MapReduce数据序列化过程

MR在数据处理过程中，需要对数据进行序列化。Map端将数据写到数据流中，需要将基本数据类型转换成二进制数据。Reduce端，需要将二进制流反序列化。下面以WordCount为例，分析经典MR和NativeTask 二进制序列化和反序列化的过程。

# 1、经典MR

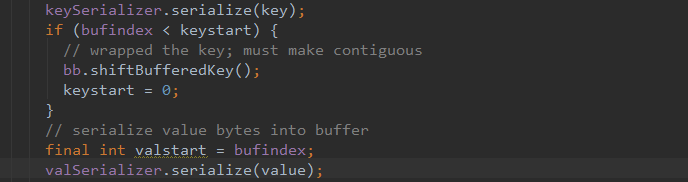
## 1.1 序列化

在Map执行阶段进行序列化，WordCount使用 new MR API，执行序列化之前的过程如下：

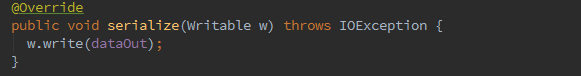
MapTask#run -> MapTask#runNewMapper ->NewOutputCollector#write->MapOutputBuffe

r#collect <K key,V value>

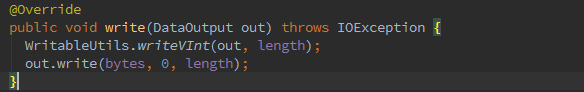
collect过程，首先将K/V序列化后，写入缓存，当缓存写满后spill到磁盘中，下面分析序列化格式，序列化源码：



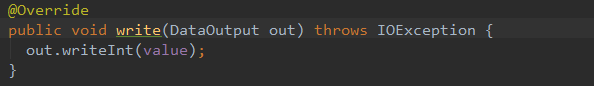
WordCount中Key:Text, Value:IntWritable，均使用序列化类WritableSerializer。执行源码如下：



Text.Write(dataOut) ：



IntWritable.Write(dataOut):



在缓存中的序列化格式如下：



写入顺序是先写Text长度，然后写Text二进制数据，然后再写IntWritable数据

## 1.2 反序列化

Map执行结束后，spill的文件经过merge后形成一个经过排序的文件，Reducer启动shuffle，从Map中拉取该Reduce的对应的数据，过程如下：

ReduceTask#run -->

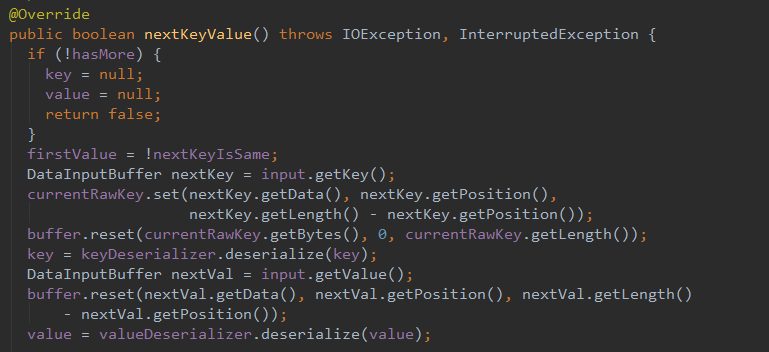
Shuffle#run: 返回RawKeyValueIterator -->

MergeManagerImpl#close : 输入源为Fetcher获取的数据，从map拉取数据的线程 -->

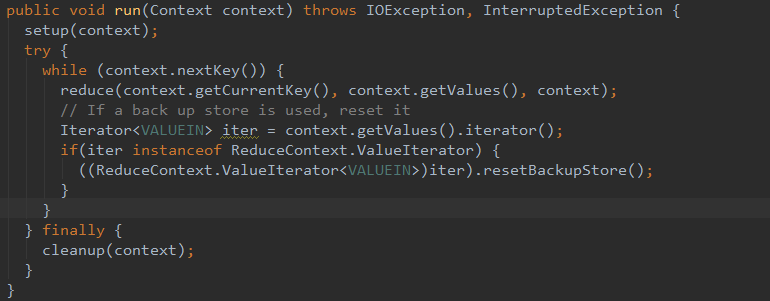
MergeQueue#merge: 返回的结果为MergeQueue，该类为Iterator，该类中的成员变量segements:List<Segement<K,V>>，从MapOutput 文件中读取数据，List每个对象都对应一个输出文件。-->

Segment#reader:Reader<K,V>，FSdataInputStream，K,V为基本参数 ,Reader从数据流中获取RawKey及RawValue，二进制值，然后经过Key.class及Value.class进行反序列化。-->

ReducerContext.ReduceContextImpl，ReduceContextImpl用于存储Iter数据，input：RawKeyValueIterator，由Shuffle获取。当前的key及value，从input中获取，并反序列化：--> 源码如下：



