# JVM

## 虚拟机信息

|  |
| --- |
| java -version  java version "1.8.0\_181"  Java(TM) SE Runtime Environment (build 1.8.0\_181-b13)  Java **HotSpot**(TM) 64-Bit **Server** VM (build 25.181-b13, mixed mode)  Java虚拟机其实也有client 和Server之分:（只是启动的形式和参数不一样）   * client :主要针对c/s架构的桌面级应用，可能内存占用不多，所以分配的一些内存大小比较小（如：堆的新生代较小） * Server:针对服务器应用，如b/s架构应用   修改jvm的安装client或server形式启动如下：/jre/lib/i386或amd64...下jvm.cfg文件：    如上：第一行是server则默认启动server形式的虚拟机。 |

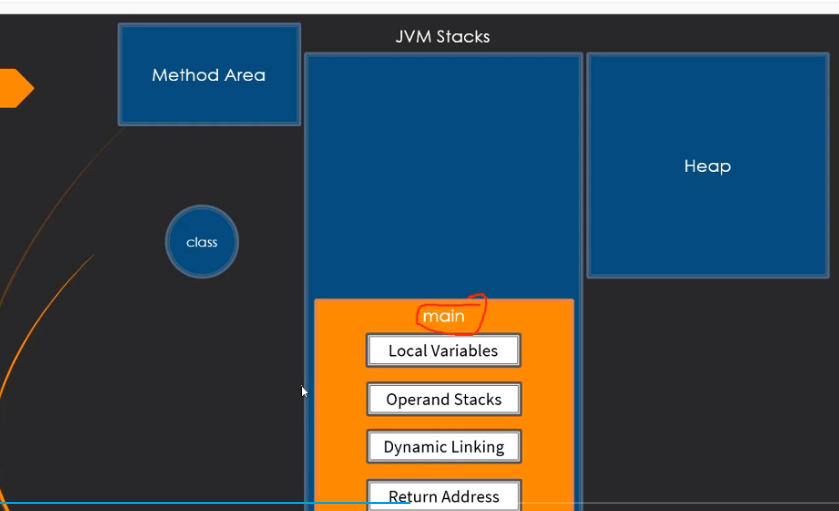
## JVM总体结构

|  |
| --- |
| * **类加载子系统**将.class文件从文件系统/网络中加载（Classloader就为其中一模块）   信息存放在**方法区内存空间**中。方法区中有类信息/常量池   * **Java堆**：会在jvm启动时建立，几乎所有的对象都在堆中存储 * **直接内存**：不受jvm限制，取决于物理内存大小，堆外内存。速度优于java堆。（NIO多使用直接内存） * **垃圾回收系统**：回收java堆/方法区/直接内存 * **Java栈：**虚拟机栈中存储**栈帧**（frame）,而每个方法对应一个**栈帧。** * **本地方法栈：**本地方法（c/c++）方法执行时用到的内存。 * **Pc寄存器：**当前线程真正执行的指令地址**。** * **执行引擎：**执行虚拟机的字节码 |

## JVM内存结构

|  |
| --- |
|  |

## 虚拟机栈



|  |
| --- |
| 虚拟机栈中存储**栈帧**（frame）,而每个方法对应一个**栈帧。** |

## Java堆结构

|  |
| --- |
| 堆内存是jvm内存中最大的一块，也是垃圾回收最频繁的一块区域。  给堆**分代（分区域），**来提高对象内存分配和垃圾回收的效率。如果不分代，垃圾回收后造成内存碎片化严重，可能导致内存浪费（**被回收的内存片不够大对象存储**！）     * 当一个对象被创建后，如果对象比较大会直接被放在老年代 ，否则会被放在新生代伊甸区 （回收率达到70%-95%） * 在新生代中的对象经过一次gc后没有被回收，该对象会被存放到一个幸存区如：s0，再次gc后，s0中还没被回收会被放到s1中并把伊甸区中未被回收的对象放到s1中，再次gc后，s1中对象会被放到s0中并把伊甸区未被回收的对象放入，两个幸存区来回拷贝。（两个幸存区互相拷贝效率很高） * 经过多次gc后幸存区中还没被回收的对象会被放入老年代   新生代中 伊甸区，幸存区s0,幸存区s1 比例为：8:1:1  老年代中的gc频率比较低，回收速度比较慢。  Jdk1.7之后把永久代去掉了。 |

## 垃圾回收

|  |
| --- |
| 垃圾：没有被引用的对象，以及互相引用或者环形引用如下图：    **如何确定垃圾**：1.引用计数：会有上图循环引用的问题  2.正向可达：从根部顺藤摸瓜，不可以摸到的为垃圾  **垃圾回收算法**：   1. Mark-Sweep标记清除（第一阶段：使用根搜索算法将活跃对象标记，第二阶段：遍历整个堆，将没有标记的回收），缺点内存碎片化了，（**会暂停应用**）会FUll GC      1. Coyping复制:（新生代幸存区使用）效率高但会内存浪费，它会将内存分成两份进行互相复制。存活的对象比较少，占用内存较少，效率也高。   IMG_256   1. Mark-compact标记压缩：标记-清除算法的优化，第一阶段：标记活跃对象，第二阶段：清除未标记对象，并将活跃对象压缩到一块（即连续存放）。解决了内存碎片问题。   **注意：**  从年轻代空间(包括Eden和 Survivor 区域)回收内存被称为 **Minor GC**,  对老年代GC称为**Major GC**,而**Full GC**是对整个堆来说的。FullGC耗时较长，应该避免。 |