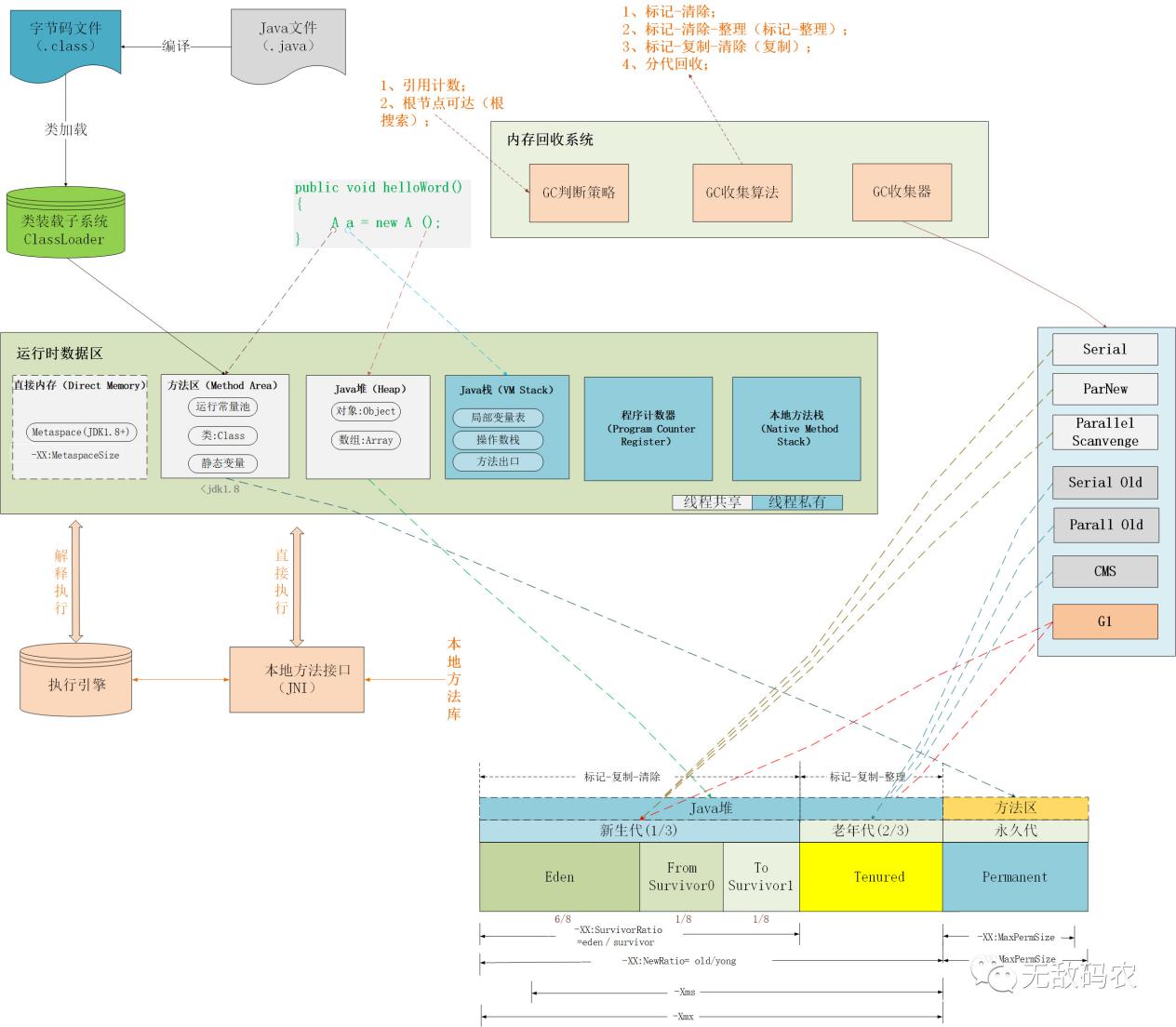
# JVM发展历史

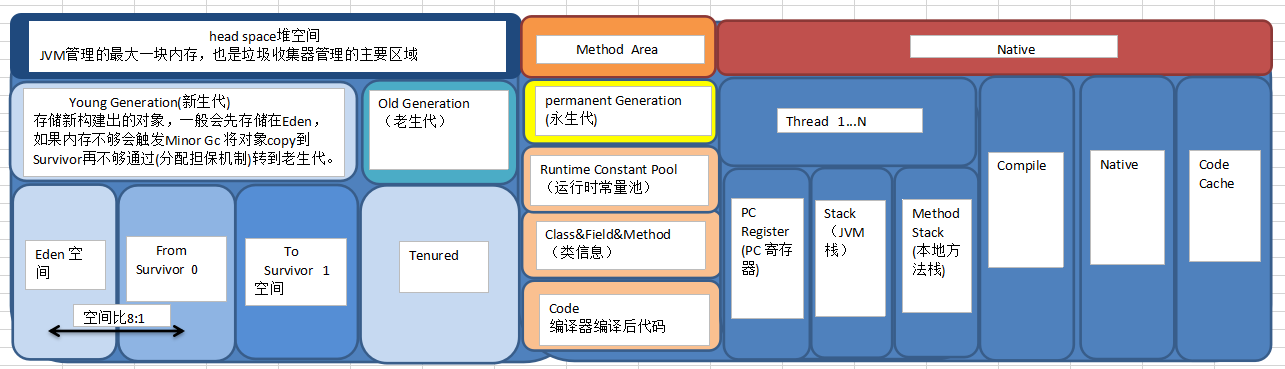
第一商用虚拟机Classic VM，随后出现Exact VM 其因内存管理相对优秀而出名，但是不久之后就被Hotspot VM取代了（也就是现在使用最多的虚拟机），Hotspot 原型始于Longview Technology公司的Strongtalk VM，1997被sun公司收购继承了前面sun公司两款虚拟机的有点产生了Hotspot VM。而后2008-2009年Oracle公司收购sun公司和BEA公司得到两款虚拟机分别是Hotspot和JRockit，而后Oracle将JRockit上的优点结合到了Hotspot上。

虚拟机概图：



# JVM的自动内存管理机制

内存结构图：



## 运行时数据区域

运行时数据区域分私有区和共享区：

私有区：PC寄存器、JVM栈、本地私有栈

共享区：堆、方法区

下面分别介绍。

### 程序计数器（PC寄存器）：

在CPU上的一小块内存，在字节码解析工作的时候通过改变该计数器的值来获取下一条需要执行的字节码指令，分支、循环、跳转、异常处理、线程恢复等基础功能都需要通过该计数器完成。

Java虚拟机的多线程是轮流执行CPU的，为了能够切换后能恢复到线程原来的位置，就是通过该技术器来记录线程的位置也可以理解为地址，每个线程有独立的计数器互不影响。

