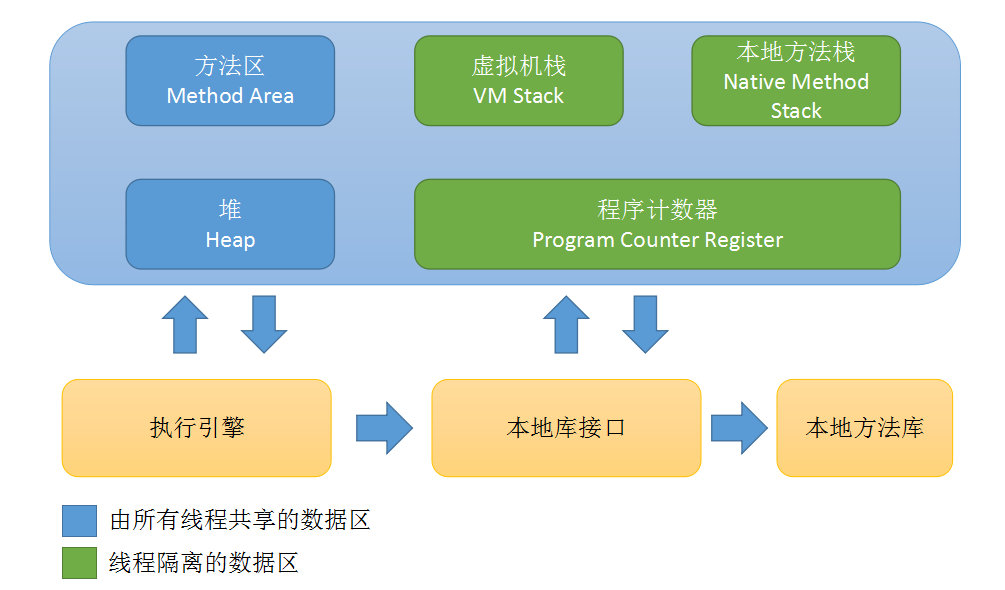
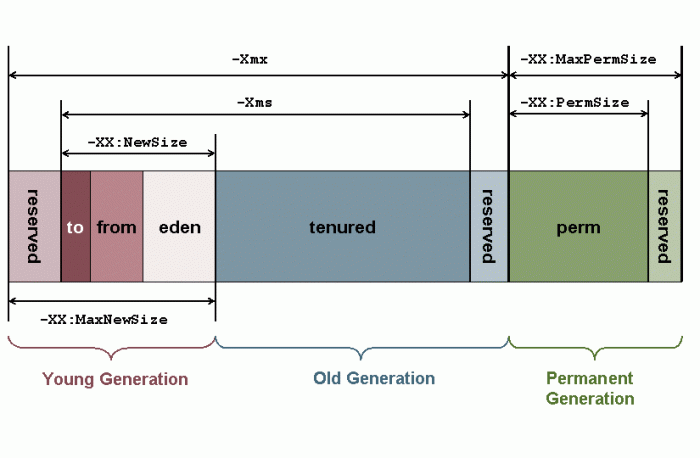
JDK 堆内存与堆外内存

JVM是Java平台的重要组成部分，其运行是数据区如下所示：



在执行JVM程序的过程中会把其管理的内存划分为若干不同的数据区域，其中堆是jvm中最大的一块内存区域，所有线程共享该块内存区域，在虚拟机启动时创建，用于存储对象实例和数组，该块内存称为堆内内存（On-heap memory）。该区域是JVM GC的主要区域，根据GC的过程将堆进行划分如下几个模块：



虚拟机定期垃圾内存进行回收，当Old Generation使用率达到一定程度会进行Full GC，该过程对所有分配对堆内存进行完整的扫描，会影响java程序的性能。

一种解决方案是使用堆外内存（off-heap memory），其含义是将内存对象分配在java虚拟机堆以外的内存，这些内存由操作系统管理（而不是虚拟机），从而使jvm堆保持较小的空间，可以将生命周期较长的对象保持到堆外内存中，减少gc对应用的影响，同时使大内存具有良好的伸缩性，在进程间可以共享数据减少虚拟机间的复制。

