Pyspark累加器(accumulator)陷阱

原创 zlbingo 于 2021-01-14 20:37:23 发布 447 收藏 1 版权

累加器(accumulator)陷阱

> 【前置知识】： Spark 惰性求值运算机制，持久化的使用。

- 首先给出一个例子：

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setMaster('local[*]').setAppName('rookie')
sc = SparkContext(conf=conf)
acc = sc.accumulator(0)


def judge_even(row_data):
    """
    过滤奇数，计数偶数个数
    """
    global acc
    if row_data % 2 == 0:
        acc += 1
        return 1
    else:
        return 0


a_list = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
even_num = a_list.filter(judge_even)
print(f'the accumulator value is {acc}')
```

大家可以先猜一下输出的结果是什么

以下是输出的结果

> the accumulator value is 0

【分析】为什么会出现这个结果呢?

这是因为Spark中的一系列的转化（transform）算子操作会构成一长串的任务链，只有当存在行动（action）算子操作时，才会进行真正的运算。

累加器（accumulator）也同理。

上述代码中并没有action算子，因此累计器并没有进行累加。

那么接下来，验证我们上述的分析，增加一个action算子

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setMaster('local[*]').setAppName('rookie')
sc = SparkContext(conf=conf)
acc = sc.accumulator(0)
...
...
even_num = a_list.filter(judge_even)
# 增加一个action算子count操作
print(f'even_num.count {even_num.count()}')
print(f'the accumulator value is {acc}')
```

执行结果为:

> even_num.count 5
> the accumulator value is 5

我们可以看到，当我们增加action算子之后，累加器就会按照代码规则进行累加了

- 例子2

  接下来我们再看另一个例子

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setMaster('local[*]').setAppName('rookie')
sc = SparkContext(conf=conf)
acc = sc.accumulator(0)
...
...
a_list = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
even_num = a_list.filter(judge_even)
print(f'even_num.count {even_num.count()}')
print(f'even_num.collect {even_num.collect()}')
print(f'the accumulator value is {acc}')
```

  那么再让我们猜一下，累加器输出的结果是啥

  以下是代码的输出结果：

> even_num.count 5
> even_num.collect [2, 4, 6, 8, 10]
> the accumulator value is 10

  【分析】我们可以看到实际上经过过滤之后的偶数为5个，但是累加器给出的数值是10个，为两倍的关系，那么为什么会是这种结果呢?

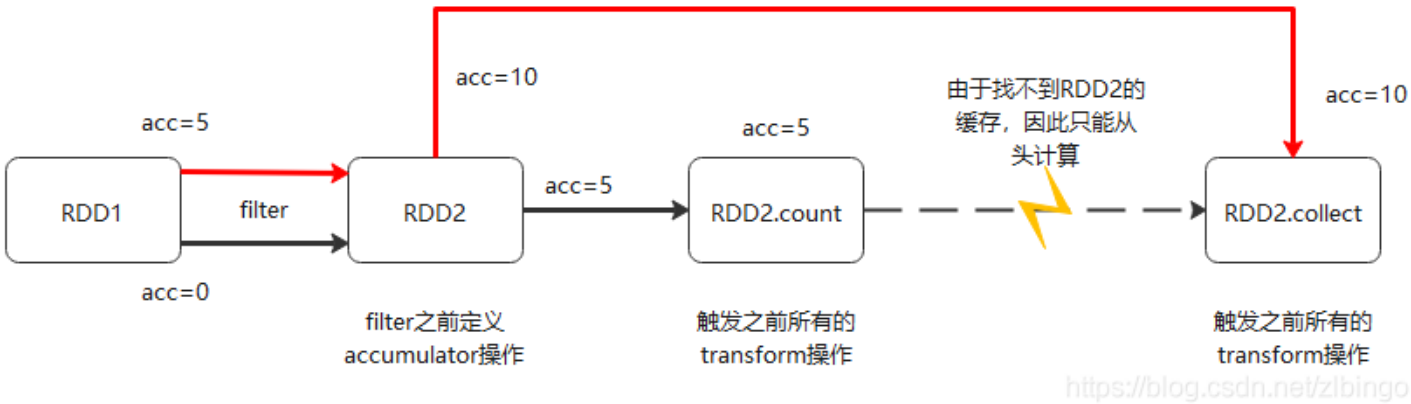
  这就涉及到Spark运行机制的问题了

  附官方描述：

> For accumulator updates performed inside actions only, Spark guarantees that each task’s update to the accumulator will only be applied once, i.e. restarted tasks will not update the value. In transformations, users should be aware of that each task’s update may be applied more than once if tasks or job stages are re-executed.

我们都知道Spark是惰性求值（Lazy Evaluation）机制，只有遇到action算子才会执行运算。

- 分析过程如图所示



- 当我们遇到第一个action算子count的时候，他就会从头开始计算，这是累计器就会累加到5，直到输出count的值。
- 当我们遇到第二个action算子collect时，由于前面没有缓存数据可以直接加载，因此也只能从头计算，在从头计算时，这时accumulator已经是5了，在计算过程中累计器同样会被再执行一次，因此最后会输出10

接下来验证我们的分析，运行如下代码

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setMaster('local[*]').setAppName('rookie')
sc = SparkContext(conf=conf)
acc = sc.accumulator(0)
...
...
a_list = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
even_num = a_list.filter(judge_even)
print(f'even_num.count {even_num.count()}')
```