### JVM栈

生命周期与线程相同，每个方法执行时都会创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态链接、方法出口等。局部变量表存储编译期可以知道的基本数据类型和对象引用类型（reference），如下面的str

String str = new String(“asd”)

### 本地方法栈

起作用和JVM栈基本相似，不过他处理的是除java本身之外的其他语言的方法函数，也就是Native（本地）方法服务。

### Java堆

是Java虚拟机管理内存中最大的一块，java虚拟机启动时就被创建，几乎所有的对象示例都在这里分配内存，因此也是垃圾收集管理的主要区域，也称之为GC堆。它可分为新生代和老年代。新生代里又分为Eden空间，Survivor-From空间，Survivor-To空间。

### 方法区

也是线程共享区域，运行时常量池(Runtime Constant Pool) 类信息(Class & Field & Method data) 编译器编译后的代码(Code)。

### 运行时常量池

是方法区的一部分，说这个之前我们先了解常量池。常量池：主要用于存储字面量和符号引用量（字面量相当于Java语言层面常量的概念，比如：字符串常量、声明为final的常量等),当然常量池还会存储包含类的完整名称、字段名称和描述符、方法名称和描述符等。就如下例子：

public class M {

private int m;

private String mstring = "chen";

public void f() {

}

}

常量池：

⇒ javap -verbose M

......

