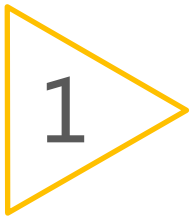
Java线程的创建

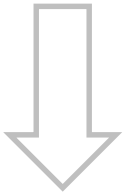


规范

创建一个Java线程

- 用户创建一个线程**唯一**的方法就是创建一个Thread对象。当start()方法调用时，线程将会启动。

——Java 语言规范 Java SE 7 Edition Chapter 17



```
Thread t = new Thread(runnable);
t.start();
```

创建Java线程——api视角

- new Thread()：创建java.lang.Thread对象

java.lang.Thread.Thread()

Allocates a new `Thread` object. This constructor has the same effect as Thread `(null, null, gname)`, where `gname` is a newly generated name. Automatically generated names are of the form `"Thread-"+`*n*, where *n* is an integer.

- thread.start();

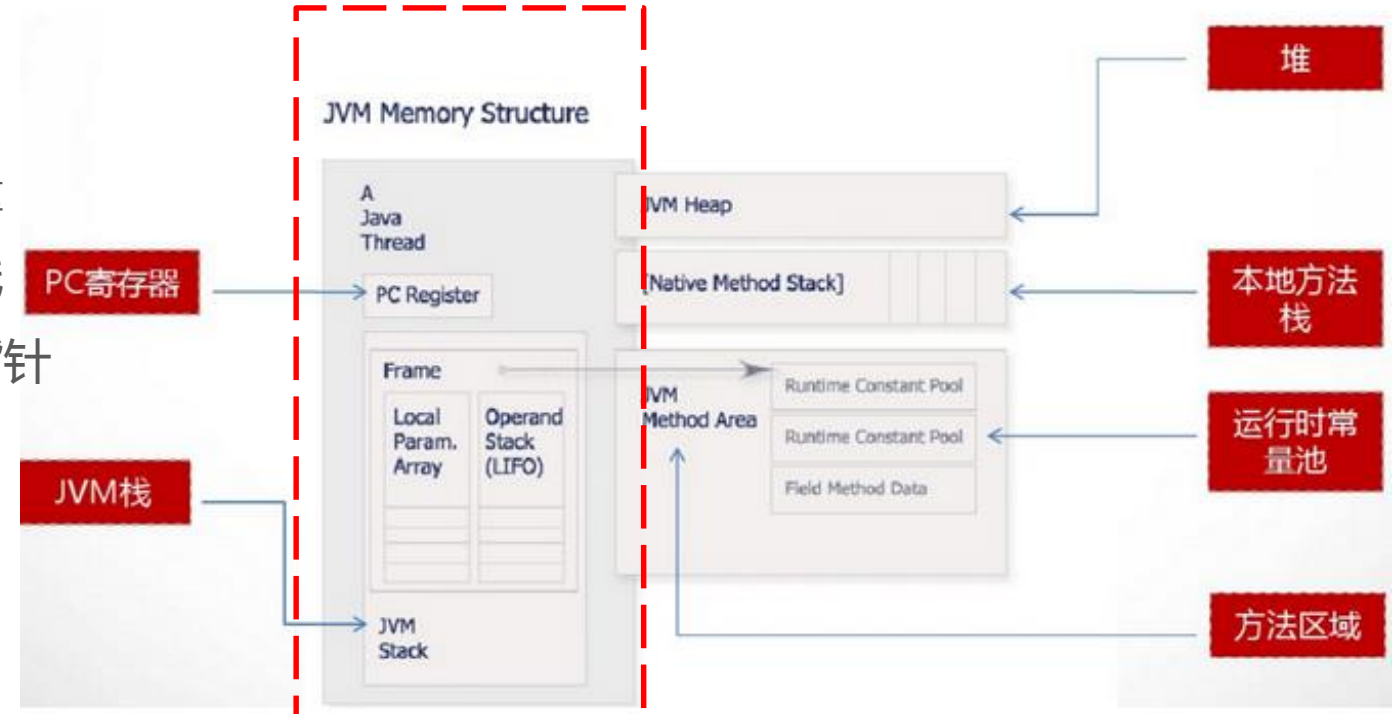
void java.lang.Thread.start()

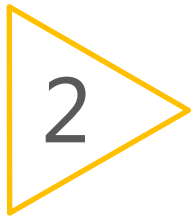
Causes this thread to begin execution; the Java Virtual Machine calls the `run` method of this thread.

The result is that two threads are running concurrently: the current thread (which returns from the call to the `start` method) and the other thread (which executes its `run` method).

