# -----------回顾基础知识

JRE:JVM（bin）+Java 语言的核心类库（lib）

JDK:JRE+Java的开发工具



# JVM的工作原理

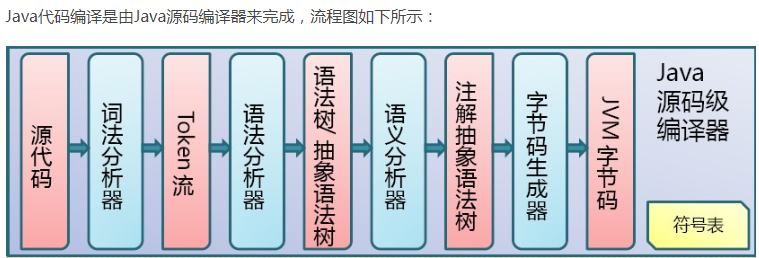
# JVM是什么？

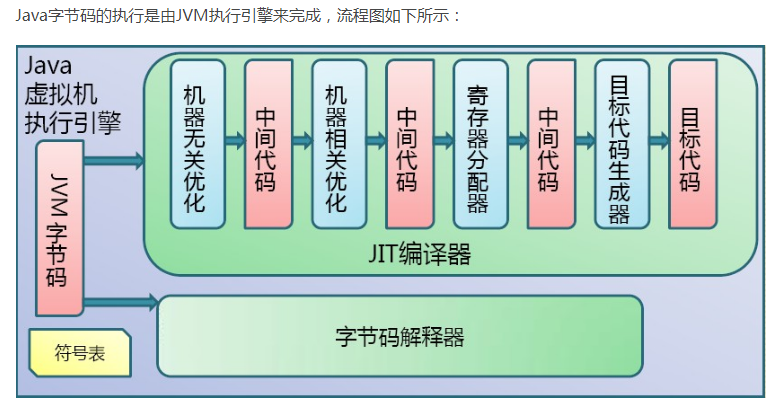
关键字：接口、跨平台

## 接口

JVM 是 编译后的 Java 程序（.class文件）和硬件系统之间的接口

( 编译后？由谁编译：由javac编译，javac 是收录于 JDK 中的 Java 语言编译器。该工具可以将后缀名为. java 的源文件编译为后缀名为. class 的可以运行于 Java 虚拟机的字节码。)





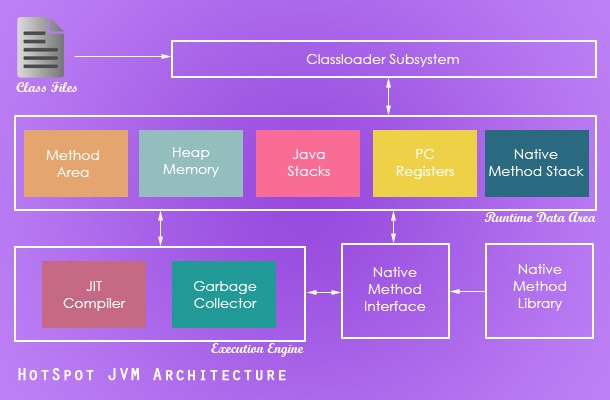
## 跨平台性（最关键的）

原理：编译后的 Java 程序指令并不直接在硬件系统的 CPU 上执行，而是由 JVM 执行。

为什么与平台无关：JVM屏蔽了与具体平台相关的信息，使Java语言编译程序只需要生成在JVM上运行的目标字节码（.class）,就可以在多种平台上不加修改地运行。Java 虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。因此实现java平台无关性。

# JVM的结构及运行

JVM = 类加载器 classloader + 执行引擎 execution engine + 运行时数据区域 runtime data area  
 classloader 把硬盘上的class 文件加载到JVM中的运行时数据区域, 但是它不负责这个类文件能否执行，而这个是 ‘执行引擎’ 负责的。



Native Method Interface

Native Method Library

通信模块：我们需要在父子模块间建立NML命令通道,父模块通过该命令通道将命令传递给子模块;还需要建立NMI 状态通道,子模块通过该状态通道将状态信息反馈给父模块...