# **1.ON/Off Heap**

# 2.Spark内存管理

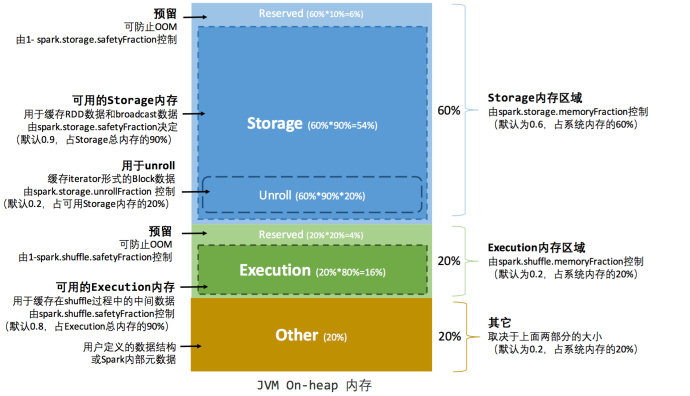
Spark是基于内存的分布式计算引擎，其内存管理模块是整个系统的核心。内存管理是建立在JVM内存管理之上，包括堆内内存和堆外内存两个区域：

1. 堆内内存

堆内内存的大小，由spark应用程序启动时-executor-memory或者spark.executor.memory参数配置。Executor内运行的并发任务共享JVM堆内存，可以分为以下部分：

* Execution内存，主要用于存放Shuffle、Join、Sort、Aggregation等计算过程中的临时数据
* Storage内存，主要用于存储spark的cache数据，例如RDD的缓存、unroll数据
* 用户内存，主要用于存储RDD转化操作所需要的数据，包括rdd依赖等信息
* 预留内存，系统预留内存，存储spark内部对象

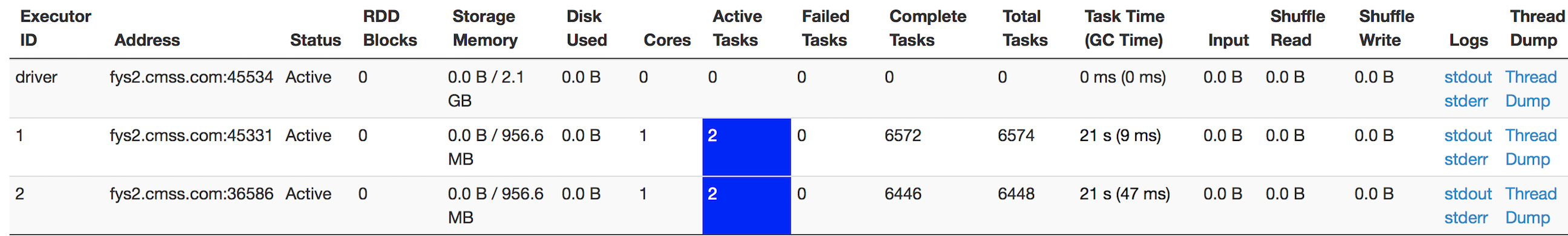
整个executor端堆内内存，概括如下：



下图设置spark提交作业的内存配置：

*--executor-memory 2g*

未设置spark.memory.fraction和storageFraction参数，Spark UI关于Storage Memory的显示如下：



executor内存大小为956m，计算公式为：

*totalOnHeapStorageMemory=*

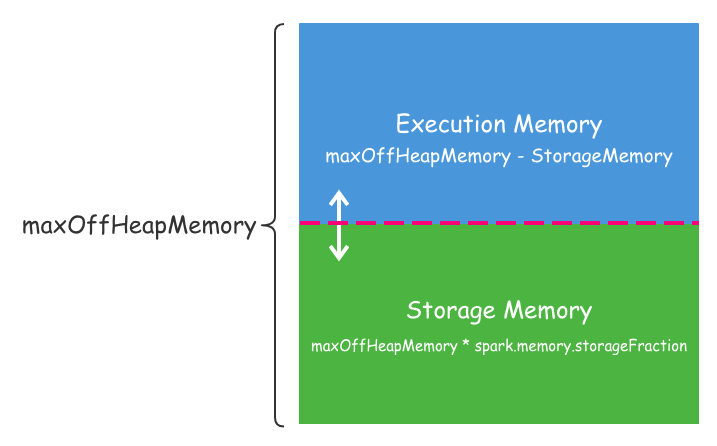
*(systemMemory-reservedMemory)\*memory.fraction\*storageFraction*

