

java虚拟机栈

什么是栈

栈存储什么数据

栈的运行原理

栈溢出 StackOverflowError

什么是栈

栈管运行，堆管存储， 请熟读并默写。

栈 (Stack)，也叫栈内存，主管Java程序的运行，在线程创建时创建。其生命期是跟随线程的生命期，是线程私有的，线程结束栈内存也就是释放。**对于栈来说，不存在垃圾回收的问题。**

可以将栈想象成子弹弹夹！

程序 = 算法 + 数据结构

程序 = 框架 + 业务逻辑

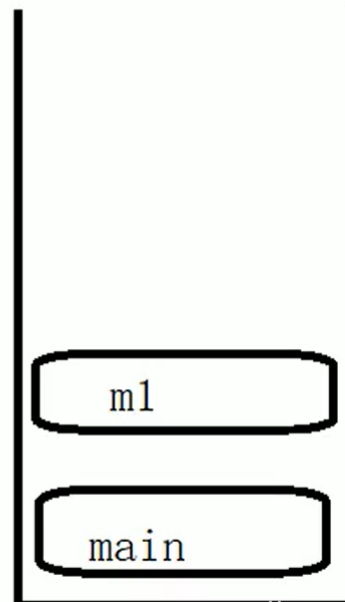
数据结构

队列 (FIFO)

栈 (FILO)

java 方法 = 栈帧

I



示例：

```

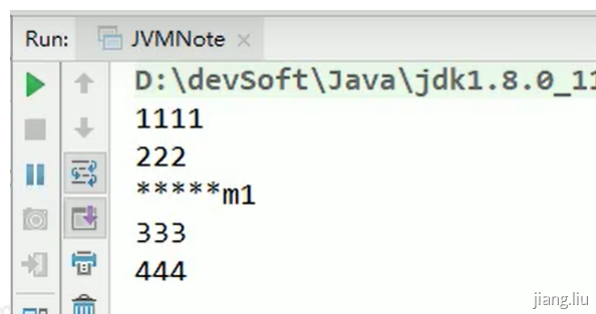
public class JVMNote
{
    public int add(int x ,int y)
    { ... }

    public static void m1()
    {
        System.out.println("222");
        System.out.println("*****m1");
        System.out.println("333");
    }

    public static void main(String[] args)
    {
        System.out.println("1111");
        m1();
        System.out.println("444");
    }
}

```

输出：



```

Run: JVMNote x
D:\devSoft\Java\jdk1.8.0_11
1111
222
*****m1
333
444

```

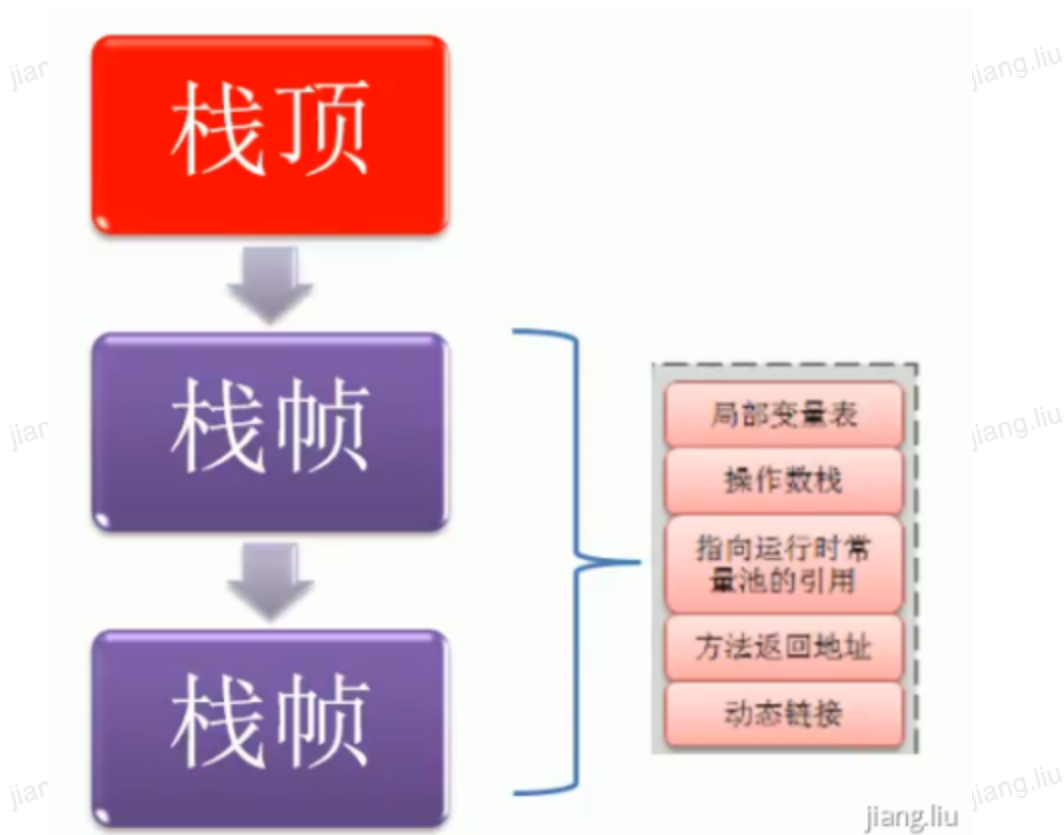
栈存储什么数据

栈主要存储8种基本类型的变量、对象的引用变量、以及实例方法。

这里引出一个名词，**栈帧**，什么是栈帧？

每个方法执行的同时都会创建一个栈帧，用于存储局部变量表、操作数栈、动态链接、方法出口等信息，每个方法从调用直至执行完毕的过程，就对应着一个栈帧在虚拟机中入栈到出栈的过程。

