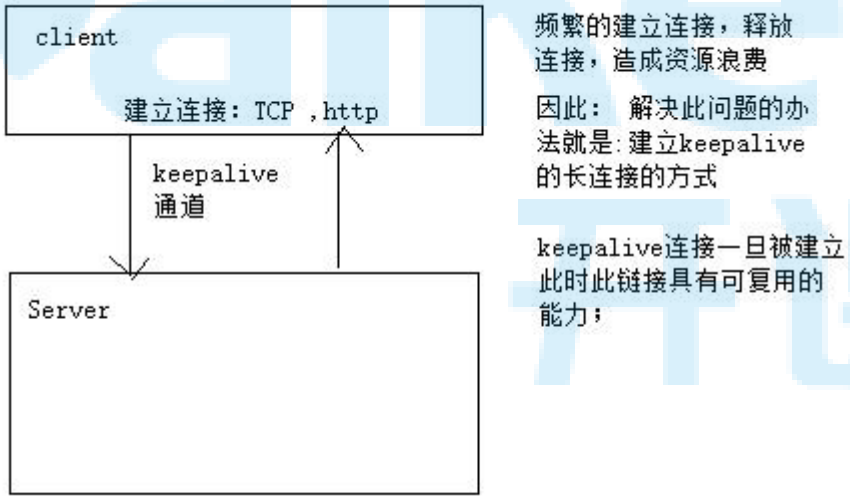
Jvm 调优原理&JVM 调优实践&GC 日志分析

今日课程主要内容：

1、keepalive 长连接
2、undertow 服务器
3、探索 JVM 调优原理（调优算法，分代模型，调优原因）
4、JVM 调优实践
5、GC 日志分析
6、根据日志分析再进行调优

1 Keepalive 长连接

1.1 什么是长连接

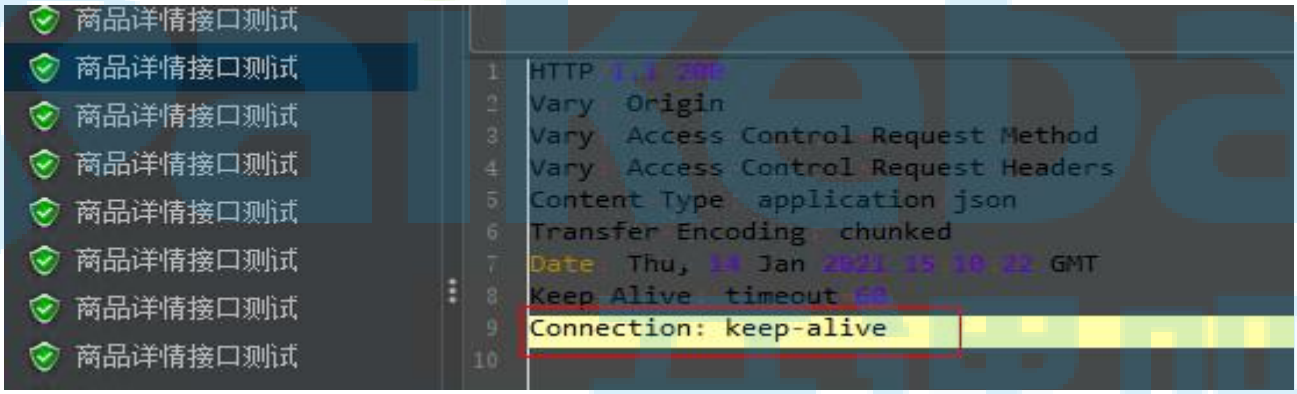


所谓的长连接就是保持一个连接长时间不释放，让其他请求线程可以进行复用；达到资源复用的情况；

1.2 Jmeter 长连接

Jmeter 连接服务端进行压力测试的时候，使用就是 keepavlie 长连接，在高并发模式下，保证连接具有复用性；

可以看见 connection: keep-alive 长连接



问题： keepalive 连接越多越好？？

答案： keepalive 连接本身消耗大量资源；如果不能及时释放，系统 TPS 上不去，因此需要设置合理的 keepalive 连接数，以及连接的超时时间；

```
/**
 * @ClassName WebServerConfig
 * @Description 对web 服务器进行改造，改写keepalive 相关配置
 * @Author hubin
 * @Date 2021/1/16 20:17
 * @Version V1.0
 **/
@Component
public class WebServerConfig implements
```

```
WebServerFactoryCustomizer<ConfigurableWebServerFactory>{

    // 定制tomcat 连接器，设置相关keepalive
    @Override
    public void customize(ConfigurableWebServerFactory
configurableWebServerFactory) {
        // 连接器

((TomcatServletWebServerFactory)configurableWebServerFactory).
addConnectorCustomizers(new TomcatConnectorCustomizer() {
            @Override
            public void customize(Connector connector) {
                // 获取httpprotocol
               Http11NioProtocol protocolHandler =
(Http11NioProtocol) connector.getProtocolHandler();

                // 如果30s 没有请求服务的话，自动释放连接
                protocolHandler.setKeepAliveTimeout(30000);
                // 允许开启的最大连接数
                protocolHandler.setMaxKeepAliveRequests(10000);

            }
        });

    }
}
```

2 Undertow

Undertow 是一个轻量级 servlet 服务器，注意：tomcat 也是一个 servlet 服务器； undertow 比 tomcat 更加轻量级，undertow 非常专一，没有任何可视化的组件，专注于 Servlet 容器领域；因此 undertow 性能略好于 tomcat 服务器；

