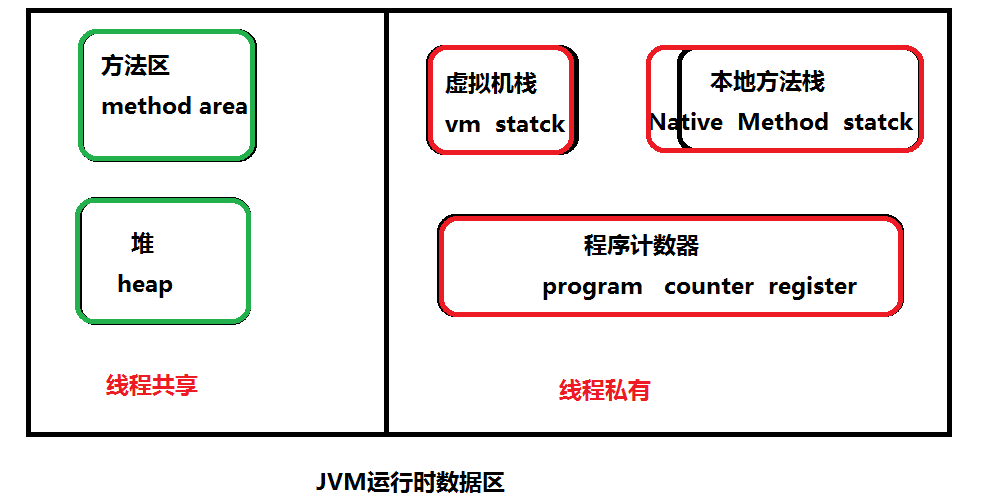
一：运行时数据区

1.在运行时数据区那些是线程共享和线程私有的?



### 01.线程共享:

堆，方法区

### 02.线程私有:

虚拟机栈，本地方法栈，程序计数器

# 程序计数器:

当前线程执行字节码文件的行号指示器;

能够在线程切换之后，找到自己正确的位置;

是唯一一个不会导致内存泄漏(OutOfMemor,00M)的区域。

虚拟机栈:

基本数据类型的变量和对象的引用变量都存在此区域;

遵循“先进后出，后进先出”的原则

##### StackOverflowError:

线程请求的栈深度大于了虚拟机栈规定的最大深度!

##### OutOfMemoryError

虚拟机在扩展栈时，无法申请到足够的内存空间!

本地方法栈:

主要服务的是Native修饰的方法

（Native：系统的 java是一个不完全的语言无法直接调用系统）

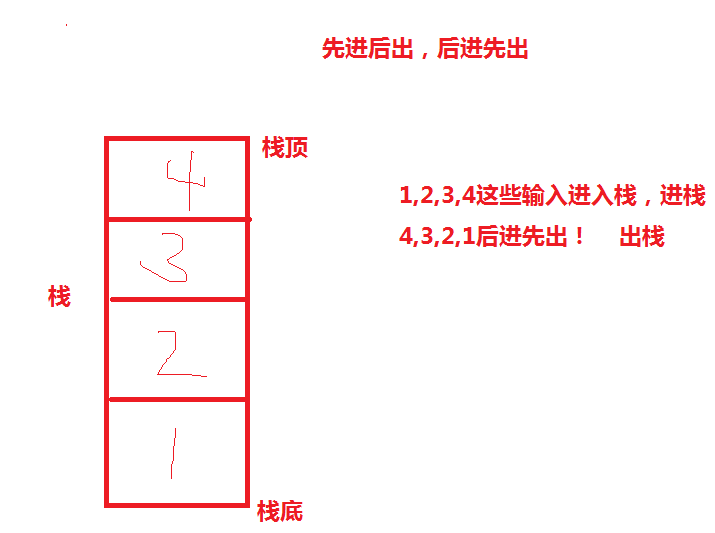
在HotSport中虚以机栈和本地方法栈合为一体!

堆:

所有对象的实例和数组开辟的空间都在此区域保存!

所有线程共享!

GC主要回收对象的区域!