方法在jvm外在java语言层次就叫**方法**，在jvm里面就叫**栈帧**。



简单来说，**栈帧**对应一个方法的执行和结束，是方法执行过程的内存模型。将方法看做栈帧，方法的输入参数、输出参数、以及方法内的变量都是保存于栈帧中。

其中，栈帧主要保存3类数据：

1. **本地变量（Local Variables）**：输入参数和输出参数，以及方法内的变量。
2. **栈操作（Operand Stack）**：记录出栈、入栈的操作。
3. **栈帧数据（Frame Data）**：包括类文件、方法等。

栈的大小跟JVM有关，一般在256K~756K之间，约等于1Mb左右。

栈的运行原理

栈中的数据都是以栈帧（Stack Frame）的格式存在，栈帧是一个内存区块，是一个数据集，是一个有关方法（Method）和运行期数据的数据集。

当一个方法A被调用时就产生了一个栈帧F1，并被压入到栈中，方法A中又调用了方法B，于是产生栈帧F2也被压入栈中，方法B又调用方法C，于是产生栈帧F3也被压入栈中

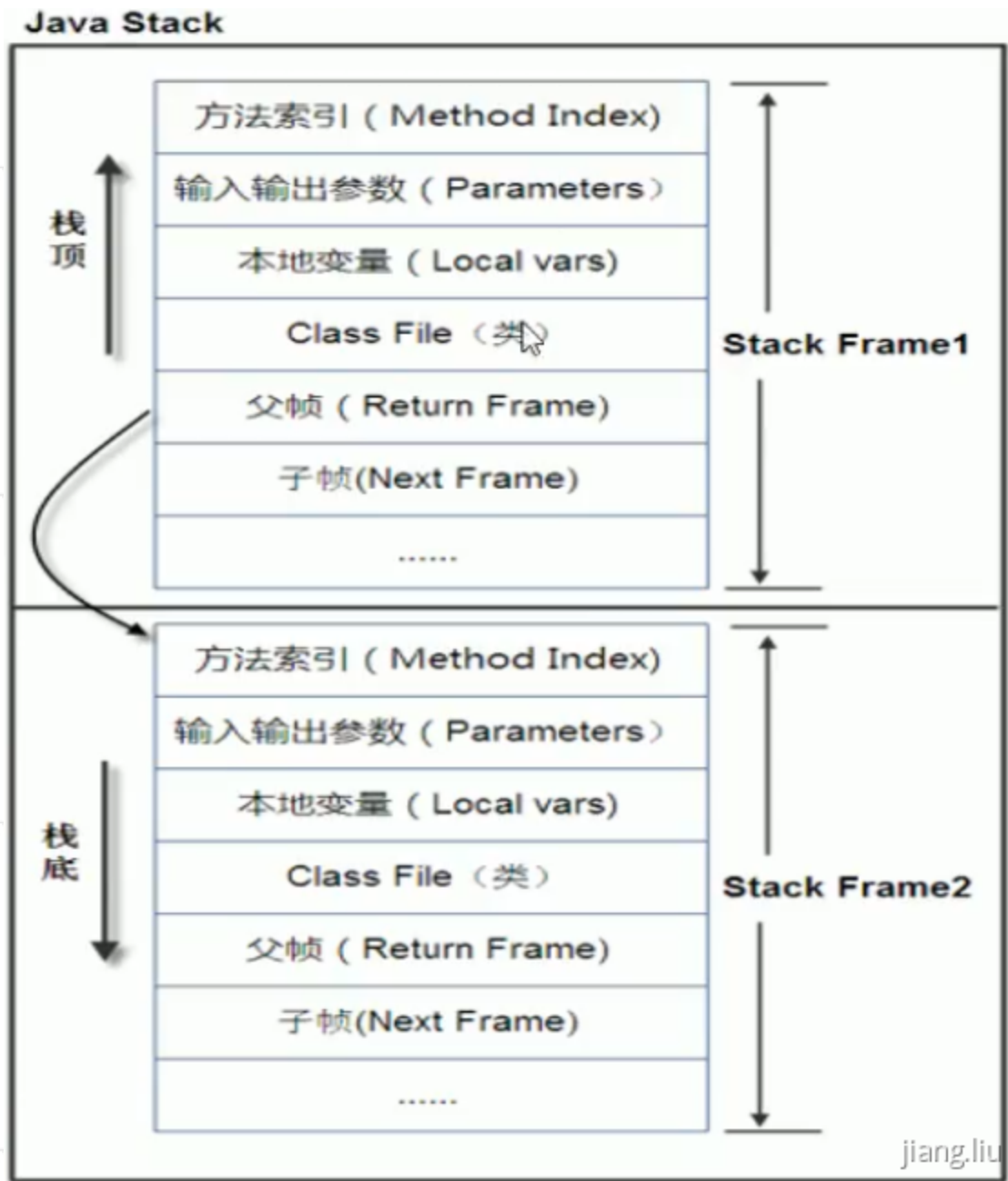
.....