1）引入 undertow 服务器

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-tomcat</artifactId>
            </exclusion>
        </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-undertow</artifactId>
</dependency>
```

2）修改性能参数配置

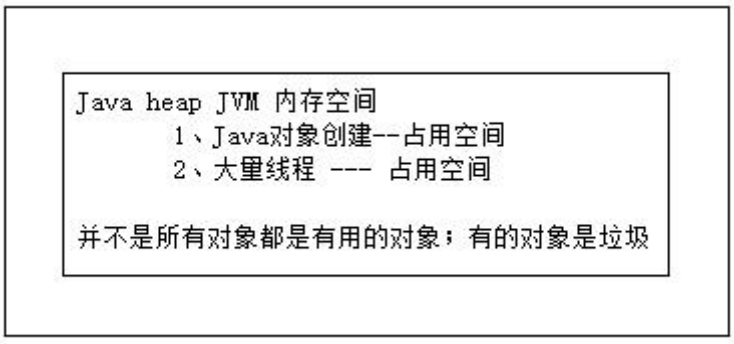
```
[root@qps004 java18]# cat application.yaml.un
server:
  port: 8888
  undertow:
    io-threads: 4
    worker-threads: 256
    buffer-size: 1024
    direct-buffers: true
    always-set-keep-alive: true
    url-charset: UTF-8
spring:
  application:
```

Undertow 优化后性能 TPS 情况：5000 TPS

Tomcat 性能情况如何：4500 TPS

3 为什么要进行 JVM 调优？

思考 1： 项目上线后，是什么原因使得我们必须进行 jvm 调优工作呢？？？



1、垃圾对象太多（java 线程，对象占满内存），内存被占满了，程序跑不动了!! --- 程序性能严重下降