## 类加载器 classloader

**（1）classloader**

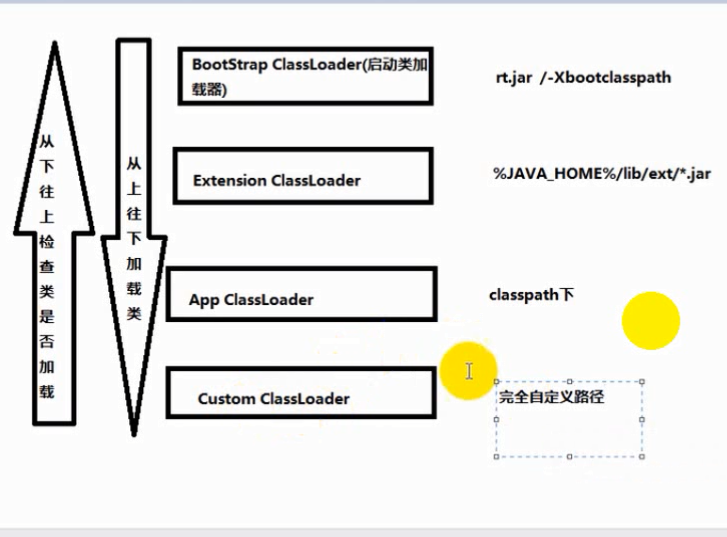
作用：装载.class文件

**（2）classloader 有两种装载class的方式 （时机）：**

1. 隐式：运行过程中，碰到new方式生成对象时，隐式调用classLoader到JVM

2. 显式：通过class.forname()动态加载(对反射比较熟的话，这里就可以引出反射聊)

**（3）classloader加载类的过程：采用双亲委托机制（双亲委派模型）**



Bootstrap class loader： 父类

当开启JVM时，这个类加载器被创建，它负责加载虚拟机的核心类库，如 java.lang.\* 等。

Extension class loader: 这个加载器加载出了基本 API 之外的一些拓展类。

AppClass Loader: 加载应用程序和程序员自定义的类。

Custom ClassLoader: 用户自定义的类加载器应该继承 ClassLoader 类。

**（4）双亲委派模型的工作过程：**

【1】当前 ClassLoader 首先从自己已经加载的类中查询是否此类已经加载，如果已经加载则直接返回原来已经加载的类。每个类加载器都有自己的加载缓存，当一个类被加载了以后就会放入缓存，等下次加载的时候就可以直接返回了。

【2】当前 classLoader 的缓存中没有找到被加载的类的时候，委托父类加载器去加载，父类加载器采用同样的策略，首先查看自己的缓存，然后委托父类的父类去加载，一直到 bootstrap ClassLoader.

【3】当所有的父类加载器都没有加载的时候，再由当前的类加载器加载，并将其放入它自己的缓存中，以便下次有加载请求的时候直接返回。

**（5）双亲委派模型的优点：**

安全！

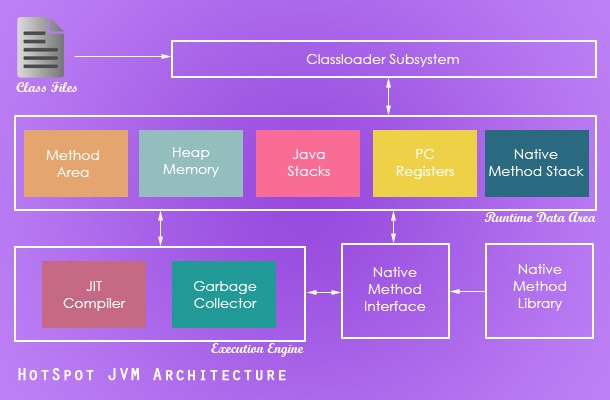
【1】避免用户自己编写的类动态替换 Java 的一些核心类，比如 String

【2】避免了重复加载，因为 JVM 中区分不同类，不仅仅是根据类名，相同的 class 文件被不同的 ClassLoader 加载就是不同的两个类，如果相互转型的话会抛java.lang.ClassCaseException.

