> 主要讲述Java运行时的时区，包括Java堆、虚拟机栈、本地方法栈、方法区和程序计数器相关内容。

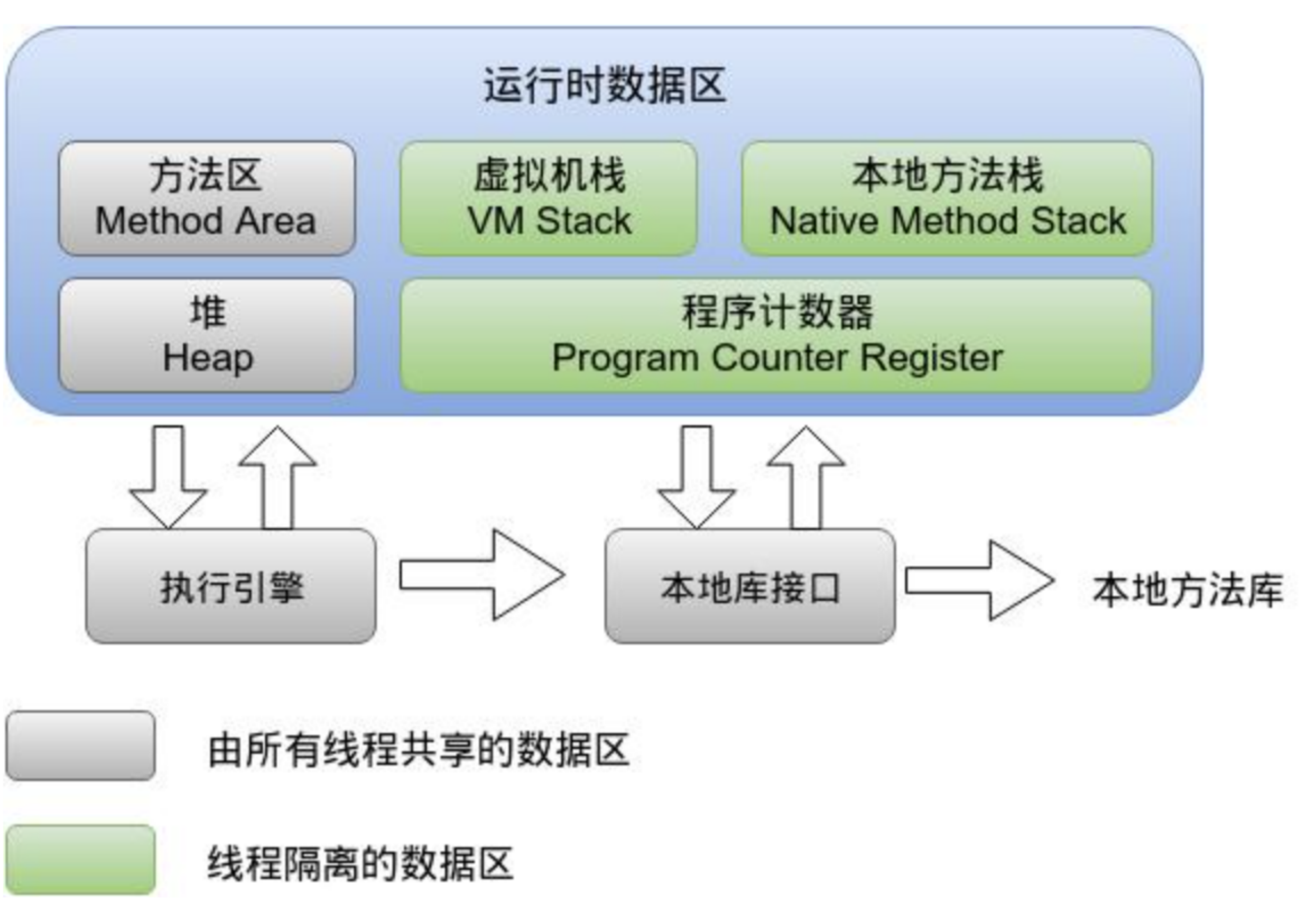
# 前言

上一篇文章《【JVM系列1】JVM内存结构》已经讲述了JVM的内存结构，其实这个讲解的并不全，只讲解了Java堆的部分，现在将其它部分也进行补充。

# 运行时数据区域

什么是运行时数据区域?