2、垃圾回收线程太多，频繁回收垃圾（垃圾回收线程也会占用内存资源，cpu 资源），必然会导致程序的性能下降

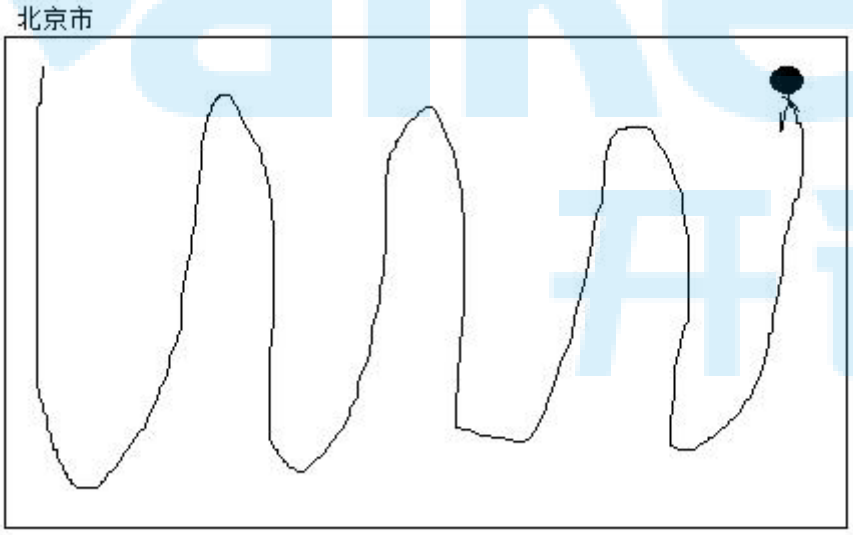
3、垃圾回收导致 STW（stop the world）

基于以上的原因： 项目上线后，必须对项目进行调优，否则程序运行一段时间后，随着流量增大，相当于为程序埋了一个定时炸弹；因此必须进行调优；

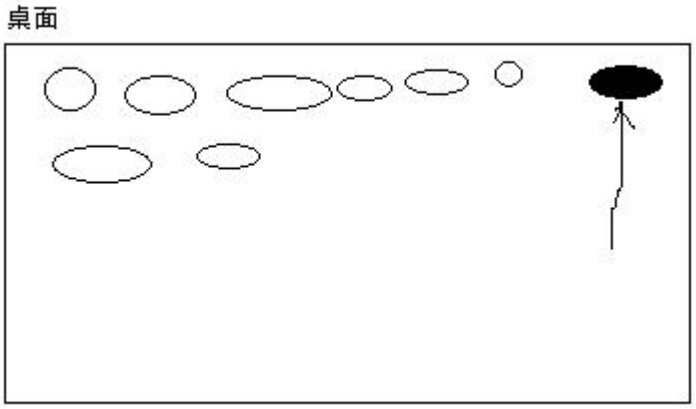
思考 2： jvm 调优本质是什么？？
答案： 垃圾回收，及时回收占用内存的垃圾对象，及时释放掉内存空间，让程序性能得以提升；让其他线程可以进入内存空间；
前提： 垃圾回收程序触发条件，是内存空间被装满了；

思考 3：是否可以把内存空间设置的足够大（更大），就不需要进行垃圾回收了？？
前提： 垃圾回收程序触发条件，是内存空间被装满了；
寻址能力：（是否有这么大的内存空间）
32 位操作系统： 2~32 == 4GB 内存
64 位操作系统： 2~64 == 16384PB 内存 ---- 满足 ，无限内存的空间
寻址速度：内存空间太大，寻址一个对象的消耗的时间比较长；
因此根据以上 2 点，可以知道： 内存空间的设置必须是一个合理的设置，不能太大，不能太小；

如果内存设置太大：一旦内存空间设置过大，且触发垃圾回收程序，寻找这个垃圾对象的过程就会非常耗时，此时程序都处于等待状态，这将会是一个灾难；



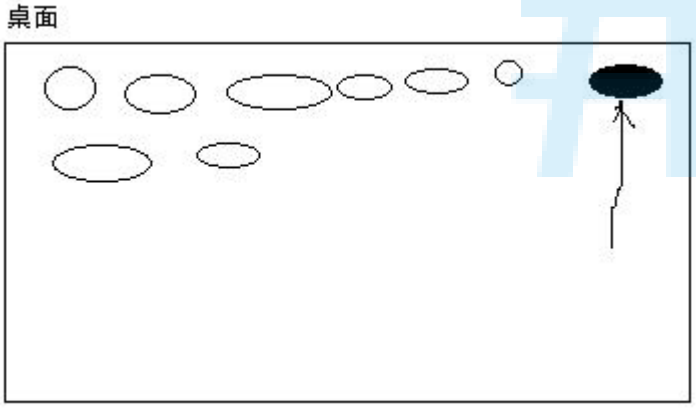
如果内存设置太小： 寻找这个垃圾非常快速，但是如果内存空间过小，将会导致频繁的垃圾回收；