Constant pool:

#1 = Methodref #5.#20 // java/lang/Object."<init>":()V

#2 = String #21 // chen

#3 = Fieldref #4.#22 // com/lwork/mdo/M.mstring:Ljava/lang/String;

#4 = Class #23 // com/lwork/mdo/M

#5 = Class #24 // java/lang/Object

#6 = Utf8 m

#7 = Utf8 I

#8 = Utf8 mstring

#9 = Utf8 Ljava/lang/String;

#10 = Utf8 <init>

#11 = Utf8 ()V

#12 = Utf8 Code

#13 = Utf8 LineNumberTable

#14 = Utf8 LocalVariableTable

#15 = Utf8 this

#16 = Utf8 Lcom/lwork/mdo/M;

// 方法名称

#17 = Utf8 f

#18 = Utf8 SourceFile

// 类名称

#19 = Utf8 M.java

#20 = NameAndType #10:#11 // "<init>":()V

#21 = Utf8 chen

#22 = NameAndType #8:#9 // mstring:Ljava/lang/String;

// 类的完整路径，注意class文件中是用"/"来代替"."

#23 = Utf8 com/lwork/mdo/M

#24 = Utf8 java/lang/Object



当然其中还包含I、V、、LineNumberTable、LocalVariableTable等代码中没有出现过的常量，其实这些常量是用来描述如下信息：方法的返回值是什么？有多少个参数？每个参数的类型是什么…… 这个示例非常直观的向大家展示了常量池中存储的内容

因为：

Class文件存储的各种信息，都需要加载到虚拟机之中才能运行和使用。

所以：

运行时常量池就可以理解为常量池被加载到内存之后的版本。

### 直接内存（堆外内存）Direct Memory

直接内存并不是虚拟机运行时数据区的一部分，也不是java虚拟机规范中定义的内存区域，使用时可向系统直接申请访问速度优于堆内存。读写频繁的情况下可以使用直接内存，性能会有提高，比如JDk1.4加入的NIO库，就是直接使用的Native 函数直接分配的直接内存，通过存储在java队中的一个对象DirectByteBuffer做这块内存的引用操作，他收到本机内存的限制，所以在通过Xmx设置虚拟机内存的时候要注意，防止报OutOfMemoryError内存溢出异常。

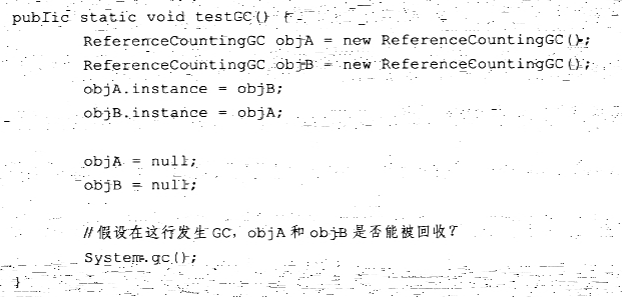
# 垃圾收集器与内存分配策略

##### 如何判断对象是否需要回收（是否已死）

1. 引用计数法：给对象添加一个引用计数器，被引用时加1，引用失效时减1。

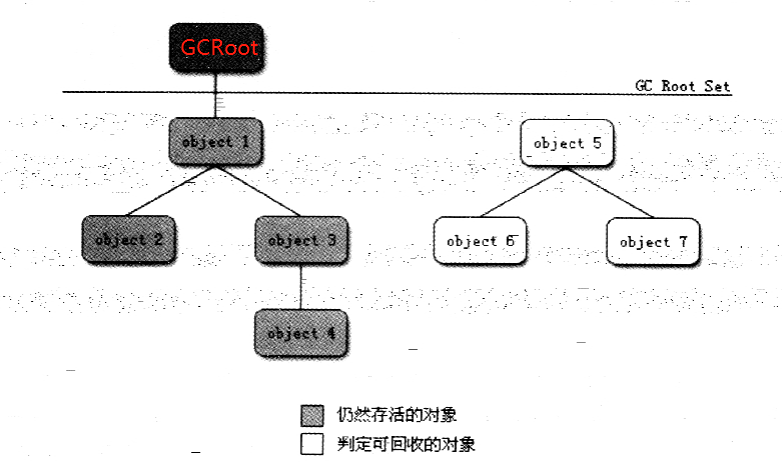
优点：实现简单效率高。

缺点：很难解决循环引用问题，如下例子



objA和objB为null时不能被回收。

1. **可达性分析算法（Reachability Analysis）（GCRoot）：**通过一系列称为GCRoot的对象作为起点从该节点向下搜索，搜索走过的路径称为引用链（Reference Chain），当一个对象与GCRoot没有任何链接时，就证明该对象不可用，如下例子：



