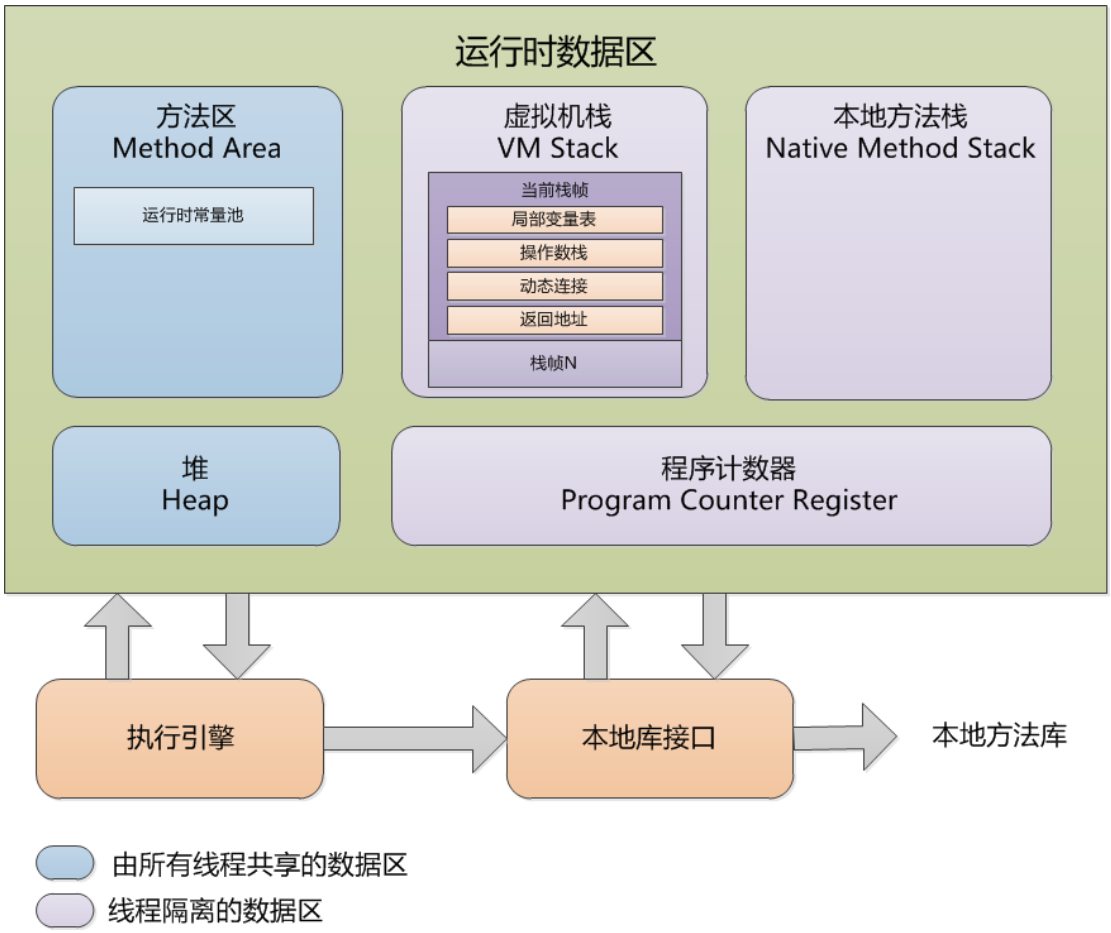
Java 内存管理机制

Java 运行时数据区：

Java 虚拟机在执行 Java 程序的过程中会把它所管理的内存划分为若干个不同的数据区域，这些区域都拥有自己的用途，并随着 JVM 进程的启动或者用户线程的启动和结束建立和销毁。 Java SE7 版中将虚拟机管理的内存划分为以下几个运行时数据区域。



程序计数器（Program Counter Register）

线程私有内存

可以看成是当前线程所执行字节码的行号指示器。

在虚拟机的概念模型中，字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令的。

此内存区域是唯一一个在 Java 虚拟机规范中没有规定任何 OutOfMemoryError 情况的区域。

Java 虚拟机栈（Java Virtual Machine Stacks）

线程私有 生命周期与线程同步

虚拟机栈描述的是 Java 方法执行的内存模型：每个方法在执行的时候都会创建一个栈帧用于存储 局部变量表、操作数栈、动态链接、方法出口等信息。 每一个方法从调用到执行完成就对应着一个栈帧在虚拟机中入栈到出栈的过程。