1. 堆外内存

Spark 1.6开始引入off-heap内存，在这种模式下，不在jvm内申请内存，而是调用java unsafe相关的API直接向操作系统申请内存，这部分内存不由JVM内存管理，可以避免频繁的GC。默认情况下，堆外内存是关闭的，通过参数：

spark.memory.offHeap.enabled

进行启用，并且通过spark.memory.offHeap.size设置堆外内存大小。如果启用堆外内存，那么executor内将同时存在堆内和堆外内存，其可用内存为两部分内存之和。与堆内内存相比，堆外内存只区分Execution内存和Storage内存，其分布如下图所示：

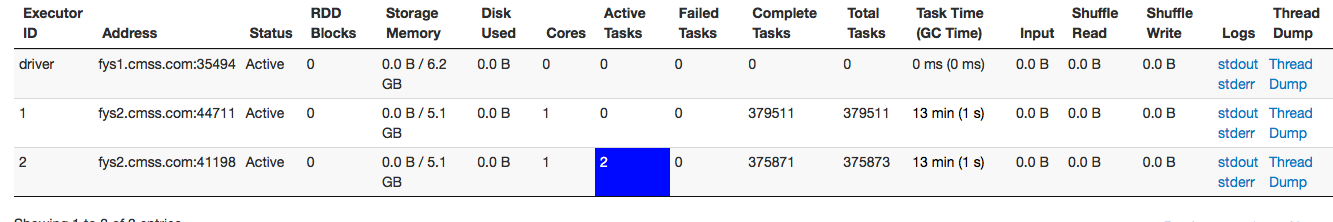


设置堆外内存，参数如下：

*spark.memory.offheap.enabled true*

*spark.memory.offHeap.size 4096000000*

启动后，内存显示如下图：

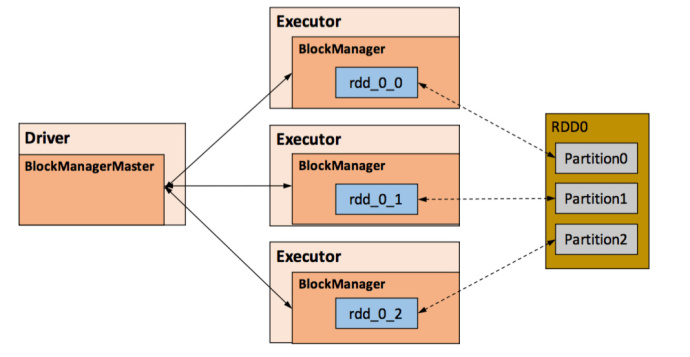


其计算公式为：

*StorageMemory = totalOnHeapStorageMemory+ spark.memory.offHeap.size*

3) 堆外内存的使用分析

RDD是Spark最基本的数据抽象，基于稳定物理存储中数据集上创建。其持久化由Spark Storage模块负责，其负责计算过程中生成的数据，将在内存或磁盘、本地或者远程数据的功能封装起来。其模块示意图如下：



Driver与Executor构成主从架构，BlockManager以Block为单位进行数据的管理，Driver负责整个应用程序的block元数据信息的管理和维护，而slave需要将block的更新等状态上报到Master，同时接收Master命令新增或者删除RDD。