基于以上说明，发现 JVM 内存空间，不能设置的太大，也不能设置的太小，设置一个 balance 的值；

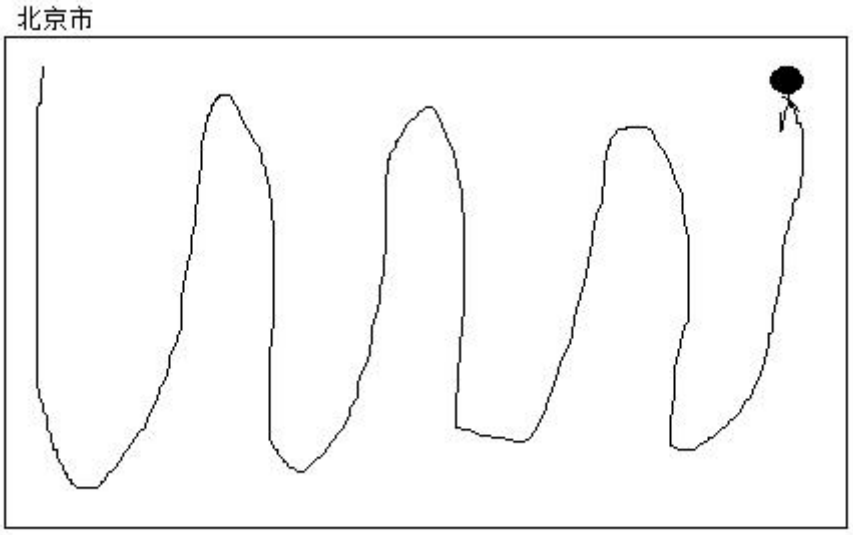
4 JVM 调优原则

1) gc 时间足够小 （堆内存设置足够小）



非常快速的定位垃圾对象，然后对此垃圾进行清除；

2）、gc 次数足够少 （堆内存设置足够大）



当内存空间被装满了，触发垃圾回收；因此“北京市” 一年才本装满一次，垃圾回收次数一年一次；回收的次数足够少；

3）发生 full gc 周期足够长 （最好不发生）

* metaspace 永久代空间合理，永久代一旦扩展，fullgc 一定会发生
* 老年代空间设置稍微大一些，防止 fullgc 发生
* 尽量让垃圾对象在年轻代被回收（90%）
* 尽量防止大对象的产生，一旦大对象多了以后，就可能发生 fullgc,甚至会发生 oom

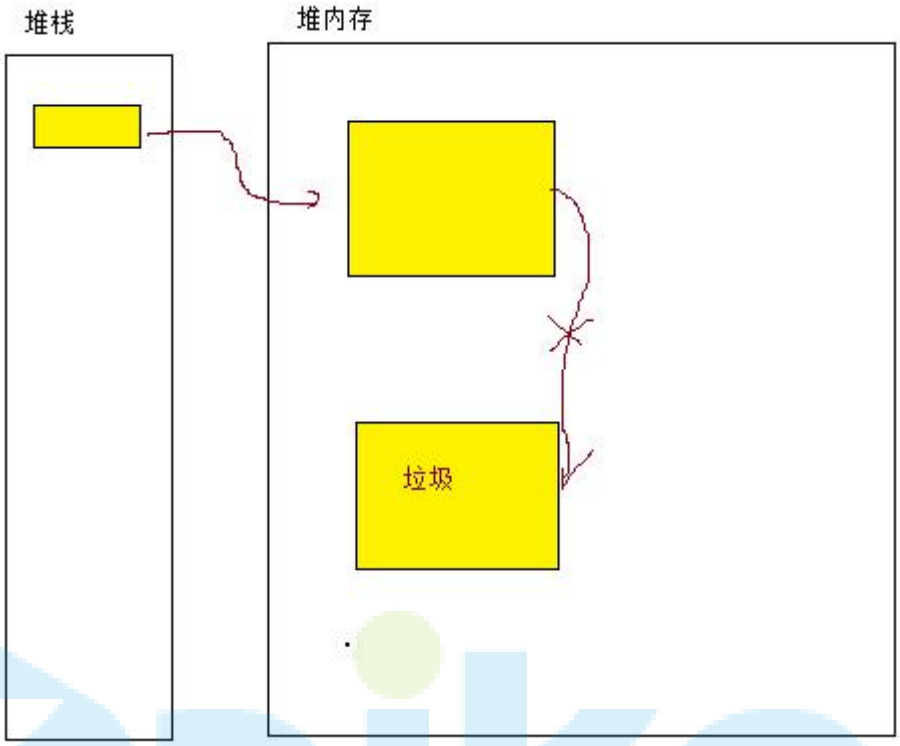
5 JVM 调优原理剖析

5.1 什么是垃圾？

JVM 调优的本质：回收垃圾对象； 但是什么是垃圾？？？

在内存中哪些没有被引用的对象就是垃圾（大量请求在内存创建很多的对象，这些对象并不会自己消失，必须进行垃圾回收，当然垃圾回收也不需要我自己编写垃圾回收程序，垃圾回收是 JVM 自动进行的）

注意： 在高并发模式下，内存会别瞬时间创建很多对象，此时内存空间必须及时被释放，否则会影响程序性能；



一个对象的引用消失了，这个对象就是垃圾，因此这个对象就必须被回收，释放掉内存空间；

5.2 怎么找垃圾?

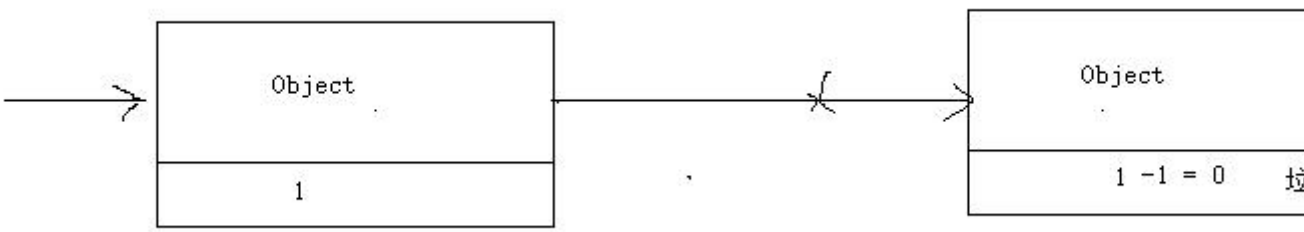