Reducer#run，从ReduceContextImp获取数据并处理，源码如下：



## 1.3 总结

MR的数据流有下面四个过程：

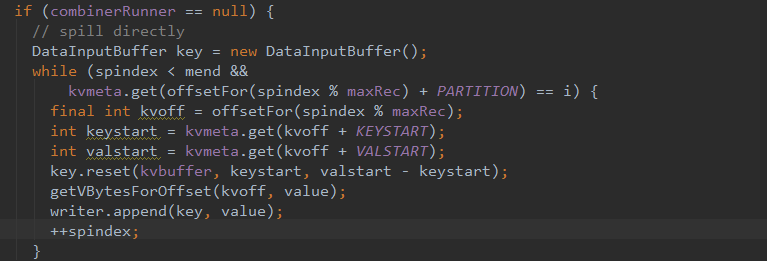
1. 序列化

在MapTask.MapOutputBuffer.collect中，调用keySerializer及valueSerializer完成序列化，然后写入缓存。格式如下：



1. 写入文件

缓存大小超过设定值时，调用MapOutputBuffer#sortAndSpill,flush及merge，生成MapOutput文件。文件的写入有IFile.Writer完成，核心代码如下：



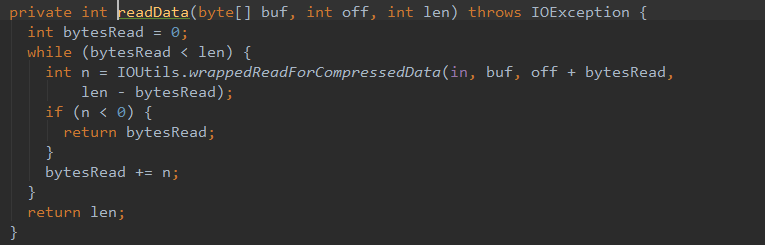
1. 读取文件

ReducerTask启动后，有LocalFetcher从MapOutputFile获取数据，上面已经分析过了。文件路径从配置文件中获取后传入Shuffle.Context。LocalFetcher初始化变量中需要有Context这个参数。Fetcher是个线程，在运行过程调用copyMapOutput，由MapOutput.shuffle等过程，将数据写入MergeManager中，这个过程涉及到硬盘及磁盘等（imMermoryMerger,onDiskMerger,memToMemMerger）。

根据文件生成finalSegments，createInMemorySegments及diskSegments等。

根据finalSegments，生成MergeQueue（RawKeyValueIterator）

生成key及value，由Reader.nextRawKey来获取，Reader有数据流InputStream。数据的获取readData：



读取的过程中，两个变量很重要，分别是currentKeyLength及currentValueLength。IFile.Reader#positionToNextRecord中进行赋值；

// Read key and value lengths

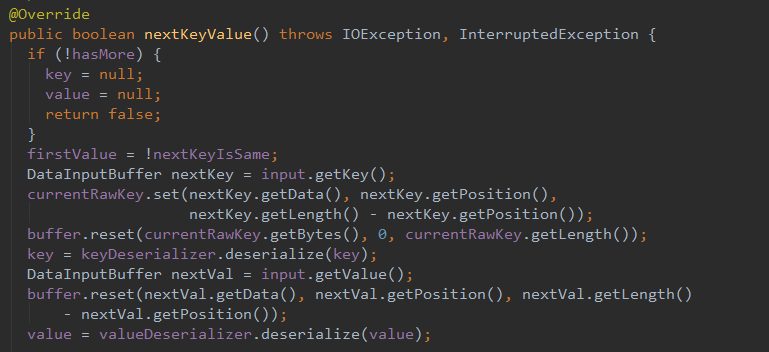
currentKeyLength = WritableUtils.readVInt(dIn);

currentValueLength = WritableUtils.readVInt(dIn);

1. 反序列化

在处理之前，将数据放入ReduceContextImpl，并反序列化。核心方法ReduceContext

-Impl#nextKeyValue



# 2、NativeTask MR

## 2.1 序列化

在Map执行阶段进行序列化，WordCount使用 new MR API，执行序列化之前的过程如下：

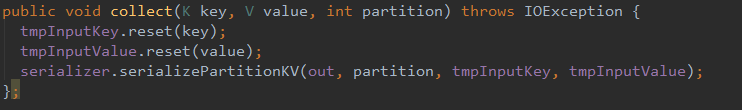
MapTask#run -> MapTask#runNewMapper ->NewOutputCollector#write->

NativeMapOutputCollectorDelegator #collect <K key,V value> -->

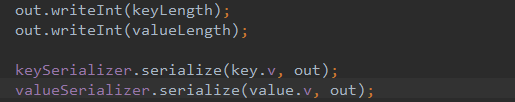
NativeCollectorOnlyHandle#collect -->

BufferPusher#collect

collect通过调用KVSerializer.serializePartitionKV(out,partition,key,value)来完成序列化过程：



KVSerializer中两个成员变量：keySerializer/valueSerializer，分别对Key/Value进行序列化。与经典MR相比，多了一个updateLength的方法，用于在序列化之前获取Key/Value的长度，序列化的过程如下：



输出的格式如下：



数据写入缓存ByteBufferDataWriter中，缓存超过设定大小后，通过NativeBatchProcessor

#sendData中，NativeTask的优化主要在这个过程。

## 2.2 反序列化

与经典MR相同。

## 2.3 总结

NativeTask MR的序列化和反序列化过程与经典MR最大的不同之处在于实现了每个类型的Serializer。

# 3.程序track

由于HiveKey的序列化与反序列化出现问题，因此在先跟踪下WordCount的序列化过程，包括以下几个地方：

1）序列化

2）反序列化

3）使用的序列化及反序列化类