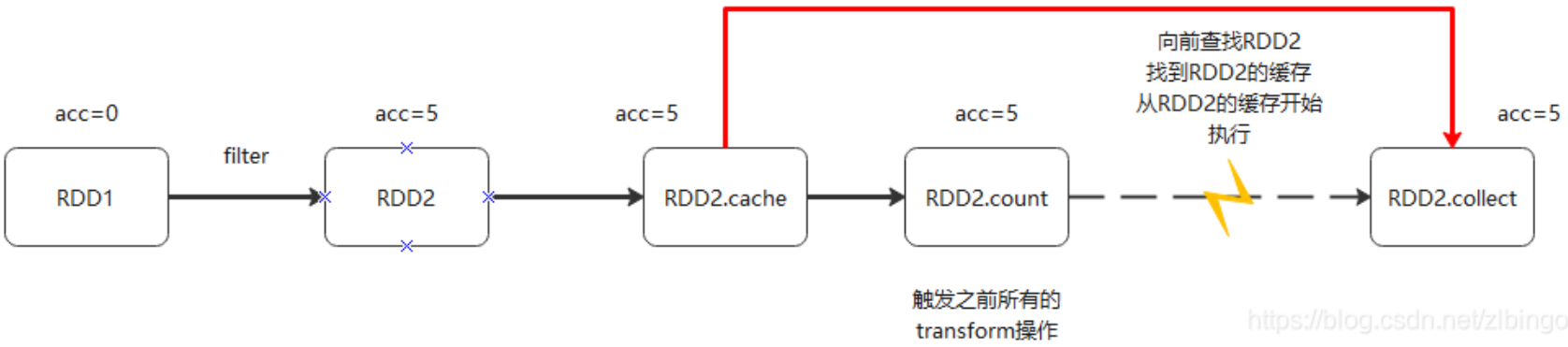
```
print(f'after the first action operator the accumulator is {acc}')
print(f'even_num.collect {even_num.collect()}')
print(f'after the second action operator the accumulator is {acc}')
```

```
even_num.count 5
after the first action operator the accumulator is 5
even_num.collect [2, 4, 6, 8, 10]
after the second action operator the accumulator is 10
```