Jvm 中有 2 种方法找到这个垃圾：

1、引用计数算法

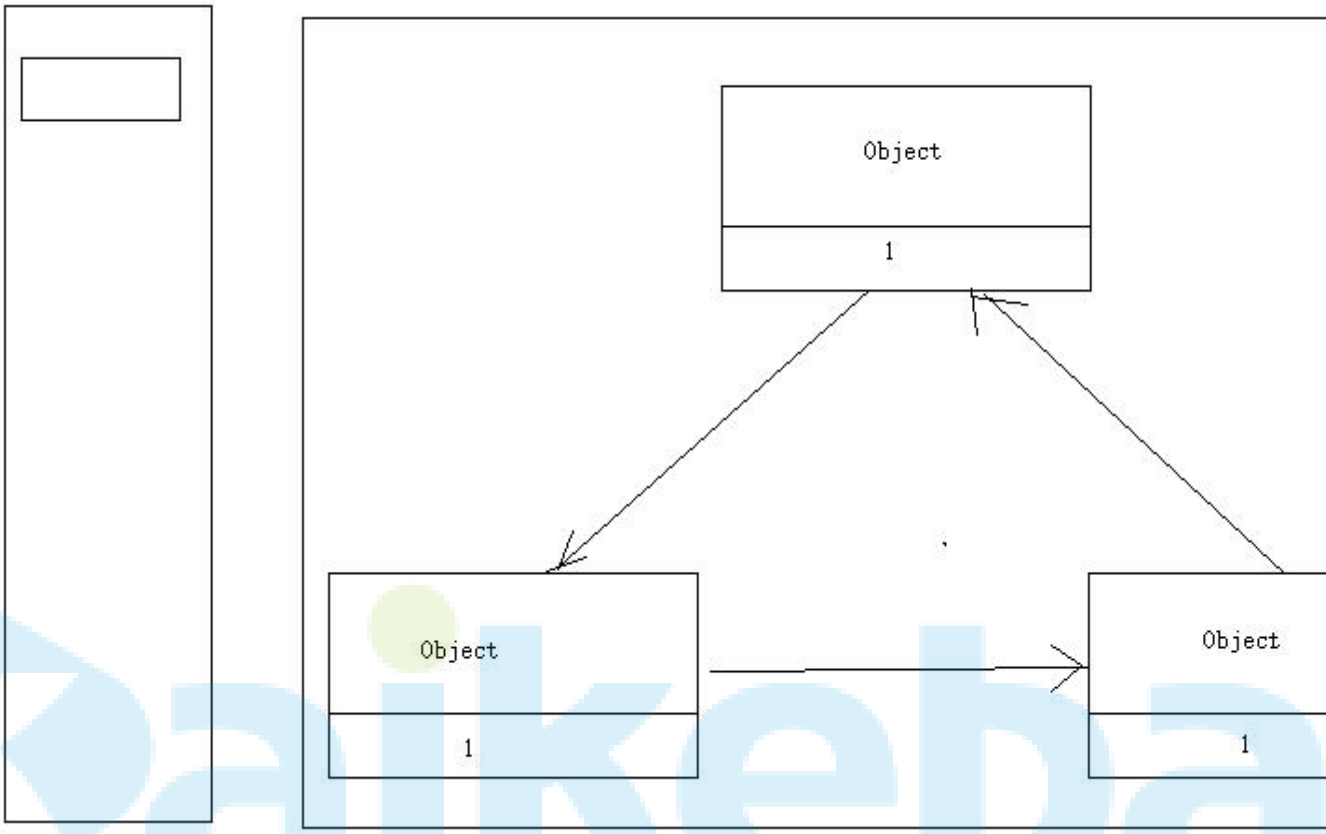
2、根可达算法 ------ hotspot　垃圾回收器都是使用这个算法

1）引用计数算法

对每一个对象都进行一个计数（对引用的计数），当计数变为 0 时候，此对象就是一个垃圾；

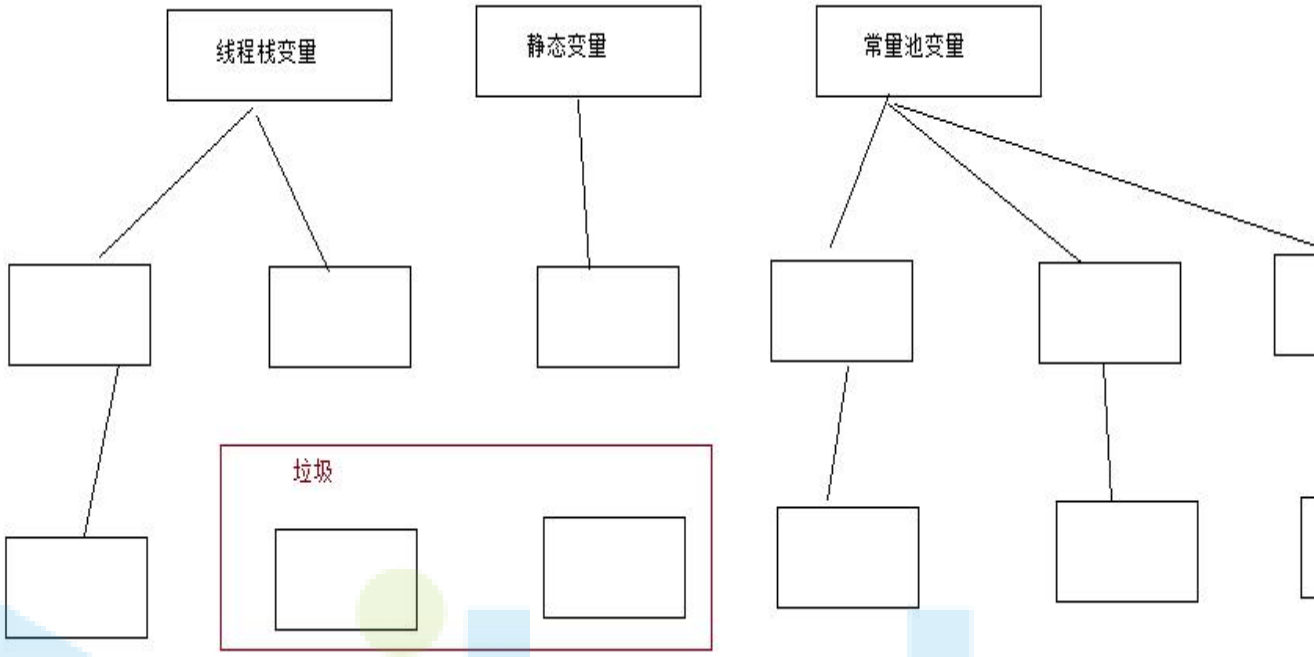


存在问题：　无法解决循环引用的问题



这 3 个对象处于循环引用状态，计数都不为 0，因此无法判断这个对象就是垃圾；

2）根可达算法



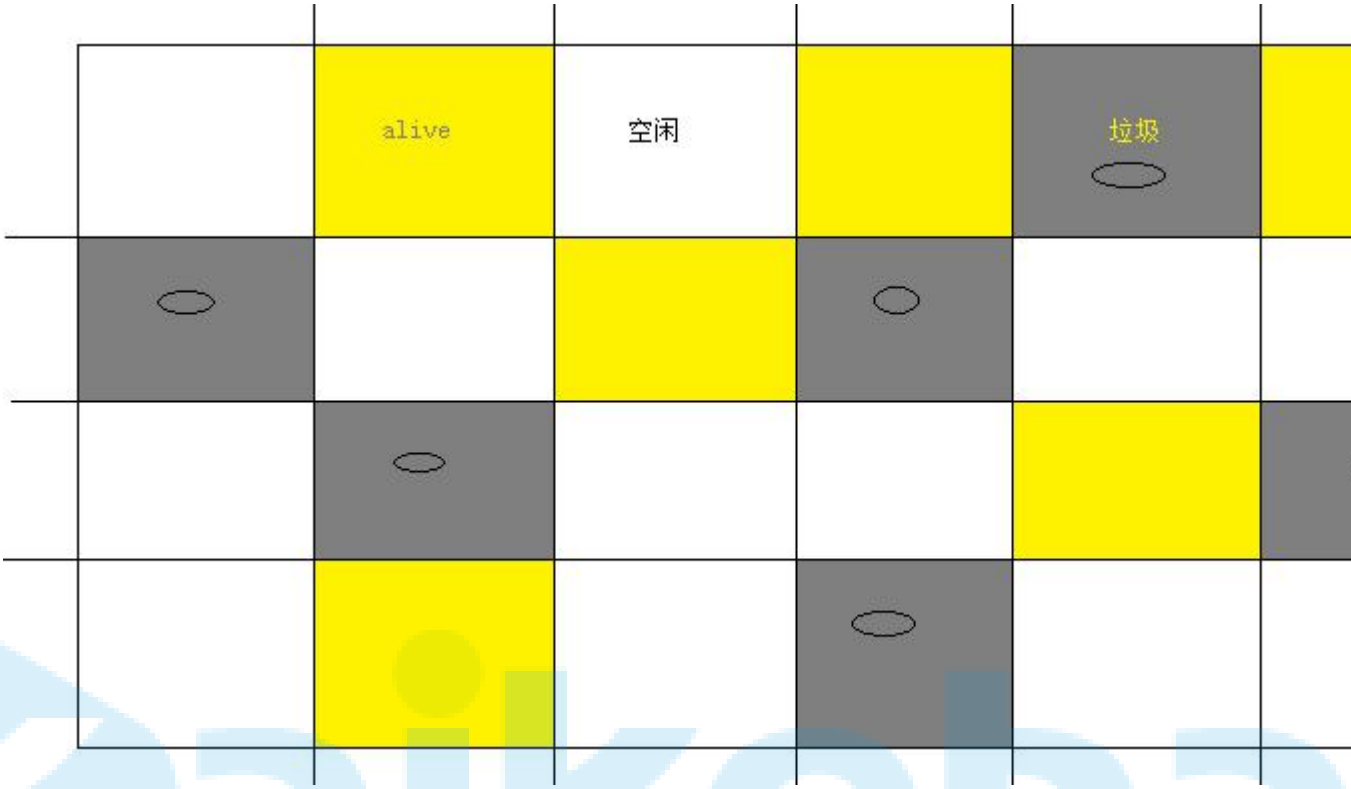
根据根对象寻找垃圾，如何可达的就不是垃圾，如果找不到的就是垃圾；

5.3 如何清除垃圾？

JVM 提供 3 种方法，清楚垃圾对象：

1、mark-sweep　标记清除算法
2、copying 拷贝算法
3、mark-compact 标记压缩算法

1）mark-sweep　标记清除算法



标记清除算法：

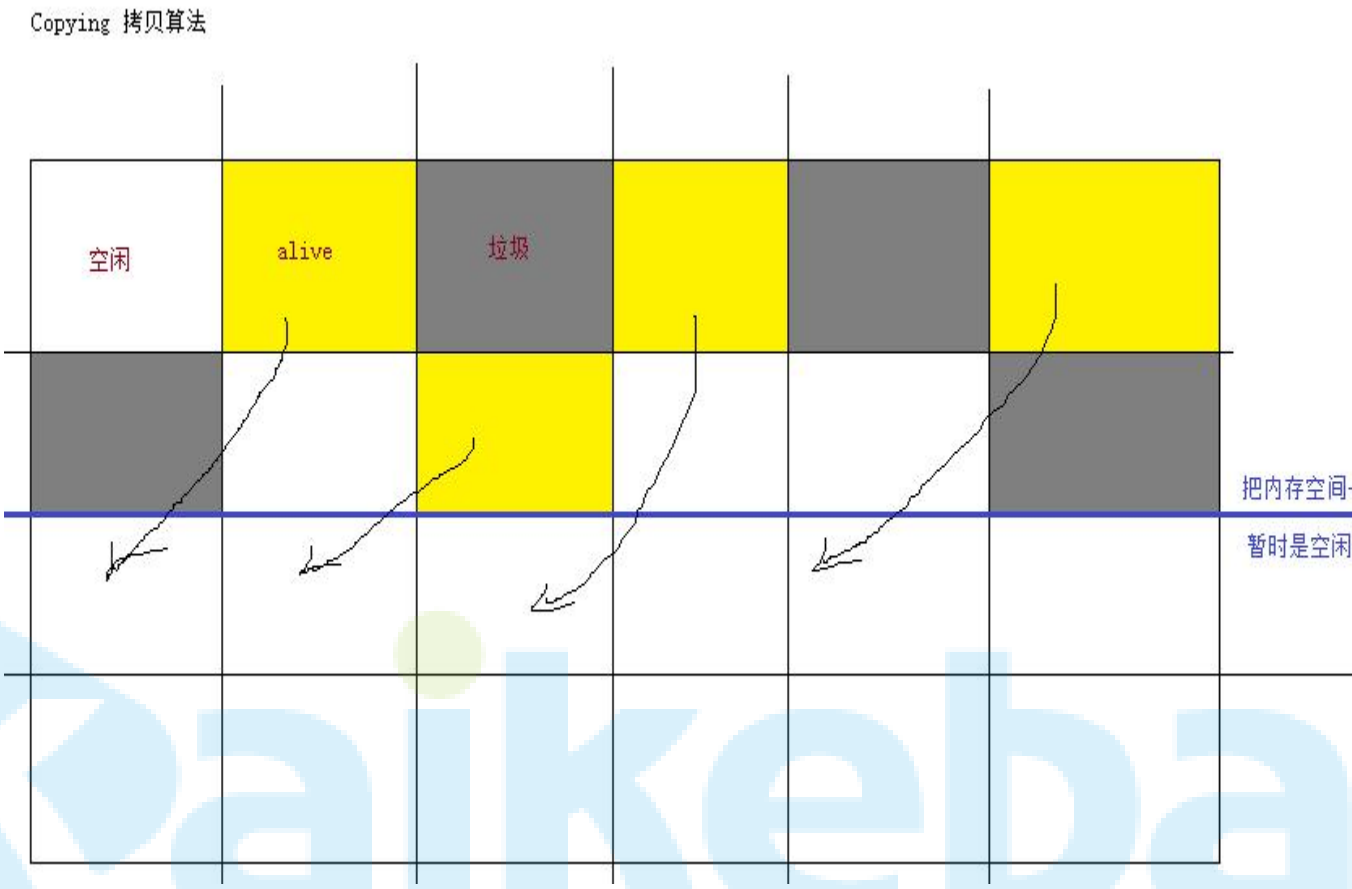
1、使用根可达算法找到垃圾，对垃圾对象进行标记

2、对标记对象进行清除即可

优点： 简单，高效

缺点： 内存碎片（删除掉垃圾对象后，留下大量的不连续的内存空间）

2、copying 拷贝算法



Copying 算法： 一开始就把内存空间分为 2 个相同大小的空间，一半存储对象，另一半用作暂时存活对象的拷贝区；

1、选择存活对象