压栈出栈过程

当方法运行过程中需要创建局部变量时，就将局部变量的值存入栈帧中的局部变量表中。

Java 虚拟机栈的栈顶的栈帧是当前正在执行的活动栈，也就是当前正在执行的方法，PC 寄存器也会指向这个地址。只有这个活动的栈帧的本地变量可以被操作数栈使用，当在这个栈帧中调用另一个方法，与之对应的栈帧又会被创建，新创建的栈帧压入栈顶，变为当前的活动栈帧。

方法结束后，当前栈帧被移出，栈帧的返回值变成新的活动栈帧中操作数栈的一个操作数。如果没有返回值，那么新的活动栈帧中操作数栈的操作数没有变

化。
局部变量表随着栈帧的创建而创建，它的大小在编译时确定，创建时只需分配事先规定的大小即可。在方法运行过程中，局部变量表的大小不会发生改变。
Java 虚拟机栈会出现两种异常：StackOverFlowError 和 OutOfMemoryError。
StackOverFlowError： 若 Java 虚拟机栈的大小不允许动态扩展，那么当线程请求栈的深度超过当前 Java 虚拟机栈的最大深度时，抛出 StackOverFlowError 异常。
OutOfMemoryError： 若允许动态扩展，那么当线程请求栈时内存用完了，无法再动态扩展时，抛出 OutOfMemoryError 异常。

本地方法栈（Native Method Stack）
与虚拟机栈作用相似。 不同之处： 虚拟机栈是为虚拟机执行 Java 方法服务，本地方法去是为了虚拟机使用到 Native 方法服务的。 HotSpot 在 SE7 的版本中将虚拟机和本地方法栈合二为一。同会抛出 OutOfMemoryError 和 StackOverFlowError 异常。

Java 堆 （Java Heap）
Java 虚拟机管理内存最大的一块。堆是用来存放对象的内存空间，几乎所有的对象都存储在堆中。
堆的特点
线程共享，整个 Java 虚拟机只有一个堆，所有的线程都访问同一个堆。而程序计数器、Java 虚拟机栈、本地方法栈都是一个线程对应一个。
在虚拟机启动时创建。
是垃圾回收的主要场所。也称为 “GC 堆”
为了更好地回收与分配内存，堆进一步可分为：
内存回收的角度：新生代(Eden 区 From Survior To Survivor)、老年代。
内存分配角度：可以分配出多个线程私有的分配缓冲区域 Thread Local Allocation Buffer （TLAB）
参数： -Xmx -Xms

方法区：(Method Area)
线程共享区域
存储被虚拟机加载的类信息、常量、静态变量、JIT 编译后的代码等数据。堆的

一个逻辑部分。

运行时常量池：（Runtime Constant Pool）

方法区的一部分， class 文件中会包含类的版本、字段、方法、接口等描述信息，还有常量池（Constant Pool Table）,用于存放编译期生成的各种字面量和符号引用。这部分内容会存放在运行时常量池当中。

Demo：

```
package com.kenshin.chapter_2_memory;

public class RuntimeConstantPoolTest {
    public static void main(String[] args) {
        String st1 = "hello word!"; //字节码常量
        String st2 = "hello word!";
        String st3 = new String( original: "hello word");

        System.out.println(st1 == st2);
        System.out.println(st1 == st3);
        System.out.println(st1 == st3.intern());  //运行时常量
    }
}
```

```
true
false
false

Process finished with exit code 0
```

会抛出 OOM 异常。

直接内存（Direct Memory）

直接内存是除 Java 虚拟机之外的内存，但也可能被 Java 使用。

操作直接内存

在 NIO 中引入了一种基于通道和缓冲的 IO 方式。它可以通过调用本地方法直接分配 Java 虚拟机之外的内存，然后通过一个存储在堆中的 DirectByteBuffer 对象直接操作该内存，而无须先将外部内存中的数据复制到堆中再进行操作，从而提高了数据操作的效率。

OutOfMemoryError 异常：

1. Java 堆内存溢出异常

```java
/**
 *  VM options: -Xms20m -Xmx20m -XX:+HeapDumpOnOutOfMemoryError
 */
package com.kenshin.chapter_2_memory;

import java.util.ArrayList;
import java.util.List;

public class JavaHeapOOMTest {

    static class HeapObject{

    }

    public static void main(String[] args) {
        List<HeapObject> objectList = new ArrayList<>();

        while (true) {
            objectList.add(new HeapObject());
        }
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home/bin/java -Xms20m -Xmx20m -XX:+HeapDumpOnOutOfMemoryError
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid2786.hprof ...
Heap dump file created [29199265 bytes in 0.182 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.util.Arrays.copyOf(Arrays.java:3719)
    at java.base/java.util.Arrays.copyOf(Arrays.java:3688)
    at java.base/java.util.ArrayList.grow(ArrayList.java:237)
    at java.base/java.util.ArrayList.grow(ArrayList.java:242)
    at java.base/java.util.ArrayList.add(ArrayList.java:467)
    at java.base/java.util.ArrayList.add(ArrayList.java:480)
    at com.kenshin.chapter_2_memory.JavaHeapOOMTest.main(JavaHeapOOMTest.java:16)

Process finished with exit code 1
```

