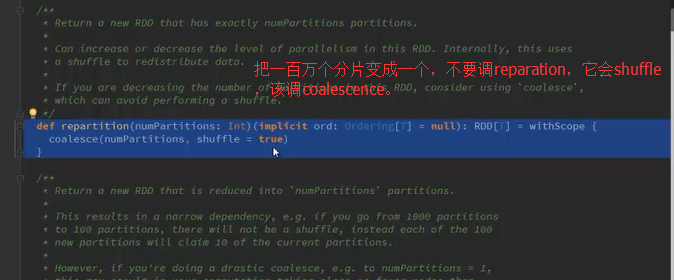
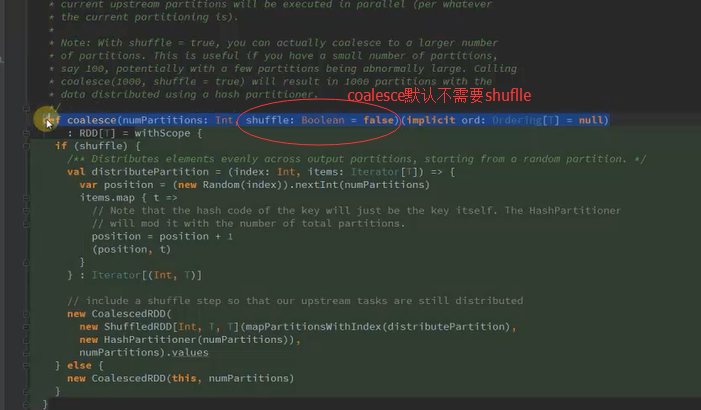
开篇：spark各种库，spark sql，spark machice learning，等这么多库底层都是封装的RDD。意味着1：RDD本身提供了通用的抽象，2：spark现在有5个子框架，sql，Streaming，流式处理，机器学习，图计算，spark R。 可以根据具体领域的内容建模，建第6个库，第7个库。必须掌握spark的这个通用抽象基石-RDD.

1:RDD，基于工作集的分布式函数编程的应用抽象。MapReduce是基于数据集的。他们的共同特征是位置，感知，容错和负载均衡是数据集和工作集都有的。基于数据集的处理：工作方式是从物理存储设备上，加载数据，然后操作数据，然后写入物理存储设备。但它不适应的场景有:1)不适合于大量的迭代。2）不适合于交互式查询，每次查询都要从磁盘上读取数据，然后查询，然后写回数据结果、如果复杂的查询有多个步骤，则要多次基于磁盘，这还是次要的，只是速度影响。重点是基于数据流的方式，不能复用曾经的结果或者中间的计算结果，这才是致命的，例如有几千人并发操作一个数据仓库，假如有一百人的查询完全是一样的，它也会重新加载数据，重新查询，而spark会对结果重用，复用中间计算结果，就是前面10个步骤是一样的，算过，数据集不会复用，spark则会复用。而RDD(Resillient Distributed Dataset)是基于工作集的，有前面讲过的4大弹性：弹性1：自动的进行内存和磁盘数据存储的切换；弹性2：基于Lineage的高校容错；弹性3：Task如果失败会自动进行特定次数的重试；弹性4：Stage如果失败会自动进行特定次数的重试，而且重试的时候只会计算失败的分片，弹性5：checkpoint和persist，链条比较长，计算比较笨重的时候，我们把数据都放在磁盘/HDFS上，这是checkpoint。而persist，是在内存中或者磁盘中对数据进行复用。这是效率和容错的延伸点。弹性6：数据调度弹性，DAG TASK和资源管理无关。弹性7：数据分片的高度弹性，如计算过程中会产生很多数据碎片，这时partition就特别小。每次都消耗一个线程去处理的话，这会降低处理效率。这时会考虑把很多partion合并成一个大的partition提升效率。另一个方面，内存不是那么多，但是partition数据比较大，数据block比较大。会考虑把它变成更小的分片，这样让spark有更多的处理批次，但是不会出现OOM。这样数据分片，我们可以人工提高并行度，降低并行度，是弹性的高度体现，而且它完全有数据本地性。



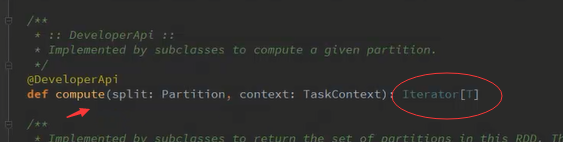


而从一万个分片变成10万个分片，则一般可能需要shuffle。RDD本身容许用户在执行多个查询时，显示的将工作集缓存在内存中。以后其它人来查询就可以重用工作集。自然极大提升查询速度。