Java程序在运行时，会为JVM单独划出一块内存区域，而这块内存区域又可以再次划分出一块运行时数据区，运行时数据区域大致可以分为五个部分：



## Java堆（Heap）

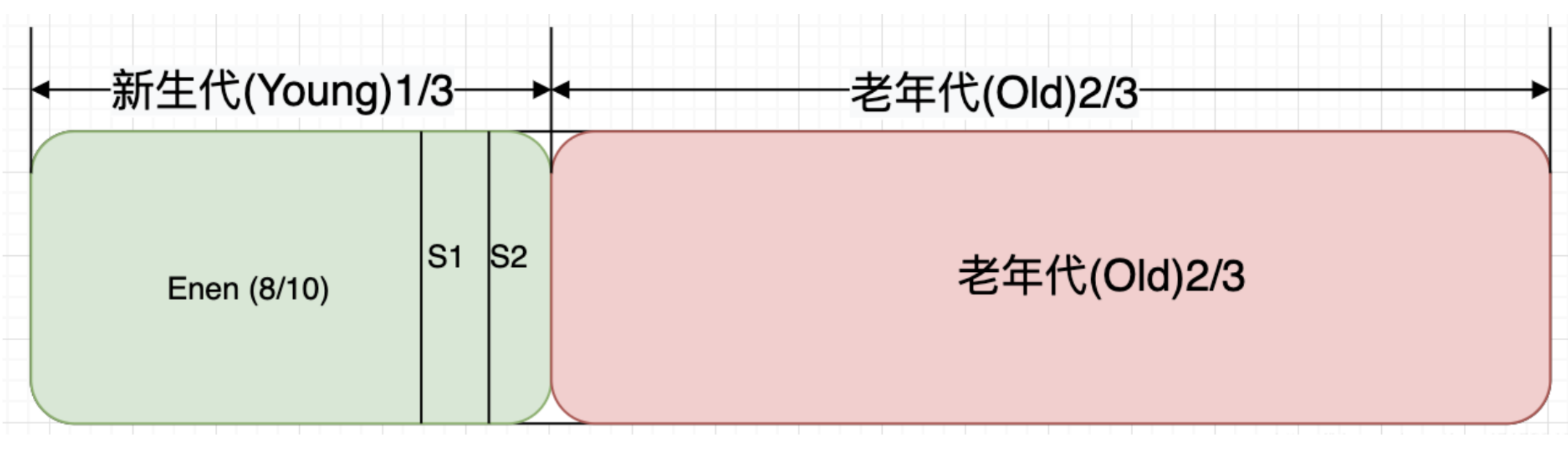
很多做开发的同学，会格外关注堆和栈，这是不是就从另一个角度说明了堆和栈的重要性?

> 一句话便是：栈管运行，堆管存储。则虚拟机栈负责运行代码，而虚拟机堆负责存储数据。

先把干货放上来，首先，Java堆区具有下面几个特点：

- 存储的是我们new来的对象，不存放基本类型和对象引用。
- 由于创建了大量的对象，垃圾回收器主要工作在这块区域。
- 线程共享区域，因此是线程不安全的。
- 能够发生OutOfMemoryError。

其实，Java堆区还可以划分为新生代和老年代，新生代又可以进一步划分为Eden区、Survivor 1区、Survivor 2区。具体比例参数的话，可以看一下这张图。



> 具体可以参考《【JVM系列1】JVM内存结构》

## 虚拟机栈（JVM Stacks）

Java虚拟机栈也是一块被开发者重点关注的地方，同样，先把干货放上来：

- Java虚拟机栈是线程私有的，每一个线程都有独享一个虚拟机栈，它的生命周期与线程相同。
- 虚拟机栈描述的是Java方法执行的内存模型：每个方法被执行的时候都会同时创建一个栈帧（Stack Frame）用于存储局部变量表、操作栈、动态链接、方法出口等信息。每一个方法被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。
- 存放基本数据类型（boolean、byte、char、short、int、float、long、double）以及对象的引用（reference类型，它不等同于对象本身，根据不同的虚拟机实现，它可能是一个指向对象起始地址的引用指针，也可能指向一个代表对象的句柄或者其他与此对象相关的位置）和returnAddress类型（指向了一条字节码指令的地址）。
- 这个区域可能有两种异常：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；如果虚拟机栈可以动态扩展（当前大部分的Java虚拟机都可动态扩展，只不过Java虚拟机规范中也允许固定长度的虚拟机栈），当扩展时无法申请到足够的内存时会抛出OutOfMemoryError异常。