GCRoot对象包括以下几种：

\*虚拟机栈中的引用对象

\*方法区中静态属性引用的对象

\*方法区中常量引用的对象

\*Native（本地方法栈）中引用的对象。

无论以上两种的判断法都没离开引用，再次引用分为以下四种：

强引用：程序中普遍存在如：Object A = new Object（），只要强引用存在就永远不会被回收。

软引用：有用但不是必需品，内存溢出异常前会把这些对象列入回收范围中进行第二次回收

弱引用：非必需品不管内存满不满都会被回收

虚引用：最弱的一钟引用，无法通过它来获取一个实例对象

1. **判断是否无用类：**

**\*该类所有的实例已经被回收，也就是java堆中不存在该类的任何实例**

**\*加载该类的ClassLoad已经被回收**

**\*该类对应的java.lang.Class对象没有在任何地方被引用，无法再任何地方通过反射访问该类**

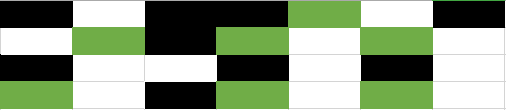
##### 垃圾回收算法

1. 标记-清除算法：

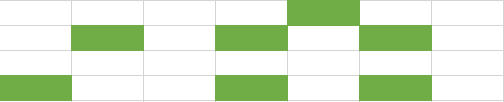
分两个阶段标记和清除，先标记出需要回收的对象，标记完成后统一回收，

两个缺点：一效率，标记清除的效率都不高；二内存碎片，空间碎片太多导致无法找到连续内存分配给大对象，从而提前出发GC

清除前：



清除后：

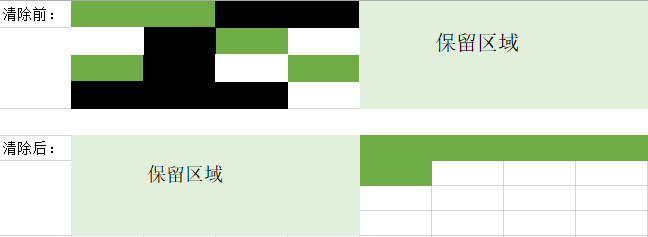


黑色代表为被标记对象；白色未使用空间；绿色存活对象

###### 复制算法：

将内存区域划分成两块相等的内存区域，当一块用完时将存活对象复制到另一块，然后把已使用过的清理掉。

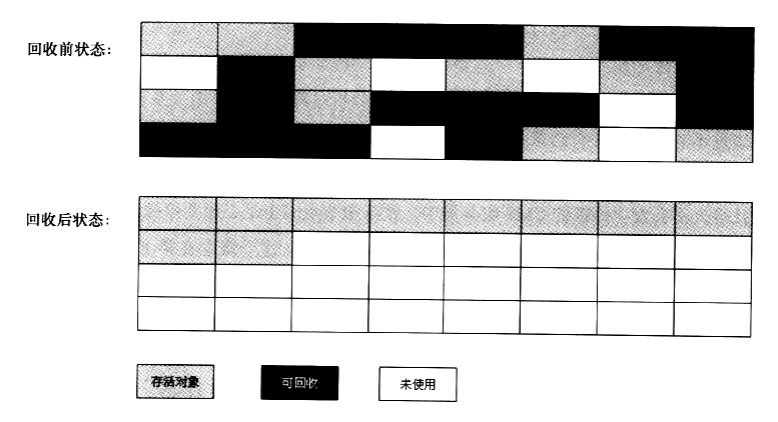
虽然解决了碎片化的问题但是这也会使珍贵的内存资源过多的浪费，进行GC的时间也会相应缩短