## 执行引擎 execution engine

作用： 执行字节码，或者执行本地方法

## 运行时数据区域 runtime data area



**（1）概念：**

JVM 运行时数据区 (JVM Runtime Area) 其实就是指 JVM 在运行期间，其对JVM内存空间的划分和分配。JVM在运行时将数据划分为了6个区域来存储。

heap, java stack，method area, native method stack, PC register.

**（2）分类**

【1】PC程序计数器：就是个指示器一块较小的内存空间，可以看做是当前线程所执行的字节码的行号指示器。

【2】栈用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每一个方法被调用直至执行完成的过程就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。

【3】本地方法栈：与虚拟机栈的作用相似，虚拟机栈为虚拟机执行执行java方法服务，而本地方法栈则为虚拟机使用到的本地方法服务。

【4】方法区

方法区和堆区域一样，是各个线程共享的内存区域，它用于存储每一个类的结构信息，例如运行时常量池，成员变量和方法数据，构造函数和普通函数的字节码内容，还包括一些在类、实例、接口初始化时用到的特殊方法。

这块区域对应Permanent Generation 持久代。 XX：PermSize指定大小。

【5】运行时常量池

其空间从方法区中分配，存放的为类中固定的常量信息、方法和域的引用信息。

**【6】Java堆：**

被所有线程共享的一块存储区域，在虚拟机启动时创建，它是JVM用来存储对象实例以及数组值的区域，可以认为Java中所有通过new创建的对象的内存都在此分配。

Java堆在JVM启动的时候就被创建，堆中储存了各种对象，这些对象被自动管理内存系统（Automatic Storage Management System，也即是常说的 “Garbage Collector（垃圾回收器）”）所管理。这些对象无需、也无法显示地被销毁。

JVM将Heap根据对象的生命周期长短，把堆分为3个代：Young，Old和Permanent

**Young（年轻代）：**朝生夕死的那种对象

**Tenured（年老代）：**顽固分子，一直被引用，一直存活着。Young:Tenured=8:1

**Perm（持久代）：**用于存放静态文件，如今Java类、方法等。持久代对垃圾回收没有显著影响，但是有些应用可能动态生成或者调用一些class，例如Hibernate等，在这种时候需要设置一个比较大的持久代空间来存放这些运行过程中新增的类。持久代大小通过-XX:MaxPermSize=进行设置。

# GC

## 1.GC的基本概念

1. JVM 分别对新生代和旧生代采用不同的垃圾回收机制，gc分为full gc 跟Scavenge GC，当每一块区满的时候都会引发gc。垃圾回收为何时间不确定？在内存中低到一定限度时才会自动运行，从而实现对内存的回收。
2. **Scavenge GC**  
   一般情况下，当新对象生成，并且在Eden(新生代)申请空间失败时，就触发了Scavenge GC，堆Eden(新生代)区域进行GC，清除非存活对象。
3. **Full GC**  
   对整个堆进行整理，包括Young、Tenured和Perm。Full GC比Scavenge GC要慢，因此应该尽可能减少Full GC。有如下原因可能导致Full GC：