## 本地方法栈（Native Method Stacks）

本地方法栈与虚拟机栈所发挥的作用是非常相似的，其区别不过是虚拟机栈为虚拟机执行Java方法（也就是字节码）服务，而本地方法栈则是为虚拟机使用到的Native方法服务。

虚拟机规范中对本地方法栈中的方法使用的语言、使用方式与数据结构并没有强制规定，因此具体的虚拟机可以自由实现它。甚至有的虚拟机（譬如Sun HotSpot虚拟机）直接就把本地方法栈和虚拟机栈合二为一。与虚拟机栈一样，本地方法栈区域也会抛出StackOverflowError和OutOfMemoryError异常。

> 什么是Native Method？简单地讲，一个Native Method就是一个java调用非java代码的接口。一个Native Method是这样一个java的方法：该方法的实现由非java语言实现，比如C。这个特征并非java所特有，很多其它的编程语言都有这一机制，比如在C＋＋中，你可以用extern "C"告知C＋＋编译器去调用一个C的函数。

## 方法区（Method Area）

方法区也是一块被重点关注的区域，主要特点如下：

- 线程共享区域，因此这是线程不安全的区域。
- 它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。
- 当方法区无法满足内存分配需求时，将抛出OutOfMemoryError异常。

虽然Java虚拟机规范把方法区描述为堆的一个逻辑部分，但是它却有一个别名叫做Non-Heap（非堆），目的应该是与Java堆区分开来。对于习惯在HotSpot虚拟机上开发和部署程序的开发者来说，很多人愿意把方法区称为“永久代”（Permanent Generation），本质上两者并不等价，那么他们之间的区别到底是什么？方法区是Java虚拟机规范中的定义，是一种规范，而永久代是一种实现，一个是标准一个是实现。不过Java 8以后就没有永久代这个说法了，元空间取代了永久代。

Java虚拟机规范对这个区域的限制非常宽松，除了和Java堆一样不需要连续的内存和可以选择固定大小或者可扩展外，还可以选择不实现垃圾收集。相对而言，垃圾收集行为在这个区域是比较少出现的，但并非数据进入了方法区就如永久代的名字一样“永久”存在了。这个区域的内存回收目标主要是针对常量池的回收和对类型的卸载，一般来说这个区域的回收“成绩”比较难以令人满意，尤其是类型的卸载，条件相当苛刻，但是这部分区域的回收确实是有必要的。