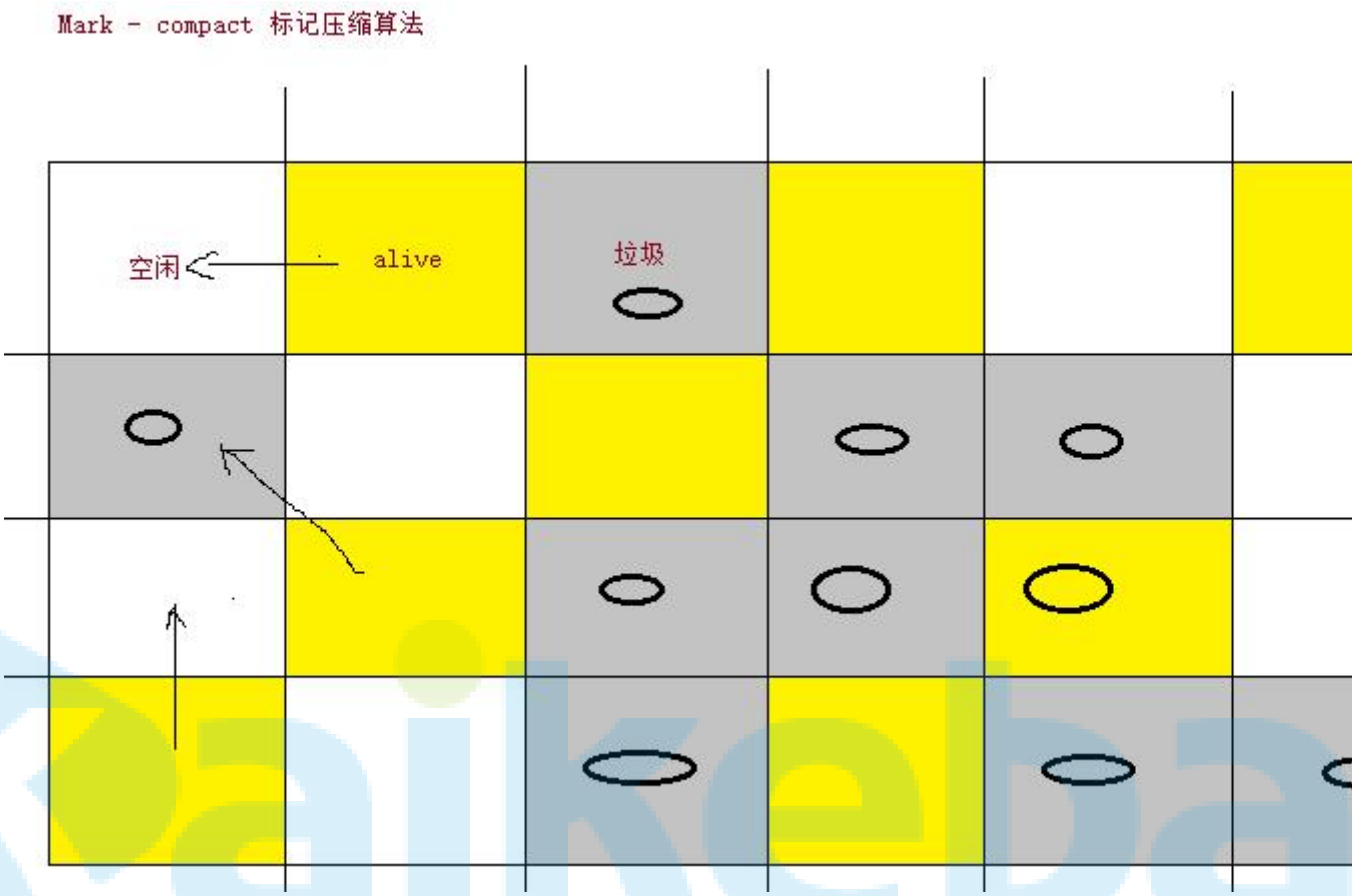
2、把存活对象拷贝到另一半空间内存中，且是连续的内存空间

3、把剩下的另一半的内存空间就全是垃圾对象，直接清除另一半空间即可；

优点： 简单，内存空间是连续，不存在内存碎片

缺点： 内存空间浪费

3、mark-compact 标记压缩算法



标记整理（压缩）算法：

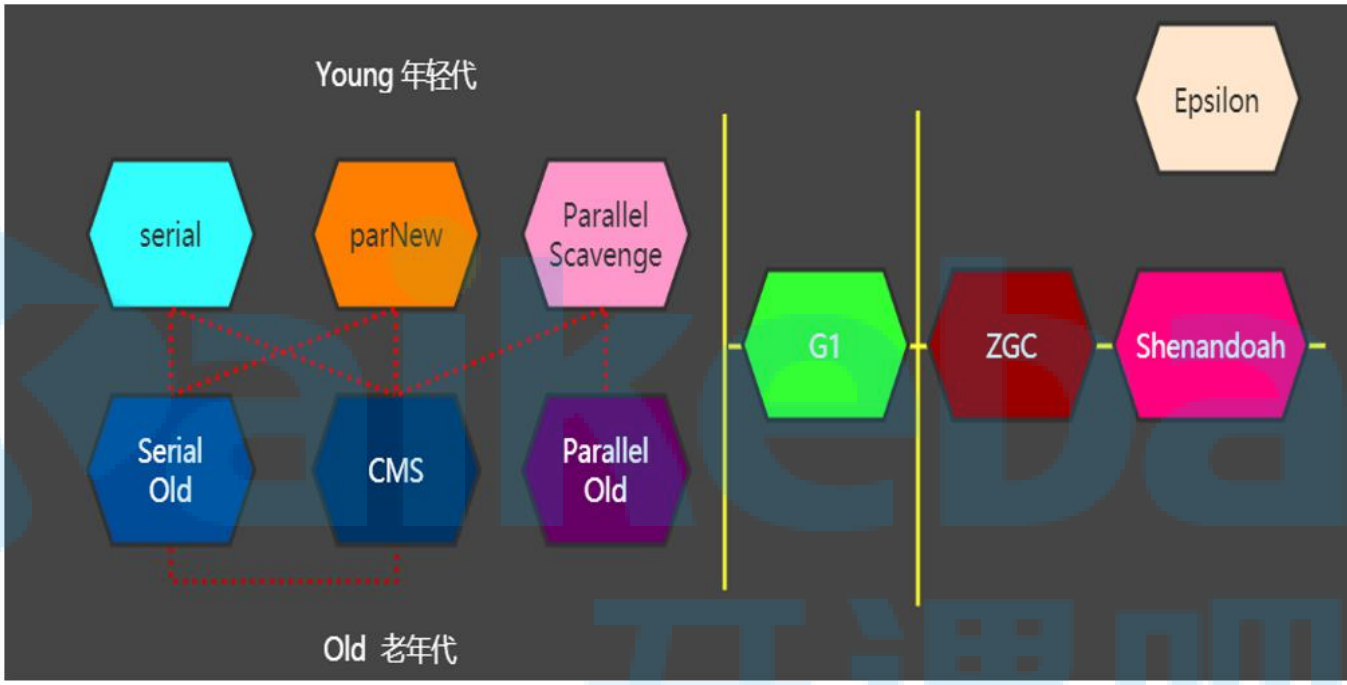
1、标记垃圾（只标记，不清除）

2、再次扫描（么有被标记的对象就是存活对象），找到存活对象，且把存活对象向内存一端进行移动（一端的内存空间是连续的内存空间）

3、清除另一端垃圾即可

5.4 垃圾回收器?

C/C++语言并没有垃圾回收器，他们的垃圾回收的工作需要程序员编写相应的代码，实现垃圾回收； 但是 Java 语言来说，提供了专门的垃圾回收器；



Jvm 提供了 10 种垃圾回收器，思考问题： 项目上线后，考虑使用哪种垃圾回收器？？？

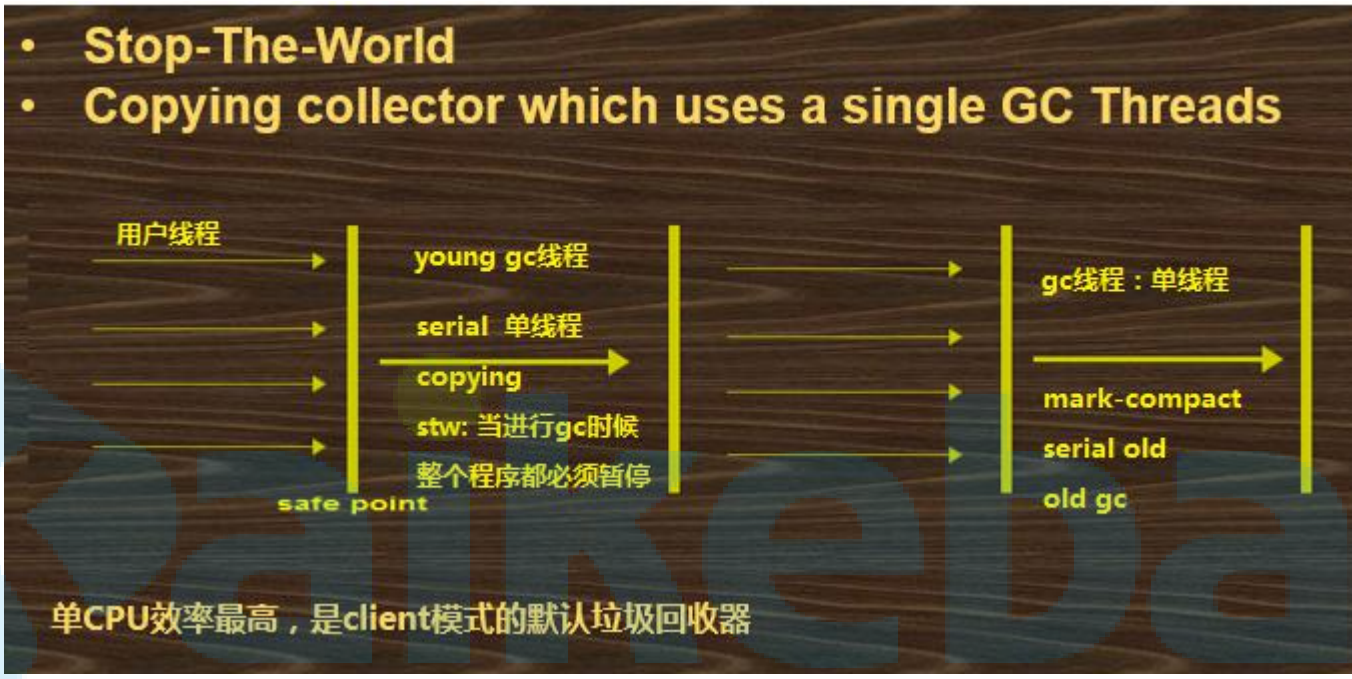
1、Serial ,serial Old : 串行化的垃圾回收器
2、parNew ， CMS 并发，并行垃圾回收器
3、parallel scavenge , parallel old 并行的垃圾回收器
4、g1 逻辑上分代垃圾回收器
5、zgc 实验室，支持 TB 级别垃圾回收
6、shenandoah openjdk 专属垃圾回收器
7、epsilon 调试用的垃圾回收器

常用垃圾回收器组合：

1、Serial ,serial Old : 串行化的垃圾回收器，适合单核心 cpu 服务器情况
