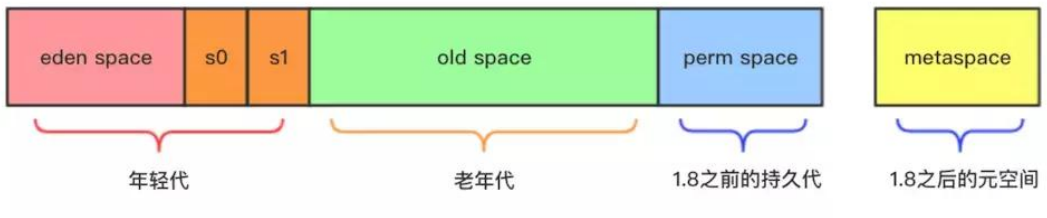
## jvm知识

**笔记本:** 我的笔记
**创建时间:** 2019/2/26 9:32 **更新时间:** 2019/3/1 9:42
**作者:** liuhouer
**标签:** jvm

---

# 1.G1之前的JVM内存模型




新生代：伊甸园区(eden space) + 2个幸存区

老年代

持久代(perm space)：JDK1.8之前

元空间(metaspace)：JDK1.8之后取代持久代

## Java内存模型中堆和栈的区别

➢管理方式:栈自动释放,堆需要GC
➢空间大小:栈比堆小
➢碎片相关:栈产生的碎片远小于堆
➢分配方式:栈支持静态和动态分配,而堆仅支持动态分配
➢效率:栈的效率比堆高

## JVM三大性能调优参数Xms -Xmx - Xss的含义

➢-Xss:规定了每个线程虚拟机栈(堆栈)的大小:
➢-Xms :堆的初始值
➢-Xmx :堆能达到的最大值

## Java内存模型中堆和栈的区别一-内存分配策略

>静恣存偌:编译时确定每个数据目禄在返行吋的存储空向需求
>栈式存侈:数据区需求在编鋒咐未知,迄行吋模抉入口前碉定
>堆式存儲:编译时或运行时模抉入口都无法确定,动态分配

## 不同JDK版本之间的intern()方法的区别一JDK6 VS JDK6+

```
  String s = new String( original: "a");
s. intern();
```

JDK6:当调用intern 方法时，如果**字符串常量池**先前已创建出该字符串对象，则返回池中的该字符串的引用。否则，将此字符串对象**添加到字符串常量池中**，并且返回该字符串对象的引用。

JDK6+:当调用intern 方法时，如果字符串常量池先前已创建出该字符串对象，则返回池中的该字符串的引用。否则，如果**该字符串对象已经存在于Java堆中**,则将堆中对**此对象的引用添加到字符串常量池中**，并且返回该引用:如果堆中不存在，则在池中创建该字符串并返回其引用。

## jdk6不断调用intern()撑爆永久代

```
public class PermGenErrTest {
    public static void main(String[] args) {
        for(int i=0; i <= 1000; i++){
            //将返回的随机字符串添加到字符串常量池中
            getRandomString(1000000).intern();
        }
        System.out.println("Mission Complete!");
    }

    //返回指定长度的随机字符串
    private static String getRandomString(int length) {
        //字符串源
        String
str="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        Random random = new Random();
        StringBuffer sb = new StringBuffer();
        for ( int i = 0; i < length; i++){
            int number = random.nextInt(62);
            sb.append(str.charAt(number));
        }
        return sb.toString();
    }
}
```

### jdk6 （false，false） jdk7或以上（false，true）

```
public class InternDifference {
    public static void main(String[] args) {
        String s = new String("a");  //堆
        s.intern(); //添加到常量池
        String s2 = "a";
        System.out.println(s == s2);  jdk6 堆！=常量池的地址

        String s3 = new String("a") + new String("a");
        s3.intern();
        String s4 = "aa";
        System.out.println(s3 == s4); jdk6 堆！=常量池的地址
    }
}
```

**类加载的过程。**

当系统主动使用某个类，如果该类还未加载到内存中，系统会加载、连接、初始化三个步骤。
1.类的加载将类的Class文件读入内存中，并为之创建一个java.lang.Class对象。
Class文件的来源：
(1)从本地加载class文件
(2)从jar包中加载(系统api)
(3)从网络加载
2.类的连接：
连接阶段负责将类的二进制数据合并到JRE中。
3.类的初始化
类的初始化时机：
(1)创建类的实例：new操作符、反射创建实例、通过反序列化；
(2)调用某个类的静态方法；
(3)访问某个类的静态属性（final属性除外）。

**双亲委派模型。**

双亲委派模型工作过程是：如果一个类加载器收到类加载的请求，它首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器完成。每个类加载器都是如此，只有当父加载器在自己的搜索范围内找不到指定的类时（即ClassNotFoundException），子加载器才会尝试自己去加载。

**有哪些类加载器。**

```
BootstrapLoader：      sun.boot.class.path
ExtClassLoader:            java.ext.dirs
AppClassLoader:            java.class.path
```

**能不能自己写一个类叫java.lang.String。**

可以