## 程序计数器（Program Counter Register）

程序计数器非常简单，想必大家都不是Java的初学者了，也都应该明白一点线程与进程的概念?（灵魂拷问，你明白么？)不明白没关系，我一句话给你讲清楚。

进程是资源分配的最小单位，线程是CPU调度的最小单位，一个进程可以包含多个线程， Java线程通过抢占的方法获得CPU的执行权。现在可以思考下面这个场景。

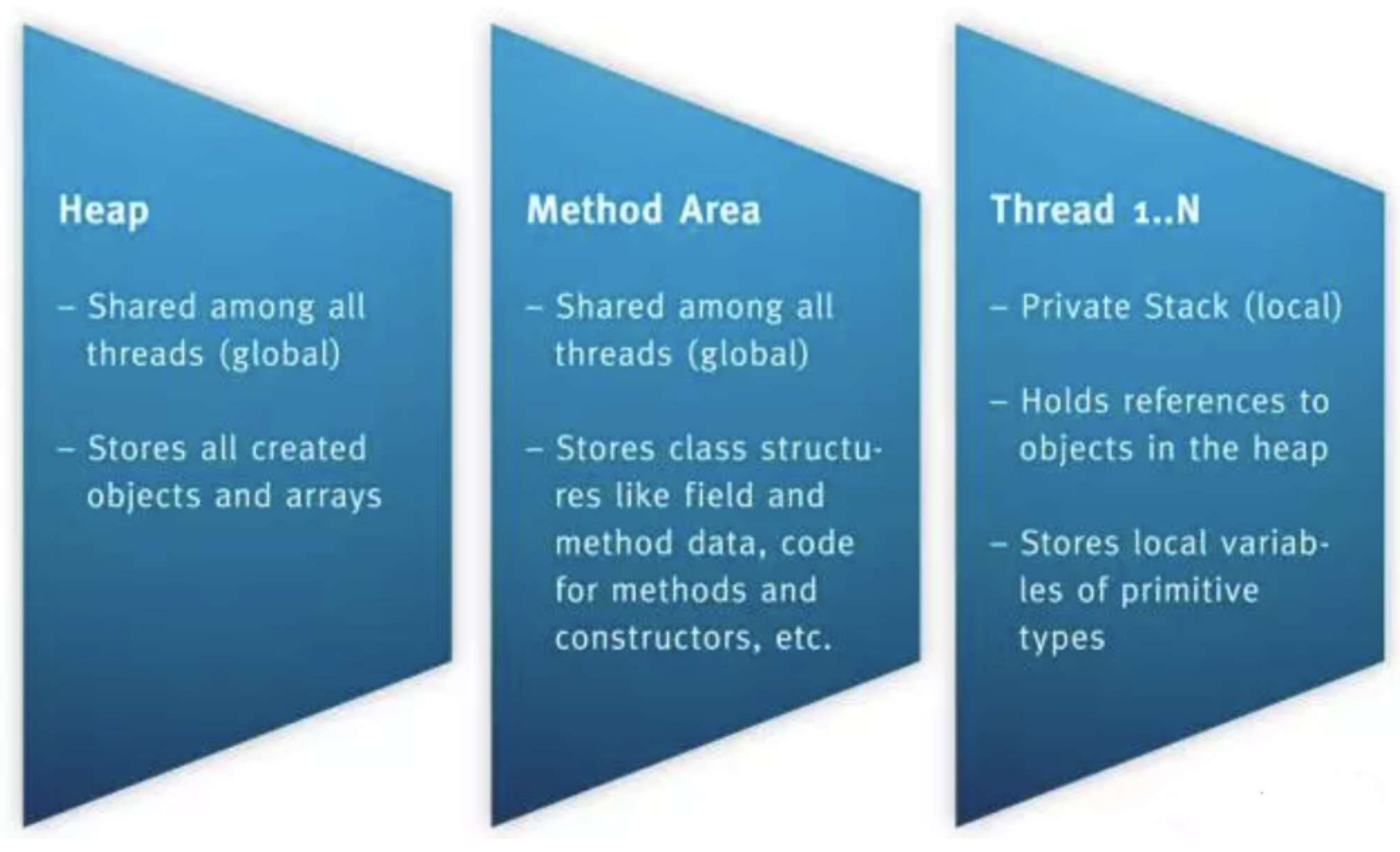
某一次，线程A获得CPU的执行权，开始执行内部程序。但是线程A的程序还没有执行完，在某一时刻CPU的执行权被另一个线程B抢走了。后来经过线程A的不懈努力，又抢回了CPU的执行权，那么线程A的程序又要从头开始执行?

这个时候程序计数器就粉墨登场了，它的作用就是记录当前线程所执行的位置。 这样，当线程重新获得CPU的执行权的时候，就直接从记录的位置开始执行，分支、循环、跳转、异常处理也都依赖这个程序计数器来完成。此外，程序计数器还具有以下特点：

- 线程私有，每一个线程都有一个程序计数器，因此它是线程安全的。
- 唯一一块不存在OutOfMemoryError的区域，可能是设计者觉得没必要。

# 代码示例分析

对于Java堆、方法区、线程独享区域（主要是虚拟机栈），方法的执行都是伴随着线程的，原始类型的本地变量以及引用都存放在线程栈中，而引用关联的对象比如String，都存在在堆中。



