性能优化：两大块

spark的瓶颈，除了内存外，首要的是“网络带宽”

所以优化，是如下二者：

1. 内存使用（ memory tuning）
2. 数据序列化（ data serialization）
3. 中间结果压缩

data serialization：

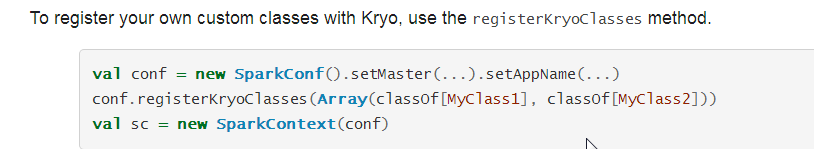
默认是java的序列化，一般是原数据的N倍，主要消耗在存储Object header和pointer上

另一种：[Kryo serialization](https://github.com/EsotericSoftware/kryo) ，能大大减少对象大小。

使用——在sparkConf中设置。

conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")

对于自定义类，还需要register一下：



**Memory Managemen**

内存分为2类：

执行内存、存储内存



存储内存是总内存的“子区域”：

执行内存，能“占用”掉存储内存，扩大自己的内存，反之不行。（简言之，执行内存优先度高）

执行内存+存储内存（R）=spark获取的总内存（M） （JVM还会分配内存给他自己）

spark.memory.fraction ——设置M占JVM的比例（默认0.6），设置过大容易OOM

spark.memory.storageFraction ——设置R占M的比例（默认0.5）

spark.memory.fraction 应该设置下，准则：适合JVM的永久代，可以优化GC

内存消耗情况：“Storage” page可以看到。

To estimate the memory consumption of a particular object, use SizeEstimator’s estimate method。

使用SizeEstimator.estimate查看具体某个对象的消耗情况。

优化——数据结构：

1. 减少java的基于指针的对象，和包装类对象：



1. 使用Array[Objects] ，多使用“数组”和“原生类”————少用引用类,可以使用java8的fastutil库：

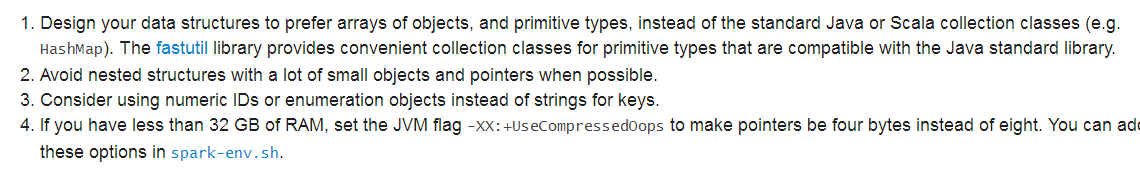
url: <http://fastutil.di.unimi.it/>

b．少使用“嵌套”结构， 即容器里有太多的“小对象”和“指针”

c . 使用“数值型”的IDS（索引） ，或枚举类， 而不用String类型（String类型的消耗特别大）

d. 如果机器RAM（内存）<32g，在spark-env.sh加入以下JVM配置：

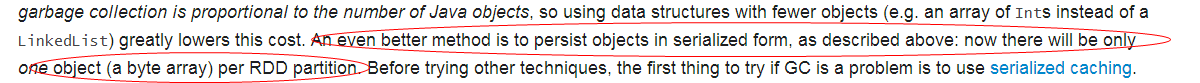
-XX:+UseCompressedOops



## Serialized RDD Storage

利用rdd的存储机制：选用内存存储，则RDD会被缓存（将整个RDD以Array[byte]形式存储）

当rdd的存储策略storage是以“serialized rdd”方式时（即存储于内存中时），整个spark程序，只有1个Object，就是Array[Byte]。所以，还是很有必要的（当然有可能OOM）



而且这样也不浪费内存。

如果没有很多对象，就不涉及到“对象回收”，也就不会频繁GC了。

关于数组，见：

<http://www.jb51.net/article/40877.htm>

1. 数组可以存放java的基本数据类型，所以他效率高，也没有“指针”等其他信息。

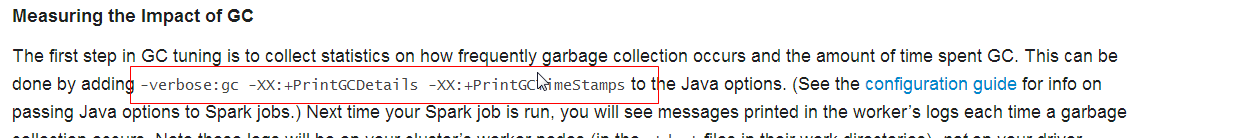
注意：数组的“浅拷贝”————若Array里放的是基本类型，则拷贝的是实际值（真的拷贝了）， 如果是放的“引用类型”，则只是拷贝了“对象的名称”（也即引用），而不是对象的“实际值或实际数据”。

