

程序计数器

每条线程都需要有一个独立的程序计数器，各条线程

之间计数器互不影响，独立存储，我们称这类内存区域为“线程私有”的内存。

Java虚拟机栈

而“栈”通常就是指这里讲的虚拟机栈，或

者更多的情况下只是指虚拟机栈中局部变量表部分。

虚拟机栈 局部变量表

如果线程请求的栈深度大于虚

拟机所允许的深度，将抛出StackOverflowError异常；如果Java虚拟机栈容量可以动态扩展[2]，当栈扩

展时无法申请到足够的内存会抛出OutOfMemoryError异常。

深度是 StackOverflow

空间不足是outofmemoryerror

堆 数组，对象实例

HotSpot里面也出

现了不采用分代设计的新垃圾收集器，再按照上面的提法就有很多需要商榷的地方了。

根据《Java虚拟机规范》的规定，Java堆可以处于物理上不连续的内存空间中，但在逻辑上它应该

被视为连续的，

2.3.1 对象的创建

当Java虚拟机遇到一条字节码new指令时，首先将去检查这个指令的参数是否能在常量池中定位到

一个类的符号引用，并且检查这个符号引用代表的类是否已被加载、解析和初始化过。如果没有，那

必须先执行相应的类加载过程，

指针碰撞和空闲列表

Serial、ParNew 压缩整理过程的收集器 指针碰撞 Bump The Pointer）

CMS基于sweep算法 空闲列表来分配内存。 free list

实际上虚拟机是采用CAS配上失败

重试的方式保证更新操作的原子性；另外一种是把内存分配的动作按照线程划分在不同的空间之中进

行，即每个线程在Java堆中预先分配一小块内存，称为本地线程分配缓冲（Thread Local Allocation

Buffer，TLAB），

2.4.3 方法区和运行时常量池溢出

public class StringTest {  
 public static void main(String[] args) {  
 // 使用 "" 创建 会直接存储在 常量池中  
 String a = "lantao";  
 // 使用 new String 创建， 会将zahngsan存储到常量池中，然后在Heap中创建对象指向b  
 String b = new String("zhangsan");  
 // 使用 字符串连接符拼接 ，会直接存储 'wangwuzhaoliu' 字符串在常量池中  
 String c = "wangwu" + "zhaoliu";  
 // 使用字符串 "引用" 拼接 ，不执行 intern 方法，不会存放到常量池中，但是会将 --- 存入到常量池中  
 String d = a + "---";  
 // 使用 new String 拼接 ，不执行 intern 方法，不会存放到常量池中，但是会将wang 和 jiu 两个字符串存到常量池中  
 String f = new String("wang") + "jiu";  
 // 使用 new String 拼接 ，不执行 intern 方法，不会存放到常量池中， 但是会将 zhao 和 ba 两个字符串存入到常量池中  
 String g = new String("zhao") + new String("ba");  
 }  
}

public class StringTest {  
 public static void main(String[] args) {  
 // 基本数据类型之间的 == 是比较值，引用数据类型 == 比较的是地址值  
 // 1：在Java Heap中创建对象 2：在字符串常量池中添加 zhangsan  
 String a = new String("zhangsan");  
 // 调用 intern 方法，因上一步中已经将zhangsan存入常量池中，这里直接返回常量池 zhangsan 的引用地址  
 String b = a.intern();  
 // a 的地址在Java Heap中 ， b的地址在 常量池中 ，所以结果是flase  
 System.*out*.println(a == b);  
 // 因为常量池中已经包含zhangsan，所以直接返回  
 String c = "zhangsan";  
 // b c 的地址一致，所以是true  
 System.*out*.println(b == c);  
 }  
}  
  
//结果  
false  
 true

Java 7以后常量池移到java堆之中

在JDK 8以后，永久代便完全退出了历史舞台，元空间作为其替代者登场。在默认设置下，前面

列举的那些正常的动态创建新类型的测试用例已经很难再迫使虚拟机产生方法区的溢出异常了。不过

为了让使用者有预防实际应用里出现类似于代码清单2-9那样的破坏性的操作，HotSpot还是提供了一

些参数作为元空间的防御措施，主要包括

第3章 垃圾收集器与内存分配策略