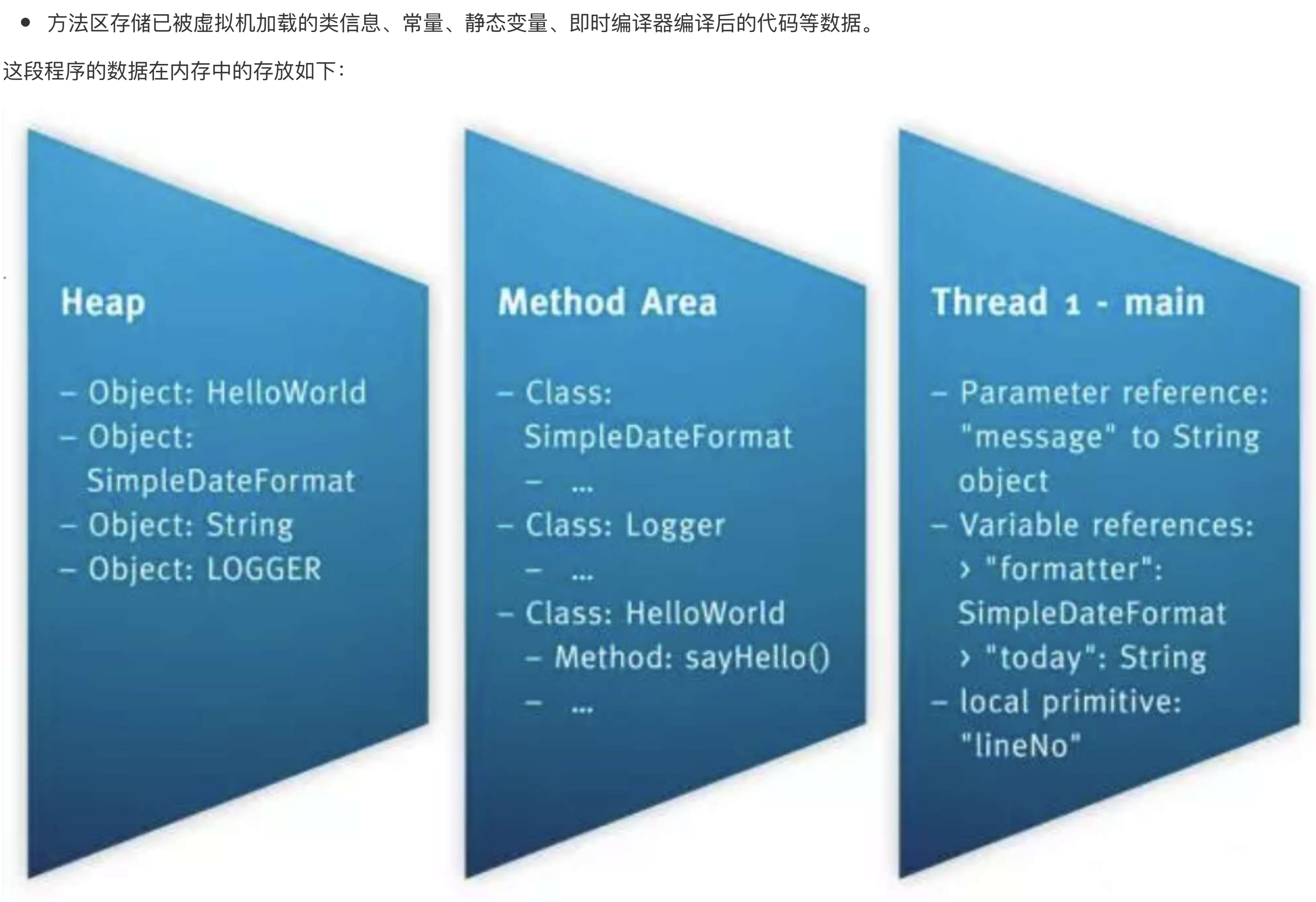
为了更好的理解上面这幅图，我们可以看一个例子：

```java
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.logging.Logger;
public class HelloWorld {
    private static Logger LOGGER = Logger.getLogger(HelloWorld.class.getName());
    public void sayHello(String message) {
        SimpleDateFormat formatter =  new SimpleDateFormat("dd.MM.YYYY");
        String today = formatter.format(new Date());
        LOGGER.info(today + ":" + message);
    }
}
```

我们先回顾一下前面学习的知识：

- 堆存储的是我们new来的对象，不存放基本类型和对象引用；
- 栈存储存放基本数据类型、对象的引用和returnAddress类型；
- 方法区存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