因大部分对象都是“朝生夕死”所以商业内存划分是将一块内存空间分成一块较大的Eden和两块一样大小的survivor，大小为Eden8 : Survivor1

###### 标记-整理算法

先进行标记，然后存活对象向一端移动然后，清除端边界以外的内存，内存利用率增大，也不会有碎片空间，但是效率相对较低。

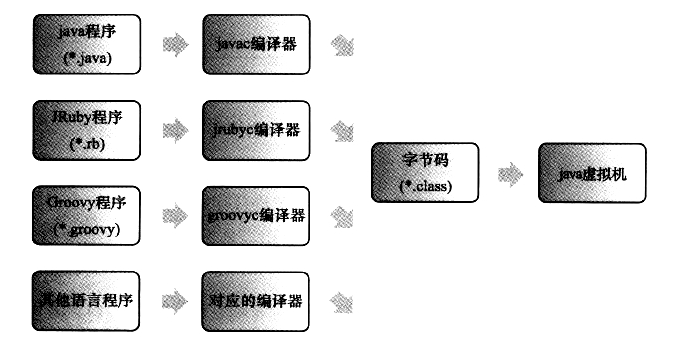


###### 分代回收算法

将java堆分成新生代（短期存活对象常用复制算法）和老年代（长期存活对象常用标记清除或者标记整理），根据对象的存活特点使用不一样的清除算法。

# 类文件结构

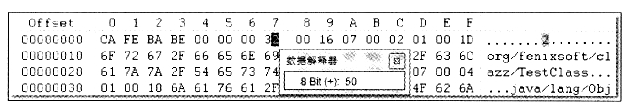
1.Java诞生之时有句话：一次编写，到处运行（Write Once，Run Anywhere），java 之所以跨平台的就是因为虚拟机，它可以通过javac编译器将.java文件编译成class文件，而JVM不与任何语言绑定，只与class文件这种特定的二级制文件格式有所关联。



1. Class文件是一组以8位字节为基础的二级制流，各个数据项目严格按照顺序紧凑地排列在Class文件之中，中间没有任何分隔符，整个Class文件中存储的内容几乎全部是程序运行的必要数据。
2. Class文件格式类似C语言中的伪结构，伪结构中有两种数据类型：无符号数和表；
3. 无符号数数据基本数据类型，u1，u2，u3，u4代表1个节，2个字节，4个字节，8个字节，其用来表示描述数字，索引引用，数量值或者按照UTF-8编码构成的字符集。
4. 表多个无符号数或者其他的表作为数据构成的复合数据，习惯以\_info结尾。
5. Class文件格式



头四个字节称魔数（0xCAFEBABE），用于确认是否是一个能被JVM接受的Class文件，存储Class文件的版本号。5-6个字节minor\_version次版本号，7-8个字节major\_version主版本号

