JVM

数据类型

- 基本类型
- 引用类型

堆与栈

栈

栈是运行时的单位

栈解决程序的运行问题，即程序如何执行，或者说如何处理数据

一个线程就会相应有一个线程栈与之对应，这点很容易理解，因为不同的线程执行逻辑有所不同，因此需要一个独立的线程栈

栈因为是运行单位，因此里面存储的信息都是跟当前线程（或程序）相关信息的。包括局部变量、程序运行状态、方法返回值等等

栈中存的是基本数据类型和堆中对象的引用。

在栈中，一个对象只对应了一个4btye的引用（堆栈分离的好处：））。

堆

堆是存储的单位

堆解决的是数据存储的问题，即数据怎么放、放在哪儿

堆是所有线程共享的

堆只负责存储对象信息

堆中存的是对象

一个对象的大小是不可估计的，或者说是可以动态变化的

- 为什么要把堆和栈区分出来呢？栈中不是也可以存储数据吗？
- 为什么不把基本类型放堆中呢？

Java对象的大小

基本数据的类型的大小是固定的

非基本类型的Java对象

- 在Java中，一个空Object对象的大小是8byte，这个大小只是保存堆中一个没有任何属性的对象的大小

  在栈中，一个对象只对应了一个4btye的引用（堆栈分离的好处：）

如何分代

年轻代

Eden区

Survivor区

Survivor区

年老代

在年轻代经历了n次垃圾回收仍然存活的对象将被复制到老年代。因此可以认为，老年代存放的对象都是生命周期较长的对象。

持久代

用于存放静态问题，如java类和方法，常量池。持久代对垃圾回收没有显著影响。

GC

- Scavenge GC

  Full GC

  对整个堆进行整理，包括新生代，老年代和持久代。Full GC要对整个堆进行整理，因此要比Scvaenge GC慢，因此应该尽可能的减少Full GC 的次数。在对JVM调优的工作中，很大一部分就是对Full GC的调节。

垃圾回收算法

按照基本回收策略分

- 引用计数（Reference Counting）

- 复制（Copying）
- 标记-整理（Mark-Compact）

按分区对待的方式分

- 增量收集（Incremental Collecting）
- 分代收集（Generational Collecting）

按系统线程分

- 串行收集
- 并行收集
- 并发收集