这段程序的数据在内存中的存放如下：



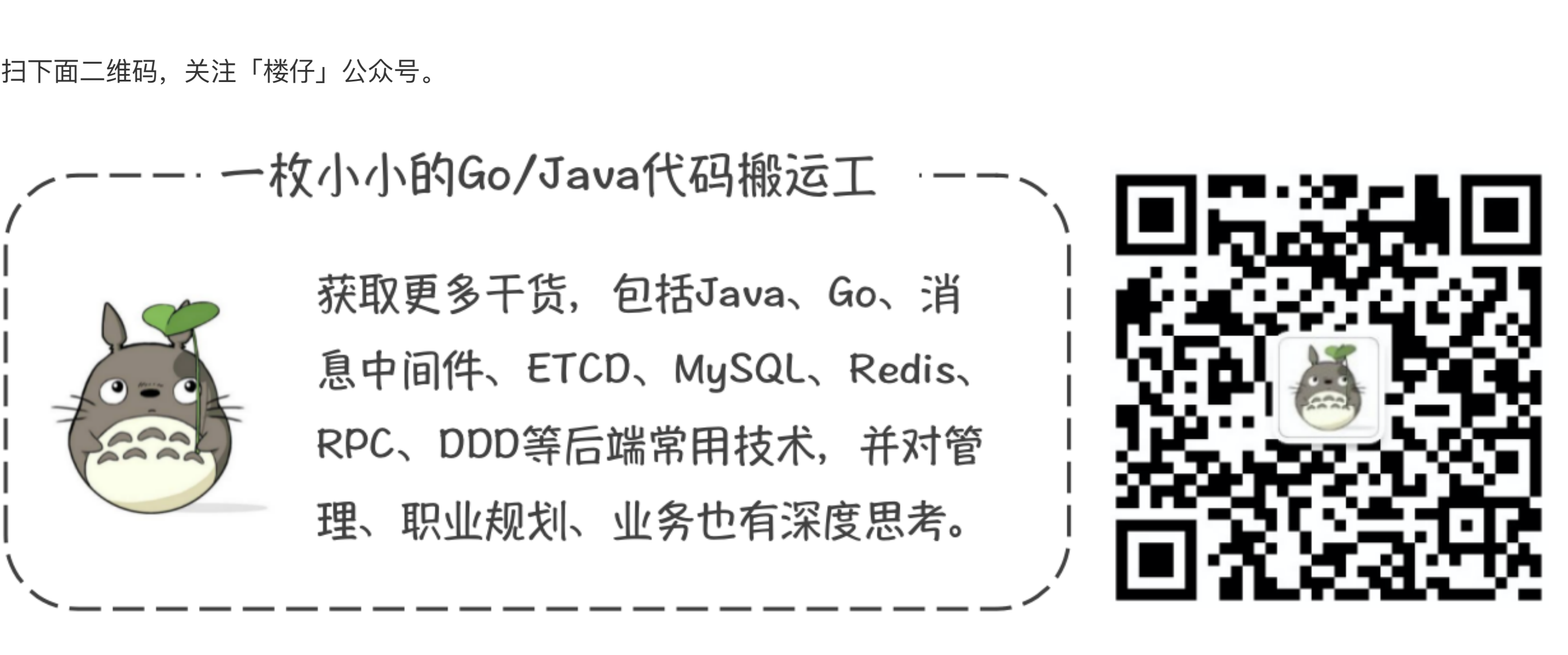
可以看出数据进行如下存放：

- Java堆：对象HelloWorld、对象SimpleDateFormat、对象String和对象LOGGER；
- 线程独享区域（主要是虚拟机栈）:message的引用、formatter的引用、today的引用；
- 方法区：类信息SimpleDateFormat、类信息Logger、类信息HelloWorld、方法sayHello()，还包括类信息的所有方法。

> 这个static Logger LOGGER，因为是静态变量，是不是应该属于方法区?

# 学习交流

可以扫下面二维码，关注「楼仔」公众号。



一枚小小的Go/Java代码搬运工

获取更多干货，包括Java、Go、消息中间件、ETCD、MySQL、Redis、RPC、DDD等后端常用技术，并对管理、职业规划、业务也有深度思考。

扫一扫 长按 关注我 让你懂技术、懂管理、懂业务，也懂生活

长按二维码，回复 **「加群」**，欢迎一起学习交流哈~~ 👏🏻👏🏻👏🏻

楼仔

湖北 武汉

扫一扫 长按
加技术群的备注：加群

> 尽信书则不如无书，因个人能力有限，难免有疏漏和错误之处，如发现 bug 或者有更好的建议，欢迎批评指正，不吝感激。