在对RDD持久化时，Spark规定了MEMORY\_ONLY、MEMORY\_AND\_DISK等7种不同的级别，如下所示：

*class StorageLevel private(*

*private var \_useDisk: Boolean,*

*private var \_useMemory: Boolean,*

*private var \_useOffHeap: Boolean,*

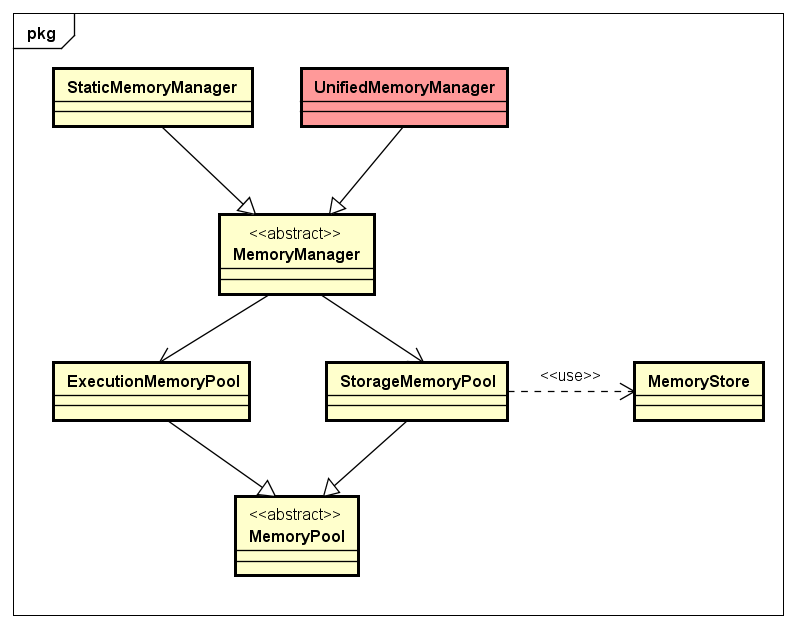
*private var \_deserialized: Boolean,*

*private var \_replication: Int = 1)*

从数据结构分析，可以将存储级别从三个维度定义了RDD的partition的存储方式：

* 存储位置，磁盘/堆内内存/堆外内存，MEMORY\_AND\_DISK是同时在磁盘和堆内内存上存储，实现冗余备份。OFF\_HEAP则只在堆外内存存储，选择堆外存储时不同存储到其他位置
* 存储形式，是否为序列化的形式，MEMORY\_ONLY是非序列化方式存储，OFF\_HEAP是序列化方式存储
* 副本数量，大于1则需要远程冗余备份到其他节点，如DISK\_ONLY\_2需要远程备份1个副本

Spark的内存分配与管理与MemoryManager来执行，其依赖的组件类及其关系如下图所示：



Spark内存大体分为两类：Execution和Storage内存，分别对应ExecutionMemoryPool和StorageMemoryPool，其实现了动态内存池的功能。内存的管理类为MemoryAllocator，对应堆内内存和堆外内存两种方式，如下所示：

*private[memory] final val tungstenMemoryAllocator: MemoryAllocator = {*

*tungstenMemoryMode match {*

*case MemoryMode.ON\_HEAP => MemoryAllocator.HEAP //HeapMemoryAllocator*

*case MemoryMode.OFF\_HEAP => MemoryAllocator.UNSAFE //UnsafeMemoryAllocator*

*}}*

MemoryAllocator提供allocate与free两个成员函数来提供内存的分配与释放，分配的内存以MemoryBlock来表示，下面是allocate的核心代码：

* HeapMemoryAllocator

*long[] array = new long[numWords];*

*MemoryBlock memory = new MemoryBlock(array, Platform.LONG\_ARRAY\_OFFSET, size);*

*if (MemoryAllocator.MEMORY\_DEBUG\_FILL\_ENABLED) {*

*memory.fill(MemoryAllocator.MEMORY\_DEBUG\_FILL\_CLEAN\_VALUE);*

*}*

* UnsafeMemoryAllocator

*long address = Platform.allocateMemory(size);*

*MemoryBlock memory = new MemoryBlock(null, address, size);*

*if (MemoryAllocator.MEMORY\_DEBUG\_FILL\_ENABLED) {*

*memory.fill(MemoryAllocator.MEMORY\_DEBUG\_FILL\_CLEAN\_VALUE);}*

其中Platform.allocateMemory使用sun.misc.Unsafe来申请对外内存，如下：

*\_UNSAFE.allocateMemory(size);*

http://ju.outofmemory.cn/entry/300202

**https://blog.csdn.net/allwefantasy/article/details/50447687**

https://blog.csdn.net/hrbeuwhw/article/details/79520247

<https://www.iteblog.com/archives/2342.html>