2、parNew ， CMS 并发，并行垃圾回收器，响应时间优先组合
3、parallel scavenge , parallel old 并行的垃圾回收器，默认 JDK8 垃圾回收器，吞吐量优先的垃圾回收器
4、g1 逻辑上分代垃圾回收器

6 垃圾回收器原理

6.1 Serial ,serial Old

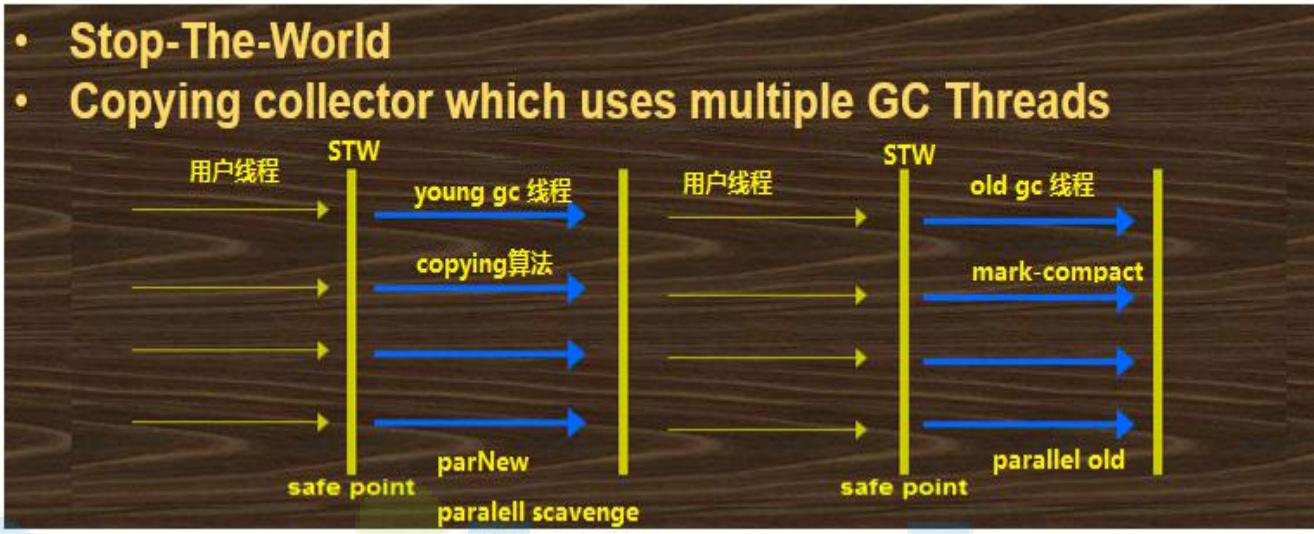


当年轻代堆内存空间被占满了后，触发垃圾回收；此时用户线程都必须暂停执行（也就是此时程序处于停顿状态，我们把这种状态叫做 STW: stop the world）,STW 必须等到 gc 垃圾回收结束后，用户线程再度恢复执行；

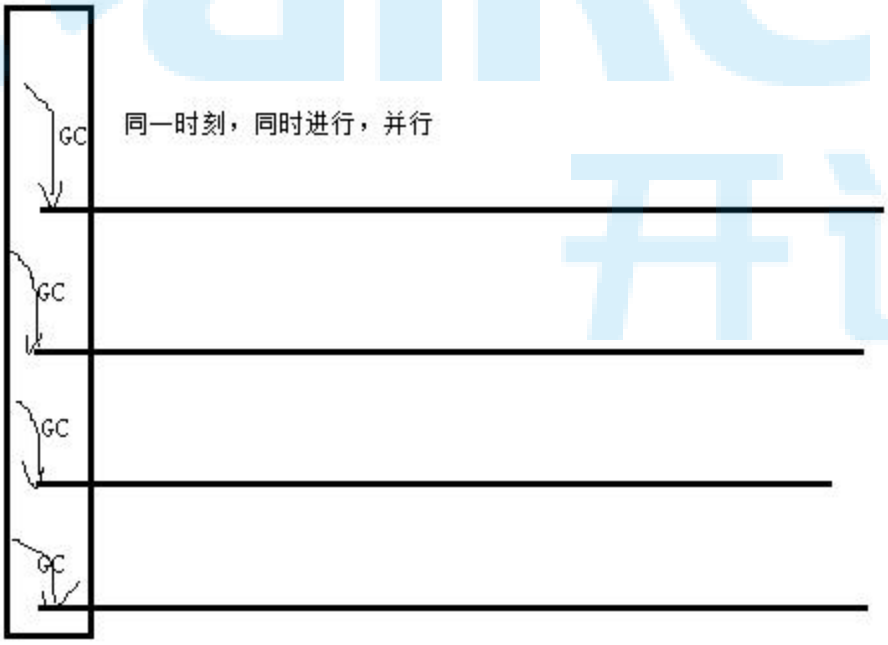
6.2 PS、PO

并行的垃圾回收器，默认 JDK8 垃圾回收器，吞吐量优先的垃圾回收器，适合多核心 cpu 并行的情况；

Parallel Scavenge + Parallel Old 组合：



并行：

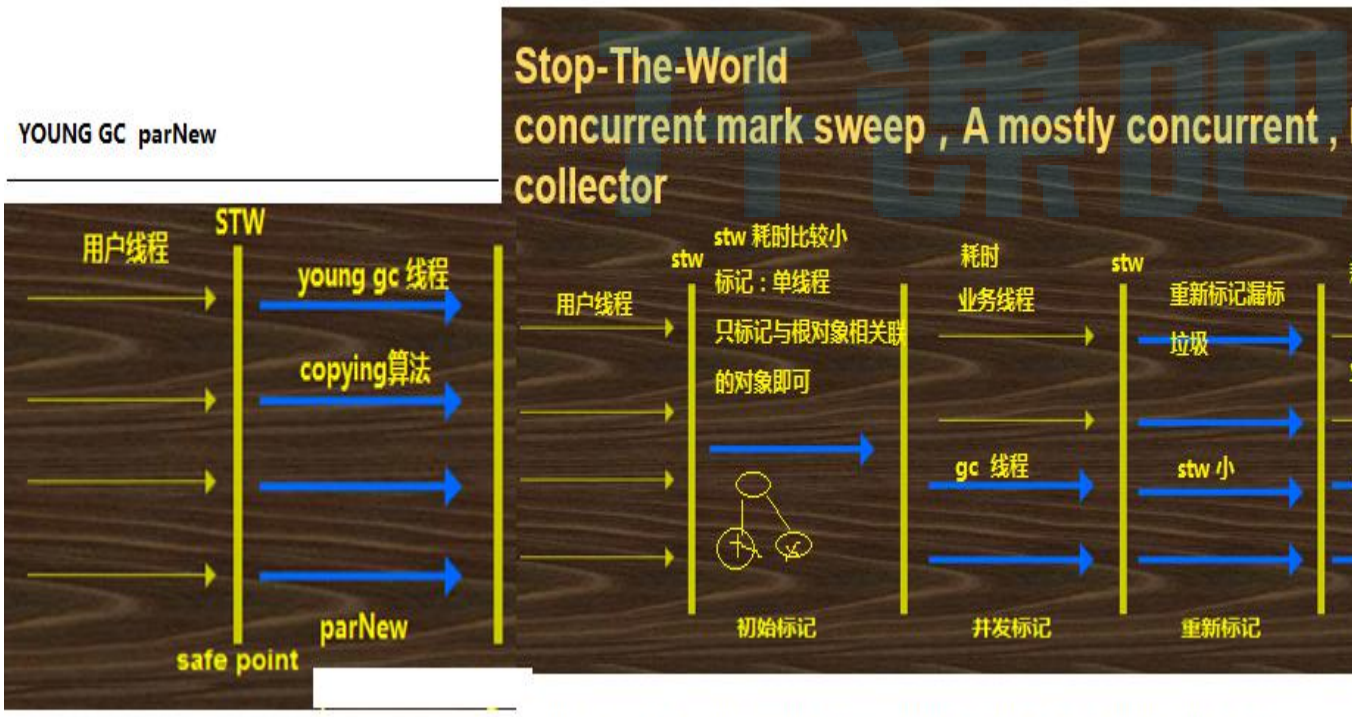


并发：

GC 抢占式执行
并发状态
一段时间之内

6.3 parNew、CMS

Old GC 老年代垃圾回收器，cms负责老年代的垃圾回收；

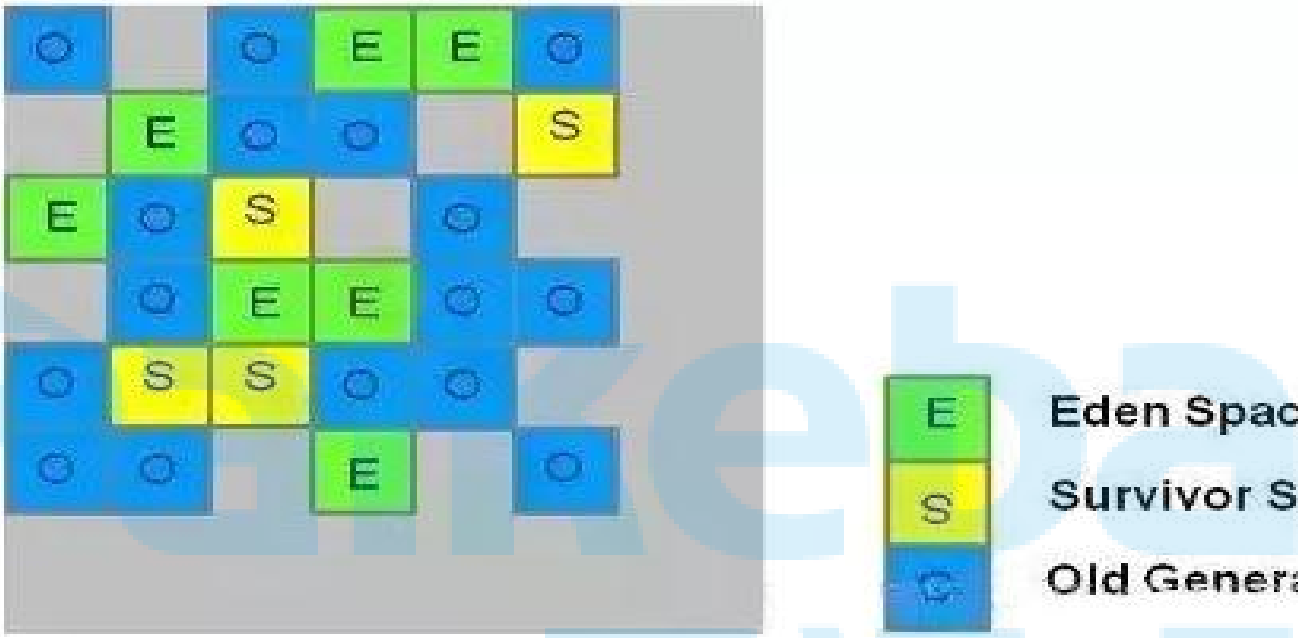


cms设计：尽量减少stw时间，因此我们把cms垃圾回收器叫做 响应时间优先 垃圾回收器；

parNew: 并行垃圾回收器
CMS : 并发垃圾回收器