以上结果刚好可以印证我们的分析。

- 遇到以上问题我们应该怎么解决这个问题呢

  解决这个问题只需要切断他们之间的依赖关系即可，即：在累加器计算之后进行持久化操作，这样的话，第二次action操作就会从缓存的数据开始计算，不会再重复进行累计器计数



执行如下代码：

```
from pyspark import SparkContext, SparkConf

```

```
conf = SparkConf().setMaster('local[*]').setAppName('rookie')
sc = SparkContext(conf=conf)
acc = sc.accumulator(0)
...
...
a_list = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
# 增加cache
even_num = a_list.filter(judge_even).cache()
print(f'even_num.count {even_num.count()}')
print(f'after the first action operator the accumulator is {acc}')
print(f'even_num.collect {even_num.collect()}')
print(f'after the second action operator the accumulator is {acc}')
```

> even_num.count 5
> after the first action operator the accumulator is 5
> even_num.collect [2, 4, 6, 8, 10]
> after the second action operator the accumulator is 5

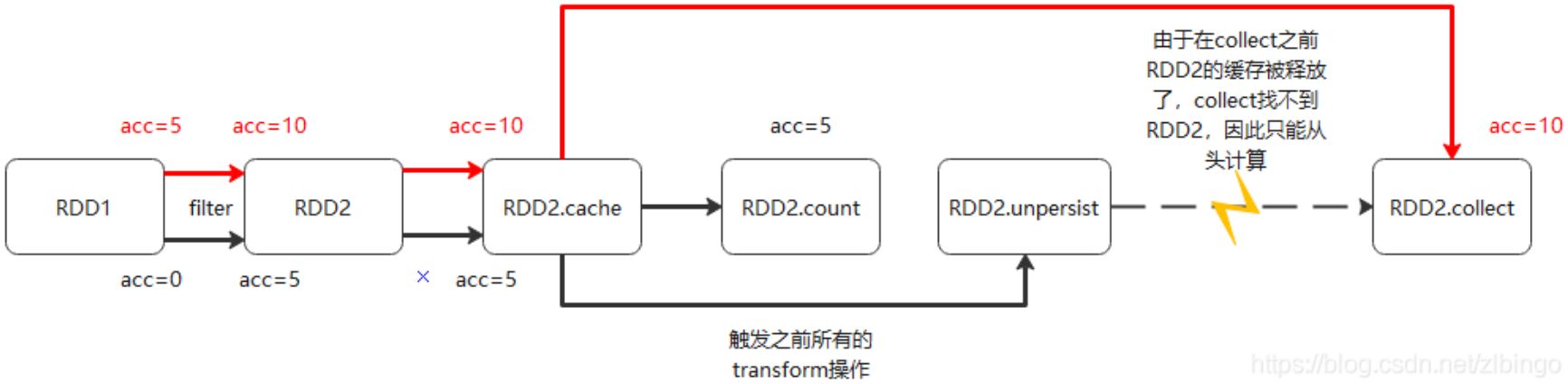
- 另外还有有一个缓存的小陷阱

  运行如下例子

```
...
a_list = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
even_num = a_list.filter(judge_even).cache()
print(f'even_num.count {even_num.count()}')
print(f'after the first action operator the accumulator is {acc}')
# 对缓存进行释放
even_num.unpersist()
print(f'even_num.collect {even_num.collect()}')
print(f'after the second action operator the accumulator is {acc}')
```

> even_num.count 5
> after the first action operator the accumulator is 5
> even_num.collect [2, 4, 6, 8, 10]
> after the second action operator the accumulator is 10

这时我们看到累加器又被多运行了一次。



这是因为第一次action算子操作后，存在一步释放缓存的操作，当执行第二个action算子时，首先会将rdd的缓存释放，然后再对rdd进行collect操作，而由于rdd没有被缓存，因此想要被collect必须从头计算，那么累加器又一次被重新计算，因此又变为两倍。

- 参考文章：
    1. Spark累加器(Accumulator)陷阱及解决办法
    2. spark 数据持久化与释放