执行完毕后，先弹出F3栈帧，再弹出F2栈帧，再弹出F1栈帧。

遵循“先进后出，后进先出”的原则。

观察下图，在图中一个栈中有两个栈帧，分别是 Stack Frame1 和 Stack Frame2，对应方法1和方法2。其中 Stack Frame2 是最先被调用的方法2，所以它先入栈。然后方法2又调用了方法1，所以 Stack Frame1 处于栈顶位置。执行完毕后，依次弹出 Stack Frame1 和 Stack Frame2，然后线程结束，栈释放。

所以，每执行一个方法都会产生一个栈帧，并保存到栈的顶部，顶部的栈帧就是当前所执行的方法，该方法执行完毕后会自动出栈。



栈溢出 StackOverflowError

大家肯定对栈溢出耳熟，那栈溢出是怎么产生的呢？

看下面代码：`m1()` 方法调用 `m1()`，自己调用自己，也叫递归。

栈是一个内存块，它是有大小长度的，而我们观察代码发现，只要代码一运行，`m1()` 方法就会一直进行入栈操作，而没有出栈操作，结果肯定会超出栈的大小，进而造成栈溢出错误，即 `java.lang.StackOverflowError`。

```
public class JVMNote
{
    public static void m1() throws InterruptedException
    {
        m1();
    }

    public static void main(String[] args) throws InterruptedException
    {
        System.out.println("1111");
        m1();
        System.out.println("444");
    }
}
```

```
D:\devSoft\Java\jdk1.8.0_111\bin\java.exe ...
1111
Exception in thread "main" java.lang.StackOverflowError
    at com.atguigu.jvm1205.JVMNote.m1(JVMNote.java:49)
    at com.atguigu.jvm1205.JVMNote.m1(JVMNote.java:49)
    at com.atguigu.jvm1205.JVMNote.m1(JVMNote.java:49)
    at com.atguigu.jvm1205.JVMNote.m1(JVMNote.java:49)
    at com.atguigu.jvm1205.JVMNote.m1(JVMNote.java:49)
    at com.atguigu.jvm1205.JVMNote.m1(JVMNote.java:49)
    at com.atguigu.jvm1205.JVMNote.m1(JVMNote.java:49)
```

所以说，老哥们，禁止套娃，禁止套娃，禁止套娃！!!!Σ(° Δ° ノ)ノ

`java.lang.StackOverflowError` 是错误（error级），不是异常！证明如下：

jiang.liu
java.lang

Class StackOverflowError

java.lang.Object

java.lang.Throwable

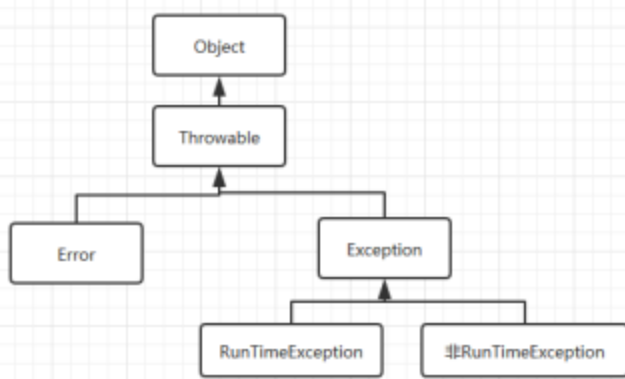
java.lang.Error

java.lang.VirtualMachineError

java.lang.StackOverflowError

题外知识：

Java 异常的类型图如下：



Throwable是Error和Exception的父类，用来定义所有可以作为异常被抛出来的类。

Error和Exception区分：

Error是编译时错误和系统错误，系统错误在除特殊情况下，都不需要你来关心，基本不会出现。而编译时错误，如果你使用了编译器，那么编译器会提示。

Exception则是可以被抛出的基本类型，我们需要主要关心的也是这个类。

Exception又分为RuntimeException和其他Exception。

RuntimeException和其他Exception区分：

1. 其他Exception，**受检查异常**。可以理解为错误，必须要开发者解决以后才能编译通过，解决的方法有两种，1: throw到上层，2, try-catch处理。
2. RuntimeException：运行时异常，**又称不受检查异常**，不受检查！不受检查！！不受检查！！！重要的事情说三遍，因为不受检查，所以在代码中可能会有RuntimeException时Java编译检查时不会告诉你有这个异常，但是在实际运行代码时则会暴露出来，比如经典的1/0，空指针等。如果不处理也会被Java自己处理。

参考资料: <https://www.cnblogs.com/hwttnet/p/12200320.html>