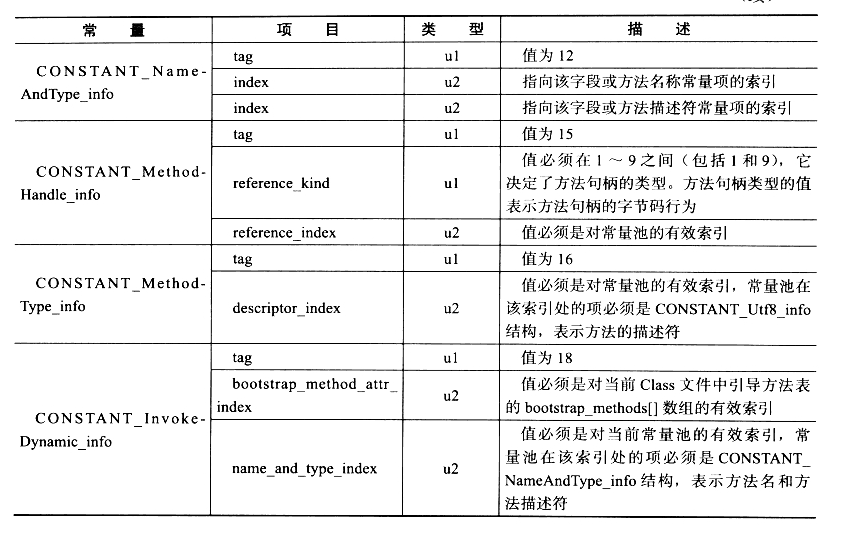


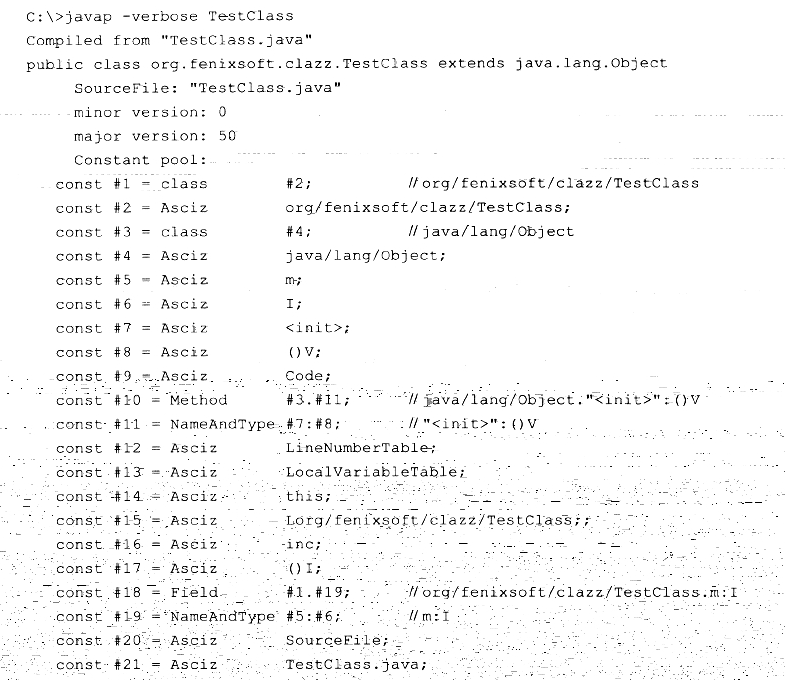
接着是常量池入口，可以理解为Class文件之中的资源仓库，它是与其他项目相关最多的数据类型，也占用Class文件空间最大的数据项目之一，也是Class文件第一个出现表类型数据项目。

由于常量池的数量不固定所以在常量池入口前放置一项u2类型的数据，代表常量池计数值（constant\_pool\_count）其地址偏移量是0x00000008十六进制0x0016即十进制22，代表常量池中有21项常量从1-21其中0做特殊考虑。

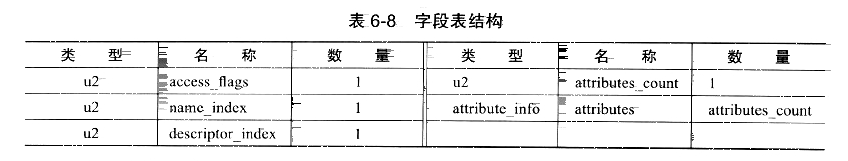
常量池中主要存储两大常量：字面量（Literal）和符号引用（Symbolic Reference），字面量指文本字符串和基本数据常量和被final修饰等，符号好引用指：类和接口权限定名，字段的名称和描述符，方法的名称和描述符。

常量池总结构表（包含了14种常量还差7种有些事高版本JDK才加入的）：

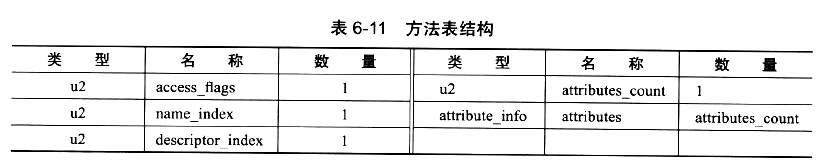


常量池测试结果对应的常量数据表：

字段集合field\_info:存储字段作用域修饰符（public，default，protect，private）、static、final、volatile、transient、字段数据类型（基本，对象，数组）、字段名称



方法集合method\_info：和字段相似，存储方法修饰信息等



属性集合attribute\_info：