“浅拷贝“： Integer[] b = a (假设a已经是一个Integer数组),这样b只是引用了a的名称，修改b时，实际上是修改的a。 你下次调用a的时候，已经不是原来的a了。

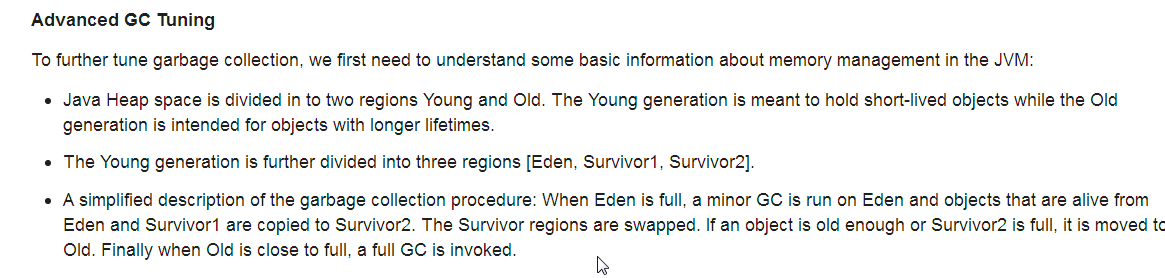
1. 集合类型，主要便利在于可扩展，但主要作用是：基于对“数组”的复制，用新数组替换而已。

要知道GC时间，可以添加配置项：

-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps

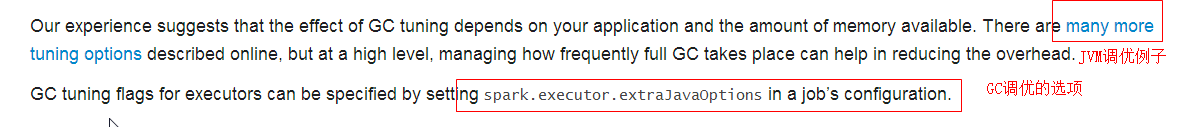


GC工作机制：



JVM有Young/Old两个区，Young有：eden ,suvivor1,suvivor2区， 对象随着前一个区满，会移动到后一个区。 Young区满了，会被移动到old区， old区满了，会发生full gc，这个非常费时间。 JVM此时完全开始GC.

基于GC的内存优化——————就是“避免发生full GC”.具体：小对象存在Young中，大的RDD存在Old中，并避免full GC



网上调优示例：

<http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html>

GC调优，spark给的建议：

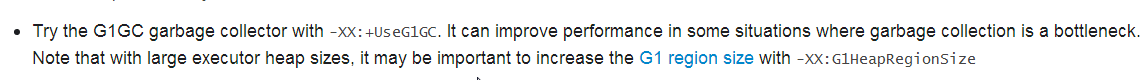
1. 查看GC统计指标，如果full GC发生，表明“内存不足”
2. 如果有很多minor collections(小gc，边执行程序边回收)，而没有太多major GC的话。要分配更多Memory给Eden区。——如果Eden区的大小为E，则要设置Young区的选项为4/3E .
3. 如果Old代满了，则减小spark.memory.fraction（也即留给非spark的内存调高一点）

因为：（少缓存对象 要优于 减慢task执行速度）

做法：减小Young Generation，或者调大NewRatios ————这是在JVM 选项调整的。

spark.executor.extraJavaOptions

如果GC是瓶颈，用G1GC方式，而不用串行或并行



其他：

1. 并行度：

map任务通过sc.textfile(data,partition)控制，默认是根据文件大小（和hadoop的切片一样）。

reduce任务：根据最大的pairRDD的分区数确定。

通过spark.default.parallelism 控制

默认设置为core总数的2-3倍

2

简言之：

spark性能调优：

内存管理（尽量避免full GC, 调整Executor执行和存储的比例）

网络IO（中间结果压缩，使用kryo序列化，而非java序列化、使用原生数组、数据类型，避免字符串、包装类，容器）