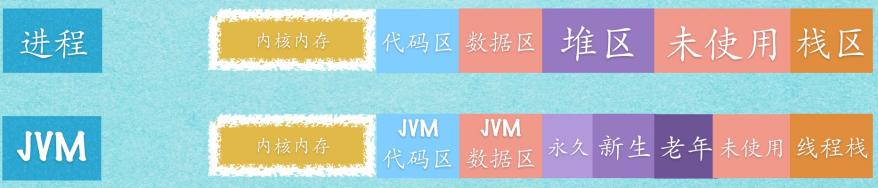


1. JVM内存结构与LINUX系统进程内存结构对比

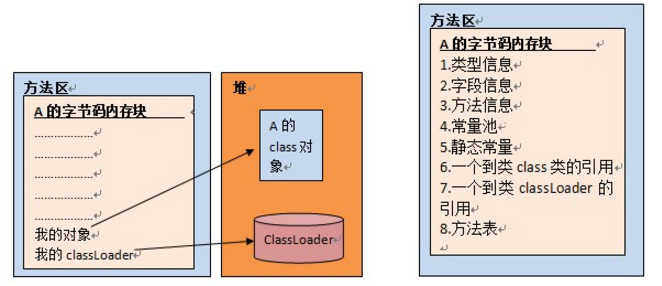


（1）JVM就是一个程序，用C语言实现JVM，内部已经实现完成各种操作（申请空间、垃圾回收等）。

（2）.class作为JVM的输入，然后JVM程序运行.class文件；根据字节码文件在c语言中创建结构体、申请空间、创建函数、创建线程等等。

（3）JVM程序只认识.class字节码文件，不认识JAVA、JRuby、Groovy等高级语言。因此，JVM与JAVA并没有什么关系，JAVA只不过给JVM提供.class文件的一种方式。

1. 方法区的实现



主要两类信息：字符串常量池与类信息

1. JDK8之前：方法区放在永久代（JVM申请的一块空间专门用于方法区）
2. JDK8后：方法区（类信息）放在元空间即本地内存中；但字符串常量 池放到JVM堆中
3. 为什么要方法区要改用元空间？
4. 字符串常量池放在永久代，容易导致内存溢出与性能问题
5. 类的信息用难以确定，因此指定永久代的大小比较困难，太小容易导致永久代溢出，太大容易导致老年代溢出，并且永久代的GC回收效率极低。
6. 运行时常量池

（1）常量池：存储的时字符串的引用（指针）

（2）如何判断运行时常量池是属于方法区还是堆？

设置永久代大小:-XX:PermSize=6M（初始值）

-XX:MaxPermSize=6M（最大值）

不断往常量池入字符串(String.intern())

看异常

1. 入池String.intern()：返回的是常量池记录的引用

new 字符串是在堆中

1. JDK8之前:会把字符串内容拷贝到永久代，然后返回的是永久代里 面的引用
2. JDK8后:因为都在堆上（字符串、常量池），所以返回首次出现的实例引用（若常量池有则直接返回；否则，看常量池外有创建则返回常量池外的实例引用，若没找到则创建并返回）。
3. 为什么JVM可以通过参数指定堆区、栈区大小等？

因为JVM本身就是用C语言实现的，C语言可以通过malloc申请内存，JVM的堆区大小就可以通过malloc指定，所以只要通过参数（main函数通过cmd可以传入参数），就可以指定JVM要申请多大的堆区。

1. JVM的堆区与直接内存区别？

JVM的堆区：事先malloc一块很大的空间，已经是JVM申请好的固定大小的堆区。

直接内存：相当于直接调用C语言的malloc，不是JVM事先申请的，申请空间在图中的未使用区。

1. 测试JVM堆内存溢出

（1）设置JVM堆内存大小:

-Xms20m（memory min）

-Xmx20m（memory max）

异常：OOM（OutOfMemory）,不断申请内存导致

1. 堆内存溢出的快照：

-XX:+HeapDumpOnOutOfMemoryError （堆内存溢出时生成快照）

-XX:HeapDumpPath=/JVM（堆内存溢出快照文件生成后存放的路径）

1. 代码

public class HeapOOM

{

public static void main(String[] args)

{

int cnt=0;

List<Object> list=new ArrayList<Object>();

while(true)

{

list.add(new Object());

cnt++;

System.out.println(cnt);

}

}

}

1. 测试栈（虚拟机栈与本地方法栈）溢出

（1）设置每个线程栈固定大小（JVM不区分虚拟机栈与本地方法栈）:

-Xss128k（stack small）

异常：SOF（StackOverFlowError），线程中不断调用方法或者声明很多变量导致。

1. 代码

-Xss128k

public class StackOverFlow

{

public int length=0;

public void stackLeak()

{

System.out.println(length++);

stackLeak();

}

public static void main(String[] args)

{

StackOverFlow stackOverFlow=new StackOverFlow();

stackOverFlow.stackLeak();

}

}

1. 方法区溢出

元空间是在本地内存，已经很难使得方法区溢出，不过HotSpot还是提供一些元空间的参数

设置元空间大小：

-XX:MetaSpaceSize（初始值）

-XX:MaxMetaSpaceSize（最大值）

-XX:MinMetaSpaceFreeRation

（控制元空间在垃圾回收后剩余容量百分比，以减少垃圾回收频率）

1. 测试直接内存溢出
2. 设置直接内存大小

-XX:MaxDirectMemorySize=10M

1. 代码

private static final int \_1MB=1024\*1024;

public static void main(String[] args) throws Exception

{

Field unsafeField= Unsafe.class.getDeclaredFields()[0];

unsafeField.setAccessible(true);

Unsafe unsafe=(Unsafe) unsafeField.get(null);

while(true)

{

unsafe.allocateMemory(\_1MB);

}

}