创建Java线程——内存视角

- 每个Java线程都有自己的JVM栈
  - PC寄存器
  - 栈帧
    - 局部变量
    - 操作数栈
    - 常量池指针





2 实现

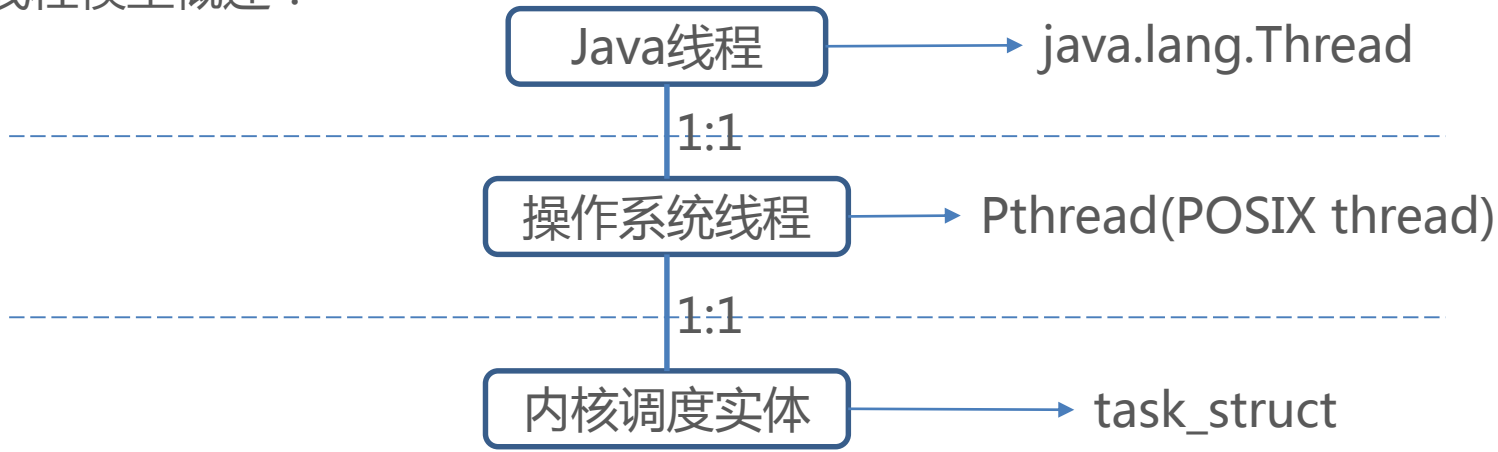
（以Hotspot虚拟机，Linux操作系统x86为例）

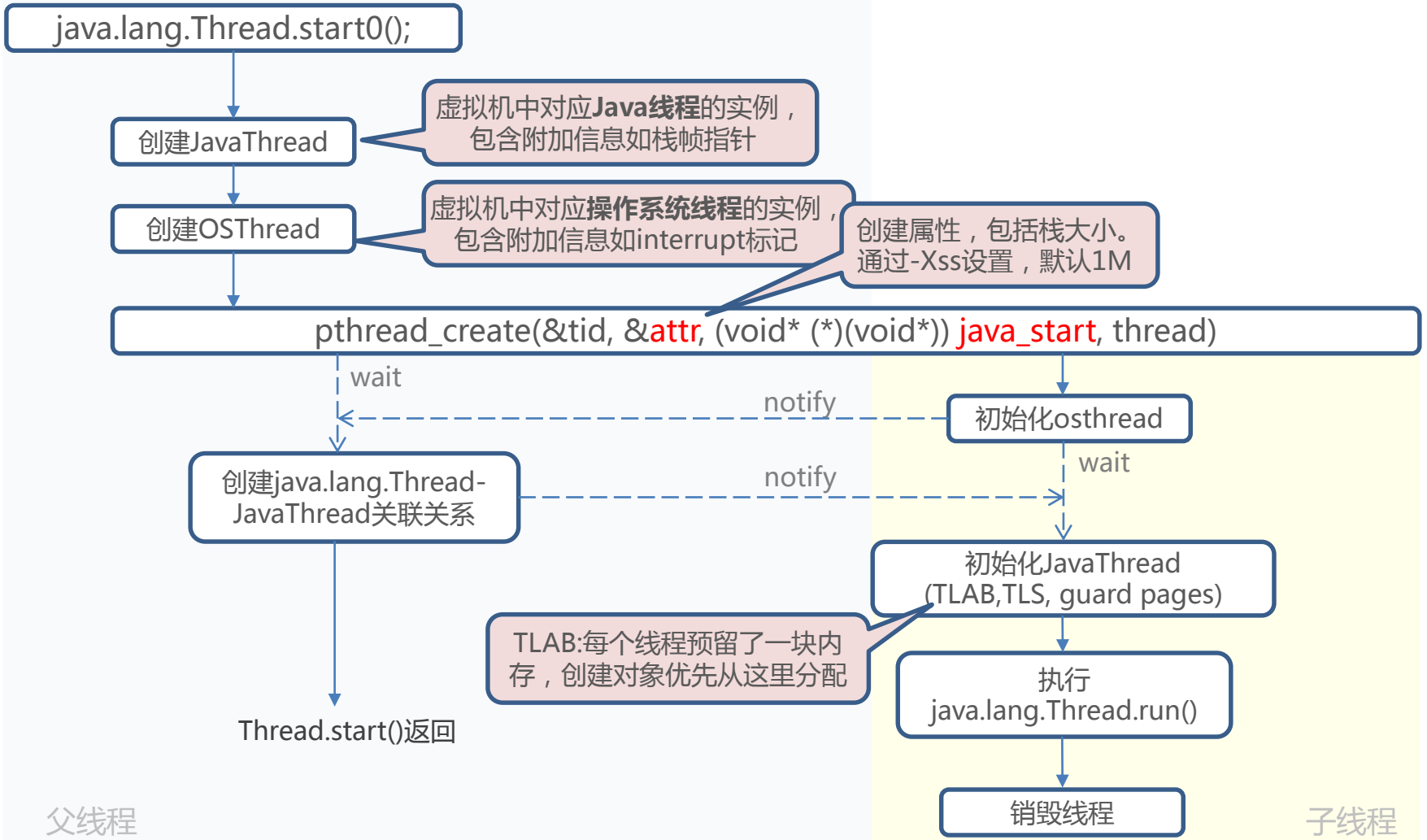
new Thread()

- new Thread()：创建java.lang.Thread对象
  - 继承父线程daemon, priority, contextClassLoader
  - 没有真正创建线程

thread.start()

- thread.start()概述：创建线程，并启动线程
- 线程模型概述：




java.lang.Thread.start0();
创建JavaThread
虚拟机中对应Java线程的实例，包含附加信息如栈帧指针
创建OSThread
虚拟机中对应操作系统线程的实例，包含附加信息如interrupt标记
创建属性，包括栈大小。通过-Xss设置，默认1M
pthread_create(&tid, &attr, (void* (*)(void*)) java_start, thread)
wait
notify
初始化osthread
wait
notify
创建java.lang.Thread-JavaThread关联关系
初始化JavaThread (TLAB,TLS, guard pages)
TLAB:每个线程预留了一块内存，创建对象优先从这里分配
执行java.lang.Thread.run()
销毁线程
Thread.start()返回
父线程
子线程

