JVM的主要组成部分

- 类加载子系统
- 执行引擎
- 垃圾回收子系统
- 运行时数据区

JVM运行时数据区

- pc寄存器
- 方法区
- 堆
- 虚拟机栈
- 本地方法栈

JVM内存为什么要分成新生代，老年代，持久代。新生代中为什么要分为Eden和Survivor。

为什么分代：提高GC性能。如果不分代，GC时就需要扫描整个堆区，性能低。

1）强引用

我们平时new了一个对象就是强引用，例如 Object obj = new Object();即使在内存不足的情况下，JVM宁愿抛出OutOfMemory错误也不会回收这种对象。

2）软引用

如果一个对象只具有软引用，则内存空间足够，垃圾回收器就不会回收它；如果内存空间不足了，就会回收这些对象的内存。

```
SoftReference<String> softRef=new SoftReference<String>(str);     // 软引用
```

复制代码

用处： 软引用在实际中有重要的应用，例如浏览器的后退按钮。按后退时，这个后退时显示的网页内容是重新进行请求还是从缓存中取出呢？这就要看具体的实现策略了。

（1）如果一个网页在浏览结束时就进行内容的回收，则按后退查看前面浏览过的页面时，需要重新构建

（2）如果将浏览过的网页存储到内存中会造成内存的大量浪费，甚至会造成内存溢出

如下代码：

```
Browser prev = new Browser();                 // 获取页面进行浏览
SoftReference sr = new SoftReference(prev); // 浏览完毕后置为软引用
if(sr.get()!=null){
    rev = (Browser) sr.get();           // 还没有被回收器回收，直接获取
}else{
    prev = new Browser();               // 由于内存吃紧，所以对软引用的对象回收了
    sr = new SoftReference(prev);       // 重新构建
}
```

复制代码

3）弱引用

具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。

```
String str=new String("abc");
WeakReference<String> abcWeakRef = new WeakReference<String>(str);
str=null;
```

等价于

```
str = null;
System.gc();
```

复制代码

4）虚引用

如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。虚引用主要用来跟踪对象被垃圾回收器回收的活动。

为什么G1不关心Young gen的对象引用关系的更新？

分代式G1模式下有两种选定CSet的子模式，分别对应young GC与mixed GC：

- Young GC：选定所有young gen里的region。通过控制young gen的region个数来控制young GC的开销。
- Mixed GC：选定所有young gen里的region，外加根据global concurrent marking统计得出收集收益高的若干old gen region。在用户指定的开销目标范围内尽可能选择收益高的old gen region。

可以看到young gen region总是在CSet内。而G1 gc时关心的是Cset外对Cset内的引用因此分代式G1不维护从young gen region出发的引用涉及的RSet更新。而CMS是只回收old gen的所以CMS的“CSet”是old gen当然要关心young gen的引用变更。YGC过程中有对RS的更新步骤。

Cset内对象变化较大，没必要浪费空间记录