一：CacheManager分析：

1. CacheManager：管理的是缓存，而缓存可以是基于内存的缓存，也可以是基于磁盘的缓存；
2. CacheManager需要通过BlockManager来操作数据；
3. 当Task运行的时候会调用RDD的compute方法进行计算，而compute方法会调用iterator方法；

*/\*\*  
 \** *Internal method to this RDD; will read from cache if applicable, or otherwise compute it.  
 \* This should* ''*not*'' *be called by users directly, but is available for implementors of custom  
 \* subclasses of RDD.  
 \*/***final def** iterator(split: Partition, context: TaskContext): Iterator[T] = {  
 **if** (*storageLevel* != StorageLevel.*NONE*) {  
 SparkEnv.*get*.cacheManager.getOrCompute(**this**, split, context, *storageLevel*)  
 } **else** {  
 computeOrReadCheckpoint(split, context)  
 }  
}

二：CacheManager源码分析

1. Cache在工作的时候会最大化的保留数据，但是数据不一定绝对完整，因为当前的计算如果需要内存空间的话，那么cache在内存中的数据必须让出空间。此时如果在RDD持久化的时候同时指定了可以把数据放在Disk上，那么部分cache的数据就可以从内存转入到磁盘。否则的话，数据就会丢失！！！

*/\*\* Gets or computes an RDD partition. Used by RDD.iterator() when an RDD is cached. \*/***def** getOrCompute[T](  
 rdd: RDD[T],  
 partition: Partition,  
 context: TaskContext,  
 storageLevel: StorageLevel): Iterator[T] = {  
//根据partition的index索引，通过BlockManager获取曾经缓存的Key  
 **val** key = *RDDBlockId*(rdd.*id*, partition.index)

**case** None =>  
 // Acquire a lock for loading this partition  
 // If another thread already holds the lock, wait for it to finish return its results

// acquireLockForPartition:因为Spark有慢任务推测，对一个partition  
 **val** storedValues = acquireLockForPartition[T](key)  
 **if** (storedValues.isDefined) {  
 **return new** InterruptibleIterator[T](context, storedValues.get)  
 }

1. Cache的数据是通过BlockManager来管理的。具体cacheManager在获得缓存数据的时候，会通过BlockManager来抓住数据。

blockManager会先从本地获得数据，如果没有的话，则会getRomote从远程获取。

*/\*\*  
 \* Get a block from the block manager (either local or remote).  
 \*/***def** get(blockId: BlockId): Option[BlockResult] = {//option关键字：

*//一种是some，一种是none，在获取的时候可能会出现数据丢失，这时候none*  
 **val** local = getLocal(blockId)  
 **if** (local.isDefined) { //优先从本地获取数据  
 logInfo(s"Found block **$**blockId locally")  
 **return** local  
 }

*//如果本地没有则从远程获取数据。*  
 **val** remote = getRemote(blockId)  
 **if** (remote.isDefined) {  
 logInfo(s"Found block **$**blockId remotely")  
 **return** remote  
 }  
 None  
}

1. 如果CacheManager没有通过BlockManager获得缓存内容的话，此时会通过RDD的如下方法来获得数据：

**val** computedValues = rdd.computeOrReadCheckpoint(partition, context)

1. computeOrReadCheckpoint不会直接计算RDD，先检查checkpoint中有没有RDD partition分区，如果没有的话，再计算。

*/\*\*  
 \* Compute an RDD partition or read it from a checkpoint if the RDD is checkpointing.  
 \*/***private**[spark] **def** computeOrReadCheckpoint(split: Partition, context: TaskContext): Iterator[T] =  
{  
 **if** (isCheckpointedAndMaterialized) {  
 firstParent[T].iterator(split, context)  
 } **else** {  
 compute(split, context)  
 }  
}

1. 上述方法首先会查看当前的RDD是否进行了checkpoint，如果进行了话，就直接读取Checkpoint的数据，否则的话，就必须进行计算。计算之后，通过putInBlockManager会把数据按照storageLevel缓存。

**val** cachedValues = putInBlockManager(key, computedValues, storageLevel, updatedBlocks)

1. 计算之后通过putInBlockManager会把数据安装StorageLevel重新缓存的时候如果需要放在内存中，

blockManager.*memoryStore*.unrollSafely(key, values, updatedBlocks) **match** {

1. unrollSafely循环遍历，看到一点就放进内存。

**while** (values.hasNext && keepUnrolling) {