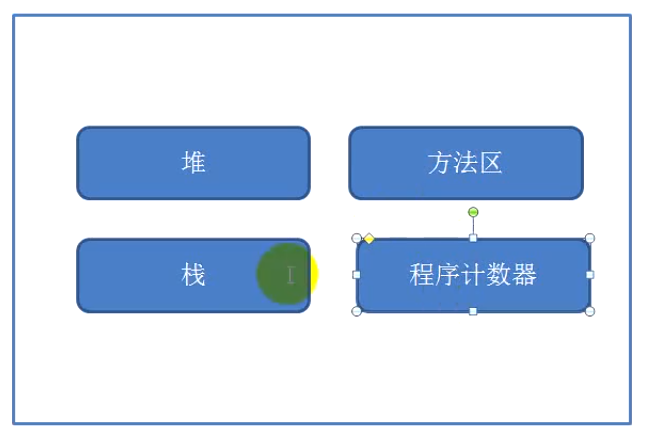
【1】上一次GC之后Heap的各域分配策略动态变化

【2】System.gc()被显示调用

【3】Perm域被写满

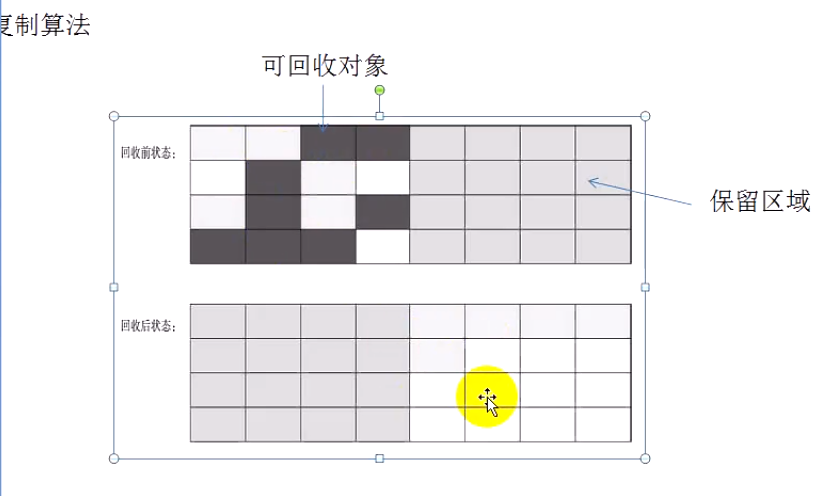
【4】Tenured被写满

## 2.垃圾回收算法

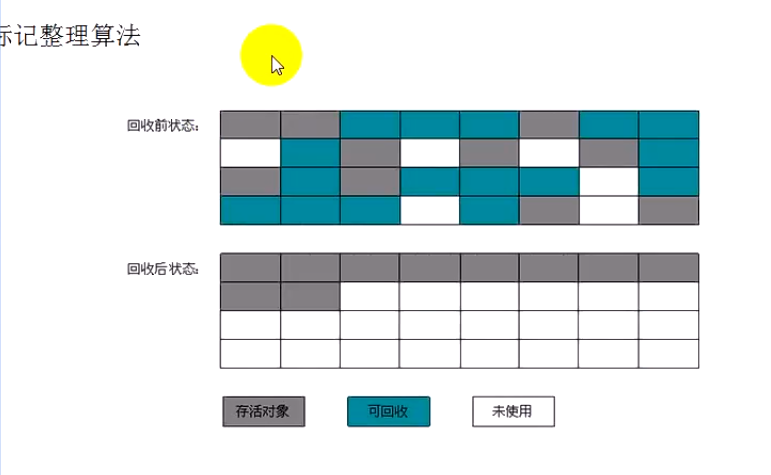




对象比较大的话，没有连续内存分配的话，会产生碎片



会缩小内存，但是这个比例不一样，所以需要去调内存大小至最优



JVM采用的：

**分代收集算法：**

新生代采用复制算法，老年代采用标记整理算法

## 3.垃圾收集器

【1】Serial：

串行，单线程，使用复制算法，新生代收集器，是默认收集器

缺点：单线程，当执行时，JVM会等待其执行，如果时间长的话，接收不了。但是占用内存小，小型的客户端常用，简单高效，但是多CPU 就没有优势以响应时间为标准

【2】Parnew：

并行，多线程，用户多个线程过来了，serial只会发起一个gc，多cpu用这个比较好，也是新生代收集器，以响应时间为标准

【3】Parallel scavenge：

新生代，并行，eclipse用的，以吞吐量为标准：代码执行时间/垃圾收集时间，css应用，大量代码。

【4】CMS：

服务端标配，并发标记整理收集器，老年代，多线程，最短时间收集，并发标记，将耗时长的和短的分开。

还有serialold parallel old 等等。。。

可以组合，JVM就是组合起来的。

一般CMS + Serial/ParNew

# 针对GC机制对JVM进行简单的调优

## 概念

【1】为何调优？

当发生：Gc fullGc时，JVM相当于停止工作，客户端相当于卡死，所以优化就是尽量减少gc、fullgc的发生

【2】注意：

堆在JVM是所有线程共享的，因此在其上进行对象内存的分配均需要进行加锁，这也是new开销比较大的原因。

鉴于上面的原因，Sun Hotspot JVM为了提升对象内存分配的效率，对于所创建的线程都会分配一块独立的空间，这块空间又称为TLAB