提示：spark的位置感知比hadoop的位置感知好很多，hadoop进行partition的时候，就不管位置在哪里，spark进行partition的时候，进行下一步stage操作，是会确定这个位置的，它更精致化。

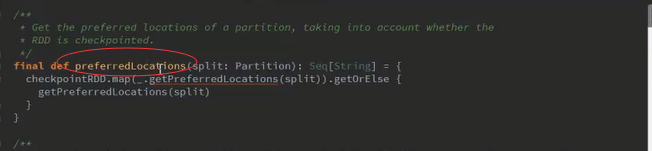
2：Spark Streaming为什么老是用checkpoint，因为经常要用到以前的东西。假设Spark如果有1000个RDD，一般不会产生1000个中间结果。假设Stage内部有一千个步骤，它中间不会产生999次中间结果，默认情况下，它只是产生一次中间结果，而hadoop会产生1000次中间结果。由于Spark的RDD它本身是只读分区的集合，但又为了应对它只对数据标记，不做计算的计算模型，所以它是lazy级别的，所以每次transformation构建的新的RDD，也都是以父RDD为自己的第一个参数传进去的，由此构成了一个链条，在计算的由最后action的时候再触发，所以只有一个中间结果，也所以这就构成了一个从后往前回溯的过程,就是一个函数展开的过程，从源码也看到它是这种从后往前的链条依赖关系，所以它容错的开销会非常低，为什么呢？因为常规的容错方式有1：数据检查点（它的工作方式要通过数据中心的网络连接不同的机器，每次操作的时候都要复制整个数据集。每次都有一个拷贝，是要通过网络的，因为要复制到其他机器上，而带宽就是分布式的瓶颈，这对存储资源也是非常大的消耗）以及2：记录数据的更新（每次数据变化了，我们都记录下，但这个第一复杂，第2耗性能，重算的时候比较难处理），既然这么多缺点?spark为什么在记录数据更新上就这么高效呢?1）RDD是不可变的所以每次操作就会变成新的RDD+lazy，不存在全局修改的问题，控制难度极大的下降。又产生了链条，可以很方便的容错。2：是粗粒度模式，最简单的想，RDD就是一个List或者Array。RDD是分布式函数式编程的抽象。基于RDD编程一般都采用高级函数。

3：Stage结束，数据会写磁盘。是粗粒度模式，是为了效率，为了简化。如果是更新力度太细太多，记录成本非常高，效率就不是那么高了。对RDD的具体的数据的改变操作(写操作)都是粗粒度的。RDD的写操作是粗粒度的(限制了它的使用场景，网络爬虫这件事就不适合Rdd去做)，但是RDD的读操作，既可以是粗粒度的也可以是细粒度的。Partition本身是一个很普通的数据结构，指向我们的具体的数据本身，即计算时知道数据在哪里。而且这系列数据分片的计算逻辑都是一样的。



4：compute为什么所有RDD操作返回都是迭代器？好处是让所有框架无缝集成，结果流处理，机器学习都可以互调，无论是机器学习操作sql，还是sql操作机器学习，还是流处理操作图计算，还是流处理操作sql，大家都是基于RDD，我才不管你是什么东西，只关心你是RDD。第2点，又有可以调用子类的具体东西，所以是不是流处理可以直接调用机器学习的具体功能去训练。因为有this.type(),所以可以通过运行时runtime，来具体把实际的实例赋值给RDD，你转过来就可以去操作它，这样使用了接口，还能调用接口下面的子类。

5：Scala中使用了接口，还能调用接口下面的子类。无缝集成的基础上，可以使用各自的功能。产生核裂变：如果我是做金融方面的，开发了一个金融类的子框架，子框架可以直接在代码中调机器学习，调图计算进行什么分享预测，行为分析，模式分析。也可以调sql进行数据挖掘。你写了个子框架，遵循RDD的规范，机器学习转过来可以调用我的金融框架。因为无缝集成，写个电商框架也可以调金融框架，每增强一个，可以让所有的增强。每提出一个新的框架，是不是可以使用其它所有的功能。



6：由于有了PreferedLocation,Spark可以处理一切数据，每次都符合完美的数据本地性。Spark就是要做一体化多元化的数据处理框架，不仅仅只是大数据。兼容一切文件系统，一切操作系统，一切文件格式。任何格式的数据，第一计算更快，第2使用更简单。但是Spark做实时事务性处理，反应没那么快，控制难度大。如银行转账。做实时处理是可以的。除此之外，Spark要一统数据处理的天下！

7：RDD的弊端：目前不支持细粒度的写操作(如网络爬虫)以及增量迭代计算(每次迭代的时候，只迭代其中的一部分数据，本身是粗粒度，不能很好的支持增量迭代)。