数，使用转换操作filter()

[19]:
```
jupyter_log = sc.textFile('../data/jupyter.log')
# filter操作不会改变=已有的jupyter_log中的数据，而是创建了一个新的RDD，jupyter_log在后续的实验中还会继续使用
jupyter_rdd = jupyter_log.filter(lambda line: 'JupyterLab' in line)
```

[20]:
```
jupyter_rdd.count()
```

[20]:
```
10
```

[21]:
```
jupyter_rdd.collect()
```

[21]:
```
['[I 06:48:27.829 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:48:27.829 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab',
 '[W 06:48:27.830 LabApp] JupyterLab server extension not enabled, manually loading...',
 '[I 06:48:27.836 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:48:27.836 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab',
 '[I 06:49:07.791 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:49:07.791 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab',
 '[W 06:49:07.792 LabApp] JupyterLab server extension not enabled, manually loading...',
 '[I 06:49:07.797 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:49:07.798 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab']
```

RDD行动操作

- 行动操作是真正触发计算的地方。Spark程序执行到行动操作时，才会执行真正的计算

- 将最终结果返回到driver程序中，或者写入到外部数据系统中

RDD行动操作

- 行动操作是真正触发计算的地方。Spark程序执行到行动操作时，才会执行真正的计算

- 将最终结果返回到driver程序中，或者写入到外部数据系统中

RDD行动操作

[20]:
```
# count()是行动操作，返回RDD中元素的数目
jupyter_rdd.count()
```

[20]:
```
10
```

[21]:
```
# collect()也是行动操作
jupyter_rdd.collect()
```

[21]:
```
['[I 06:48:27.829 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:48:27.829 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab',
 '[W 06:48:27.830 LabApp] JupyterLab server extension not enabled, manually loading...',
 '[I 06:48:27.836 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:48:27.836 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab',
 '[I 06:49:07.791 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:49:07.791 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab',
 '[W 06:49:07.792 LabApp] JupyterLab server extension not enabled, manually loading...',
 '[I 06:49:07.797 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab',
 '[I 06:49:07.798 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab']
```

[22]:
```
# 使用行动操作take() 从RDD中获得一些例子
```

[23]:
```
for line in jupyter_rdd.take(5):
    print(line)
```

```
[I 06:48:27.829 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab
[I 06:48:27.829 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[W 06:48:27.830 LabApp] JupyterLab server extension not enabled, manually loading...
[I 06:48:27.836 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab
[I 06:48:27.836 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
```

惰性求值

- 正如前面所说，RDD的转换操作都是惰性求值

- 我们不应该把RDD看作存放着特定数据的数据集，而是应该把RDD看作是，我们通过转化操作构建出来的、记录如何计算数据集的指令列表。

- 因此，当我们读取数据的时候，并不是执行textFile()就会加载数据，而是执行到行动操作时才会加载数据。

向Spark传递函数

向Spark传递函数

- 绝大部分的转化操作和一部分行动操作，都需要依赖自己定义的函数来计算

- 三种编程语言会有不同

向Spark传递函数

- 在Python中有三种方式向Spark传递函数
  - 函数比较短的时候，使用lambda表达式
  - 传递顶层函数或者局部函数

向Spark传递函数

向Spark传递函数

使用lambda表达式

[31]:
```
jupyter_rdd = jupyter_log.filter(lambda line: 'JupyterLab' in line)
jupyter_rdd.first()
```

[31]:
```
'[I 06:48:27.829 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab'
```

[27]:
```
# 定义函数 如果包含JupyterLab则返回True, 不包含则返回False
def containJupyterLab(s):
    return 'JupyterLab' in s
```

[28]:
```
jupyter_rdd = jupyter_log.filter(containJupyterLab)
```

[30]:
```
jupyter_rdd.first()
```

[30]:
```
'[I 06:48:27.829 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab'
```

向Spark传递函数

传递函数的时候需要小心，Python会在不经意间把函数所在的整个对象也序列化传出去。
当你床底的对象时某一个对象的成员时，或者包含了某个对象中一个字段的引用时（例如self.field），
Spark会将整个对象发到worker上，有可能比预想要发送的内容大。
有时候传递的内容，***包含python不知道如何序列化传输的对象***，也会导致程序失败。

[32]:
```python
# 传递一个带字段引用的函数（这是一个错误的例子，不可以这么做!!!）
class SearchFunctions():
    def __init__(self, qurey):
        self.qurey = qurey

    def isMatch(self, s):
        return self.query in s

    def getMatchesFunctionReference(self, rdd):
        # 注意!  在self.isMatch中引用了整个self
        return rdd.filter(self.isMatch)

    def getMatchesMemberReference(self, rdd):
        # 注意! 在self.query中引用了整个self
        return rdd.filter(lambda x: self.query in x)
```

向Spark传递函数

传递函数的时候需要小心，Python会在不经意间把函数所在的整个对象也序列化传出去。
当你床底的对象时某一个对象的成员时，或者包含了某个对象中一个字段的引用时（例如self.field），
Spark会将整个对象发到worker上，有可能比预想要发送的内容大。
有时候传递的内容，***包含python不知道如何序列化传输的对象***，也会导致程序失败。