线程栈举例

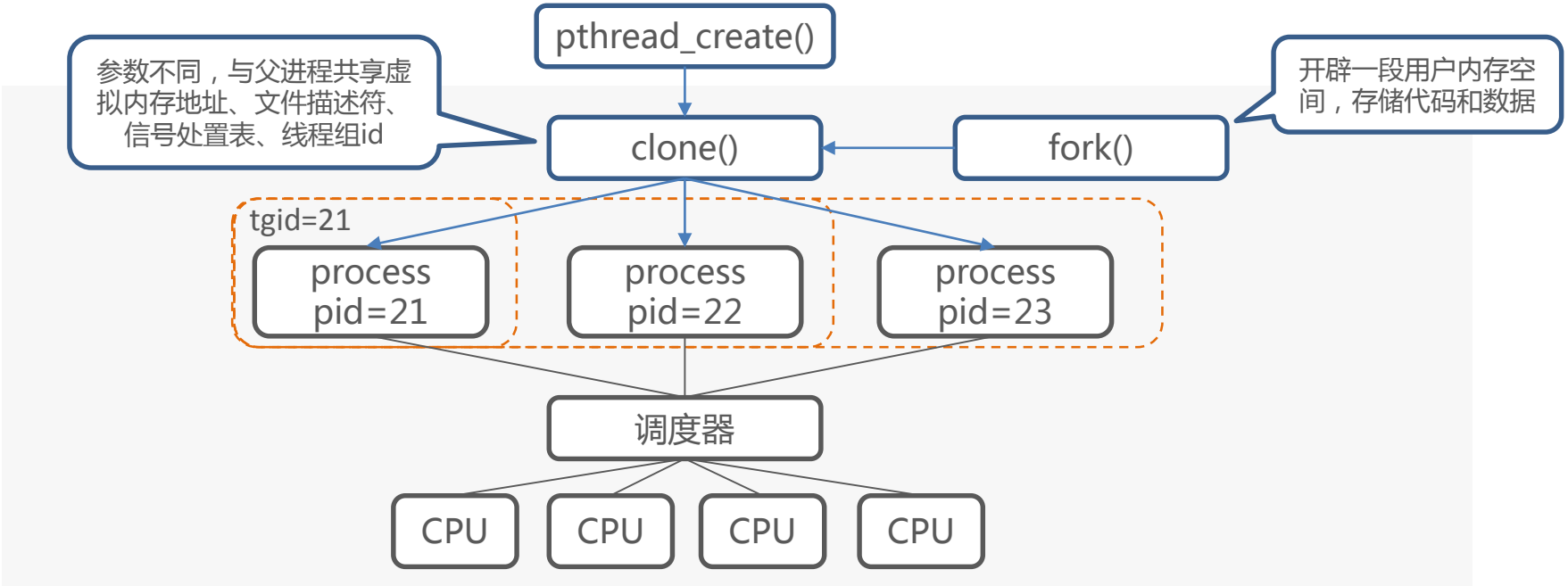
- pthread_create()是Linux/Unix系统下创建POSIX线程的方法
- Linux 2.6+ 使用NPTL(Native POSIX Thread Library) 作为pthread的实现

```
0x00007f7fe6dfef59      __pthread_cond_wait + 0xb9
0x00007f7fdc968802      * sun.misc.Unsafe.park(boolean, long) bci:0 (Interpreted frame)
0x00007f7fdc9563d0      * java.util.concurrent.locks.LockSupport.park(java.lang.Object) bci:14 line:186 (Interpreted frame)
0x00007f7fdc9563d0      * java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await() bci:42 line:2043 (Interpreted fram
0x00007f7fdc956a7e      * java.util.concurrent.LinkedBlockingQueue.take() bci:29 line:442 (Interpreted frame)
0x00007f7fdc9568a3      * com.[illegible] service.[illegible] ServiceImpl$Comsumer.run() bci:14 line:866 (Interpreted frame)
0x00007f7fdc956a7e      * java.lang.Thread.run() bci:11 line:745 (Interpreted frame)
0x00007f7fdc9504e7      <StubRoutines>
0x00007f7fe5d5f51a      JavaCalls::call_helper(JavaValue*, methodHandle*, JavaCallArguments*, Thread*) + 0x21a
0x00007f7fe5967533      JavaCalls::call_virtual(JavaValue*, KlassHandle, Symbol*, Symbol*, JavaCallArguments*, Thread*) + 0x113
0x00007f7fe5929434      __PGOSF3__Z12thread_entryP10JavaThreadP6Thread + 0x114
0x00007f7fe585a110      JavaThread::run() + 0x230
0x00007f7fe597f81a      java_start(Thread*) + 0x15a
```