TLAB仅作用于新生代的Eden Space，因此在编写Java程序时，通常多个小的对象比大的对象分配起来更加高效

这种调优运用在hotspotJVM中

实际开发中有很大很多的项目，这里主要是调优的思路，

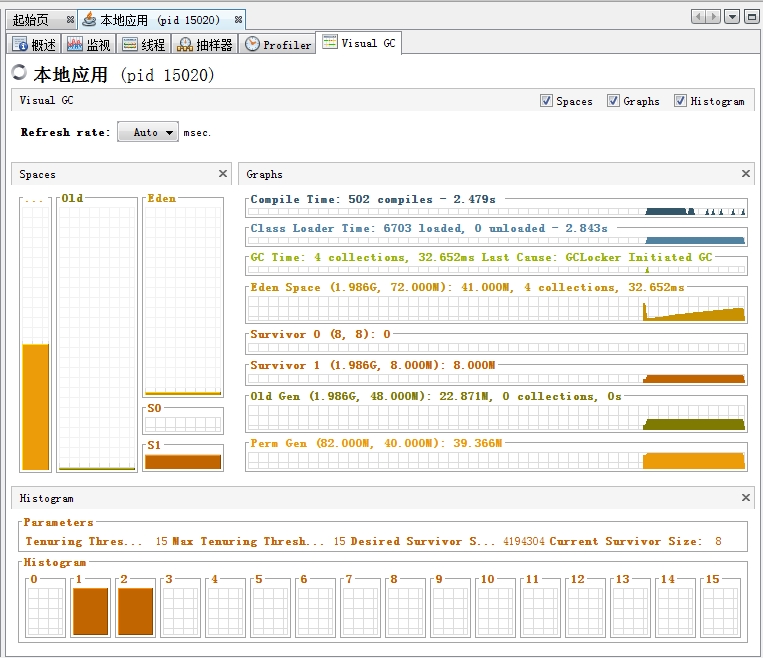
根据之前的图发生了垃圾回收几次几次到后来0次，包括从日志上看。

## 2.步骤

【1】视图工具：

Java -version查看是否是hotspot JVM

运行jvisualVM，然后左上角右键eclipse点运行



【2】日志

-verbose:gc

-XX:+PrintGCDetails

-XX:+PrintGCDateStamps

-Xloggc:D:/gc/gc.log

Xms512m

Xmx512m

NewSize=400m

MaxNewSize=400m

PermSize=90m

MaxPermSize=90

【2】配置范围

-Xms256m 最小堆内存（初始堆内存）

-Xmx1024m 最大堆内存

# 总结

**（两个中心三个基本点再加一个提升项）**

## （1）两个中心

JVM是什么

JVM的结构及运行

## （2）三个基本点

Classloader

runtime data area

GC

## 一个提升项

JVM的简单调优

阐述：首先，JVM是一个接口？，实现了java的跨平台性，它的结构是类加载器 + 执行引擎 + 运行时数据区域 ，类加载器 把硬盘上的class 文件加载到JVM中的运行时数据区域，通过执行引擎来执行，类加载器是用来加载.class文件，阐述classloader加载类的过程，而加载到运行时数据区域后，JVM在运行时将数据划分为了6个区域来存储，栈是？方法区是？...堆！存储对象，将对象分为年轻代，年老代，持久代而这些对象由垃圾回收器管理，引出GC回收机制...通过JVisualVM简单测试后，进行一个简单的调优，调优的思路。

高级程序员，架构师才会去调，我们了解即可，明白思路。

**万物皆对象，而对象基于JVM，可以根据JVM引出所有自己的强项。**