[32]:
```python
# 传递一个带字段引用的函数（这是一个错误的例子，不可以这么做!!!）
class SearchFunctions():
    def __init__(self, qurey):
        self.qurey = qurey

    def isMatch(self, s):
        return self.query in s

    def getMatchesFunctionReference(self, rdd):
        # 注意!  在self.isMatch中引用了整个self
        return rdd.filter(self.isMatch)

    def getMatchesMemberReference(self, rdd):
        # 注意! 在self.query中引用了整个self
        return rdd.filter(lambda x: self.query in x)
```

常见的转换与行动操作

常见的转换操作

常用转换操作

1. map(func)
   将每个元素传递到函数func中，并将结果返回为一个新的数据集

[9]:
```
rdd = sc.parallelize([1, 2, 3, 3])
rdd.map(lambda x: x + 1).collect()
```

[9]:
```
[2, 3, 4, 4]
```

2. flatMap(func)
   map()相似，但每个输入元素都可以映射到0或多个输出结果。通常用来切分单词。有点像像将先映射，然后在拍遍

[7]:
```
rdd = sc.parallelize(['a b', 'd e', 'ef', 'hi j k'])
```

[8]:
```
rdd.flatMap(lambda x: x.split(' ')).collect()
```

[8]:
```
['a', 'b', 'd', 'e', 'ef', 'hi', 'j', 'k']
```

常见的转换操作

3. filter(func)

   筛选出满足函数func的元素，并返回一个新的数据集

[11]:
```
rdd = sc.parallelize([1, 2, 3, 3])
rdd.filter(lambda x: x > 1).collect()
```

[11]:
```
[2, 3, 3]
```

4. distinct()

   去重

[12]:
```
rdd = sc.parallelize([1, 2, 3, 3])
rdd.distinct().collect()
```

[12]:
```
[1, 2, 3]
```

常见的行动操作

常用行动操作

1. reduce()
   并行整合rdd中的所有元素

[3]:
```
rdd = sc.parallelize([1, 2, 3, 3])
rdd.reduce(lambda x, y: x + y)
```

[3]:
```
9
```

2. count()
   返回数据集中元素的个数

[16]:
```
rdd.count()
```

[16]:
```
4
```

3. collect()
   以数组的形式返回数据集中的所有元素。一般只用于少量数据，因为要把全部数据放到内存中。

[17]:
```
rdd.collect()
```

[17]:
```
[1, 2, 3, 3]
```

常见的行动操作

4. first()

   返回数据集中的第一个元素

[18]:
```
rdd.first()
```

[18]:
```
1
```

5. take(n)

   以数组的形式返回数据集中的前n个元素。

[19]:
```
rdd.take(2)
```

[19]:
```
[1, 2]
```

6. foreach(func)

   将数据集中的每个元素传递到函数func中运行

[9]:
```
def show(x):
    print(x)
rdd.foreach(show)
```

持久化（缓存）

持久化（缓存）

- RDD采用惰性求值的机制，每次遇到行动操作，都会从头开始执行计算。在一些情形下，我们需要多次调用不同的行动操作。

- 可以通过持久化（缓存）机制避免这种重复计算的开销。可以使用persist()方法对一个RDD标记为持久化，之所以说“标记为持久化”，是因为出现persist()语句的地方，并不会马上计算生成RDD并把它持久化，而是要等到遇到第一个行动操作触发真正计算以后，才会把计算结果进行持久化，持久化后的RDD将会被保留在计算节点的内存中被后面的行动操作重复使用。

- 可以使用unpersist()方法手动地把持久化的RDD从缓存中移除

持久化（缓存）

持久化（缓存）

[12]:
```
rdd = sc.parallelize(['spark', 'hadoop'])
# 会调用persist(MEMORY_ONLY)，但是，语句执行到这里，并不会缓存rdd，这是rdd还没有被计算生成
rdd.cache()
# 第一次行动操作，触发一次真正从头到尾的计算，这时才会执行上面的rdd.cache()，把这个rdd放到缓存中
rdd.count()
```

[12]:
```
2
```

[13]:
```
# 第二次调用
print(','.join(rdd.collect()))
```

```
spark,hadoop
```

常用持久化等级

- MEMORY_ONLY
- MEMORY_AND_DISK
- DISK_ONLY
- MEMORY_ONLY_SER 数据序列化
- MEMORY_AND_DISK_SER

  以上标签如果加_2则表示存两份

分区

分区

- RDD是弹性分布式数据集，通常RDD很大，会被分成很多个分区，分别保存在不同的节点上。
- RDD分区的一个分区原则是使得分区的个数尽量等于集群中的CPU核心（core）数目。
- 对于不同的Spark部署模式而言，都可以通过设置spark.default.parallelism这个参数的值，来配置默认的分区数目，一般而言：
  - 本地模式：默认为本地机器的CPU数目，若设置了local[N],则默认为N；
  - Apache Mesos：默认的分区数为8；
  - Standalone或YARN：在“集群中所有CPU核心数目总和”和“2”二者中取较大值作为默认值；