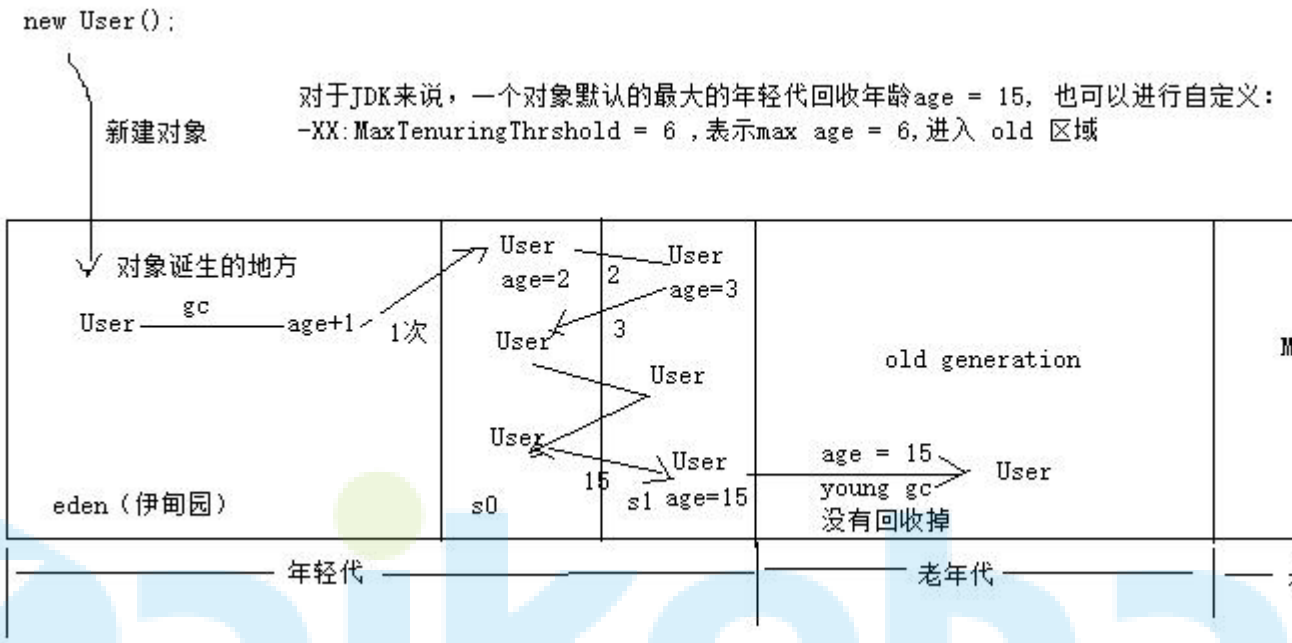
6.4 G1

G1 垃圾回收器在逻辑上进行分代，相对前面的 6 种垃圾回收器（在物理上进行分代），因此 G1 使用非常简单，把年轻代，老年代垃圾回收器合二而一；



使用 G1 垃圾回收器，把真个 JVM heap 内存空间划分为 2024 个大小独立 Region 区域，每一个 Region 区域根据堆空间内存实际大小进行设置的，整体被控制在 1MB ~ 32MB，且为 2~n 次幂，可以通过-XX:G1HeapRegionSize 设定；
一旦设定了 Regoin 区域的值，所有的 Regeion 区域的大小都是相同的，且 JVM 生命周期内不会发生变化；

7 内存分代模型



通过内存分代模型得知： 大多数对象都在年轻代结束了生命周期，很多对象都在 15 次垃圾回收中被回收掉了；只有超过 15 次还没有被回收掉的，才会进入老年代空间；

垃圾回收触发时机：

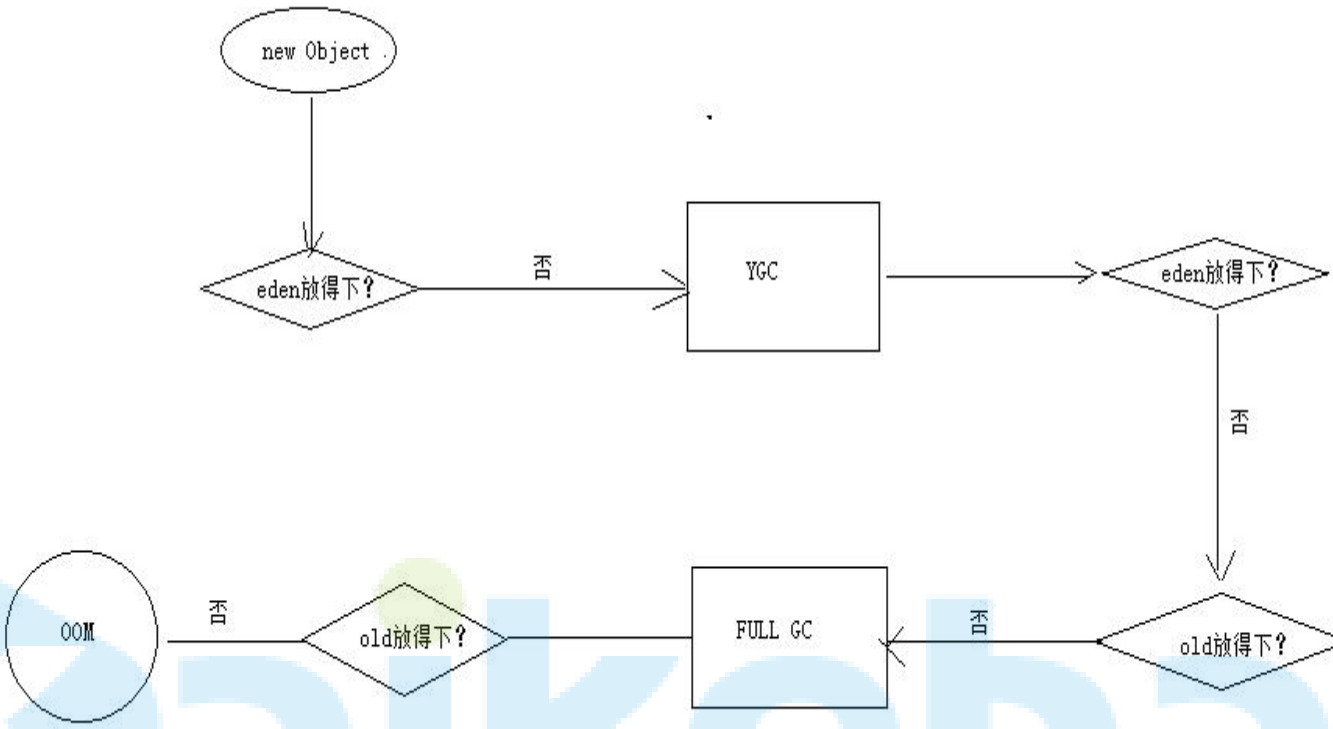
1、ps+po ： 当堆内存被占满后，触发垃圾回收（young eden 区域被占满了，触发 gc, old 区域被占满了，触发 old gc）

2、cms 垃圾回收器

* jdk1.5 : 68% 触发垃圾回收器

*Jdk1.6+: 92% 触发垃圾回收器

问题： 一个新对象来了，这个对象是一个大对象，eden 区域已经放不下了，此时会发生什么？？


new Object
eden放得下?
否
YGC
eden放得下?
否
old放得下?
否
FULL GC
否
old放得下?
否
OOM