#### 堆分区:

新生代

老年代

永久代

#### 方法区: (静态区)

所有线程共享;

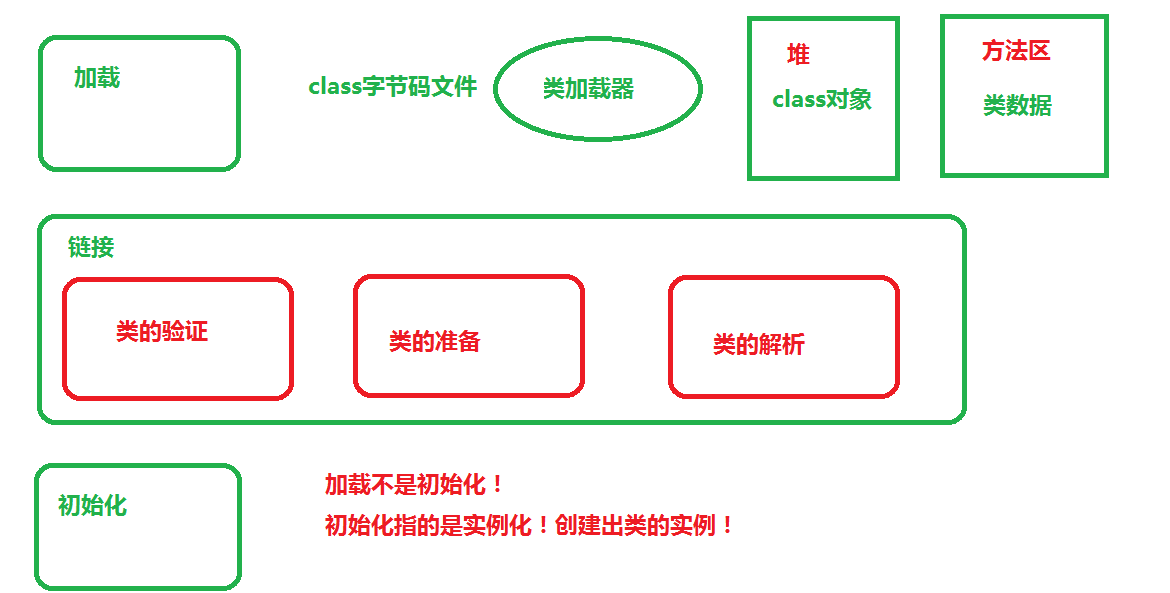
方法(包含构造函数)，接口 定义在此区域;

所有方法的信息;

存放静态变量+常量+类信息+方法信息+常量池!

常量池中存放程序在编译期间生成的各种变量以及符号引用!

二:类的生命周期



类的生命周期从类被加载，连接和初始化开始!到类的卸载结束!

01.类的生命周期中，类的2进制数据位于方法区;

02.在堆中会有一个描述这个类的Class对象;

## 2.1加载: 需要类加载器

将class字节码文件内容加载到内存中，并将这些静态数据转化成方法区中运行时数据结构!在堆中生成一个Class对象;

这个Class对象就是方法区类数据的访问入口!

## 2.2链接:

01.类的验证 在编译期间

合法？？？

001类文件的结构检查

002.语义检查 比如说final修饰的变量在编译期间发现再次赋值

003.字节码验证，确保字节码能被jvm识别

02.类的准备

为类中所有static修饰的内容开辟空间，并且赋予初始值! ! !

static int num =100;

这时候num=0;

03.类的解析

把类中所有的符号引用转换成直接引用!

符号引用: 就是用字符串的形式来表示某个变量或者是类，能让我们看懂

String a="a"; Student stu=new Student();

a, stu都是符号引用!

直接引用:是根据符号引用翻译出来的地址!

## 2.3 初始化

加载不是初始化!

初始化指的是实例化!

创建出类的实例!

初始化的时机:

01.类的主动引用

001. new一个类的对象

002.通过反射的newInstance（）

003.再初始化子类的时候必须先初始化父类

02.类的被动引用

001.通过类名访问静态的内容

002.调用类的静态常量也不会初始化类

003.用类作为对象数组存在时，也不会初始化类

004.子类调用父类的静态变量不会加载子类的静态代码块(不会执行类的准备阶段)

三．垃圾回收机制（GC）

3.1 为什么需要垃圾回收机制

01.只要是对象创建，那么就会在虚拟机的堆中开辟空间；

02.程序运行过程中会创建n个对象，每个对象都会有自己的空间；

03.如果每个对象都永久的占有这块内存空间，显然内存是不够的

3.2

3.3