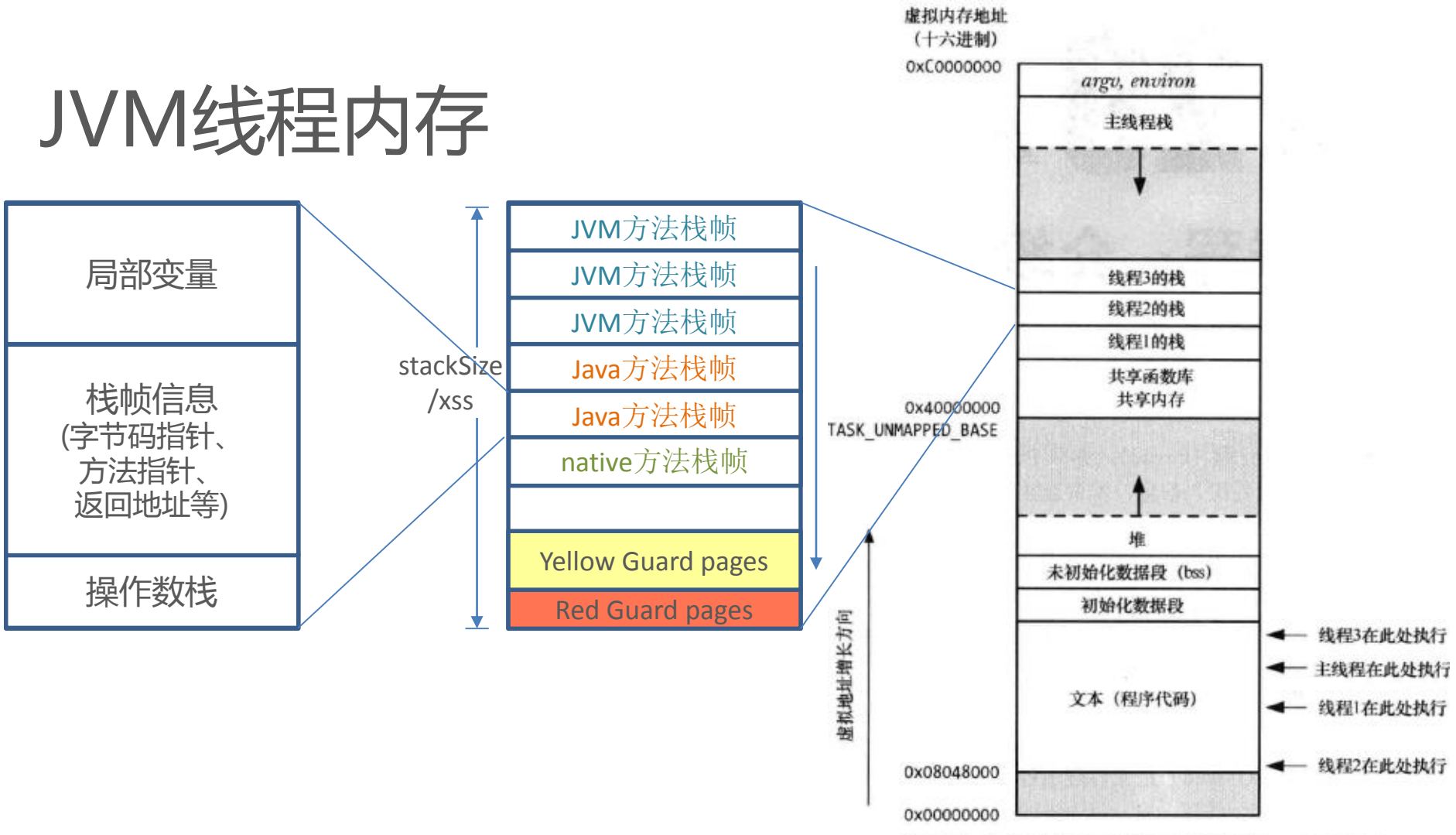
- jstack –m <pid>

pthread_create()

- pthread_create()是Linux/Unix系统下创建POSIX线程的方法
- Linux 2.6+ 使用NPTL(Native POSIX Thread Library) 作为pthread的实现



JVM线程内存


局部变量
栈帧信息
(字节码指针、
方法指针、
返回地址等)
操作数栈
stackSize
/xss
JVM方法栈帧
JVM方法栈帧
JVM方法栈帧
Java方法栈帧
Java方法栈帧
native方法栈帧
Yellow Guard pages
Red Guard pages
虚拟内存地址
(十六进制)
0xC0000000
argv, environ
主线程栈
线程3的栈
线程2的栈
线程1的栈
共享函数库
共享内存
0x40000000
TASK_UNMAPPED_BASE
堆
未初始化数据段（bss）
初始化数据段
文本（程序代码）
0x08048000
0x00000000
虚拟地址增长方向
线程3在此处执行
主线程在此处执行
线程1在此处执行
线程2在此处执行

线程优先级

- Java线程优先级　1(低)~10(高)
- Linux线程nice值　-20(高)~19(低)
- 优先级越高，享受的CPU时间越多

```java
/**
 * The minimum priority that a thread can have.
 */
public final static int MIN_PRIORITY = 1;

/**
 * The default priority that is assigned to a thread.
 */
public final static int NORM_PRIORITY = 5;

/**
 * The maximum priority that a thread can have.
 */
public final static int MAX_PRIORITY = 10;
```

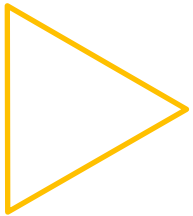
```
int os::java_to_os_priority[MaxPriority + 1] = {
  19,               // 0 Entry should never be used

   4,               // 1 MinPriority
   3,               // 2
   2,               // 3

   1,               // 4
   0,               // 5 NormPriority
  -1,               // 6

  -2,               // 7
  -3,               // 8
  -4,               // 9 NearMaxPriority

  -5                // 10 MaxPriority
};
```



THANKS