2. **虚拟机栈和本地方法栈溢出**。

   StackOverFlow 异常： 线程请求的栈深度大于虚拟机允许的最大深度。

   OutOfMemoryError 异常：虚拟机在扩展栈时无法申请到足够的内存看空间。

```java
/**
 * VM options: -Xss144k
 */
public class JavaVMStackSOFTest {
    private int stackLength = 1;

    public void stackLeak() {
        stackLength++;
        stackLeak();
    }

    public static void main(String[] args) {
        JavaVMStackSOFTest javaVMStackSOFTest = new JavaVMStackSOFTest();
        try {
            javaVMStackSOFTest.stackLeak();
        } catch (Throwable e) {
            System.out.println("stack Length: " + javaVMStackSOFTest.stackLength);
            throw e;
        }
    }
}
```

```
stack Length: 498
Exception in thread "main" java.lang.StackOverflowError
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:10)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
	at com.kenshin.chapter_2_memory.JavaVMStackSOFTest.stackLeak(JavaVMStackSOFTest.java:11)
```

OutOfMemoryError 异常

```java
package com.kenshin.chapter_2_memory;

public class JavaVMStackOOMTest {
    private int threadCount = 1;

    private void dontStop() {
        while (true) {

        }
    }

    public static void main(String[] args) {
        JavaVMStackOOMTest javaVMStackOOMTest = new JavaVMStackOOMTest();

        while (true) {
            javaVMStackOOMTest.threadCount++;
            new Thread(new Runnable() {
                @Override
                public void run() {
                    System.out.println("This is No. " + javaVMStackOOMTest.threadCount + " Thread.");
                    javaVMStackOOMTest.dontStop();
                }
            }).start();
        }
    }
}
```

电脑卡死了 OOM 异常没出现

3. 运行时常量池

```java
/**
 * --XX: PermSize=10M -XX:MaxPermSize=10M
 */

package com.kenshin.chapter_2_memory;

import java.util.ArrayList;
import java.util.List;

public class RuntimeConstantPoolOOM {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        int i = 0;
        while (true) {
            list.add(String.valueOf(i++).intern());
        }
    }
}
```

JDK < 1.7　OOM 异常

JDK > 1.7　长时间没有出现异常

4. 方法区异常

```java
package com.kenshin.chapter_2_memory;

import net.sf.cglib.proxy.Enhancer;
import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

import java.lang.reflect.Method;

public class MethodAreaOOM {
    public static void main(String[] args) {
        while (true) {
            Enhancer enhancer = new Enhancer();
            enhancer.setSuperclass(OOMObject.class);
            enhancer.setUseCache(false);
            enhancer.setCallback(new MethodInterceptor() {
                @Override
                public Object intercept(Object o, Method method, Object[] objects, MethodProxy methodProxy) throws Throwable {
                    return methodProxy.invokeSuper(o, objects);
                }
            });
            enhancer.create();
        }
    }

    static class OOMObject {

    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home/bin/java -XX:MetaspaceSize=10m -XX:MaxMetaspaceSize=10m "-javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by net.sf.cglib.core.ReflectUtils$2 (file:/Users/shentao/Downloads/cglib-nodep-3.1.jar) to method java.lang.ClassLoader.defineClass(java.lang.String
WARNING: Please consider reporting this to the maintainers of net.sf.cglib.core.ReflectUtils$2
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Exception in thread "main" java.lang.OutOfMemoryError: Metaspace

Process finished with exit code 1
```

5. 直接内存异常

```java
/**
 * -Xmx20M -XX:MaxDirectMemorySize=10M
 */
package com.kenshin.chapter_2_memory;

import sun.misc.Unsafe;

import java.lang.reflect.Field;

public class DirectMemoryOOM {
    private static final int _1MB = 1024 * 1024;

    public static void main(String[] args) throws IllegalAccessException {
        Field unsafeField = Unsafe.class.getDeclaredFields()[0];
        unsafeField.setAccessible(true);
        Unsafe unsafe = (Unsafe) unsafeField.get(null);
        int i = 0;
        while (true) {
            unsafe.allocateMemory(_1MB);
            i++;
            System.out.println("allocate: " + i + "MB");
        }

    }
}
```