# [CMS收集器](https://www.cnblogs.com/chenpt/p/9803298.html)

一种以获取最短回收停顿时间为目标的收集器。

**特点**：基于标记-清除算法实现。并发收集、低停顿。

**应用场景**：适用于注重服务的响应速度，希望系统停顿时间最短，给用户带来更好的体验等场景下。如web程序、b/s服务。

**CMS收集器的运行过程分为下列4步：**

**初始标记**：标记GC Roots能直接到的对象。速度很快但是仍存在Stop The World问题。

**并发标记**：进行GC Roots Tracing 的过程，找出存活对象且用户线程可并发执行。

**重新标记**：为了修正并发标记期间因用户程序继续运行而导致标记产生变动的那一部分对象的标记记录。仍然存在Stop The World问题。

**并发清除**：对标记的对象进行清除回收。

 CMS收集器的内存回收过程是与用户线程一起并发执行的。

 CMS收集器的工作过程图：



CMS收集器的缺点：

* 对CPU资源非常敏感。
* 无法处理浮动垃圾，可能出现Concurrent Model Failure失败而导致另一次Full GC的产生。
* 因为采用标记-清除算法所以会存在空间碎片的问题，导致大对象无法分配空间，不得不提前触发一次Full GC。

# Java内存溢出

## 引起内存溢出的原因有很多种，常见的有以下几种：

● 内存中加载的数据量过于庞大，如一次从数据库取出过多数据；

● 集合类中有对对象的引用，使用完后未清空，使得JVM不能回收；

● 代码中存在死循环或循环产生过多重复的对象实体；

● 使用的第三方软件中的BUG；

● 启动参数内存值设定的过小；

## 内存溢出的解决方案：

● 第一步，修改JVM启动参数，直接增加内存。(-Xms，-Xmx参数一定不要忘记加。)

● 第二步，检查错误日志，查看“OutOfMemory”错误前是否有其它异常或错误。

● 第三步，对代码进行走查和分析，找出可能发生内存溢出的位置。重点排查以下几点：

1.检查对数据库查询中，是否有一次获得全部数据的查询。一般来说，如果一次取十万条记录到内存，就可能引起内存溢出。这个问题比较隐蔽，在上线前，数据库中数据较少，不容易出问题，上线后，数据库中数据多了，一次查询就有可能引起内存溢出。因此对于数据库查询尽量采用分页的方式查询。

2.检查代码中是否有死循环或递归调用。

3.检查是否有大循环重复产生新对象实体。

4.检查对数据库查询中，是否有一次获得全部数据的查询。一般来说，如果一次取十万条记录到内存，就可能引起内存溢出。这个问题比较隐蔽，在上线前，数据库中数据较少，不容易出问题，上线后，数据库中数据多了，一次查询就有可能引起内存溢出。因此对于数据库查询尽量采用分页的方式查询。

5.检查List、MAP等集合对象是否有使用完后，未清除的问题。List、MAP等集合对象会始终存有对对象的引用，使得这些对象不能被GC回收。

● 第四步，使用内存查看工具动态查看内存使用情况。