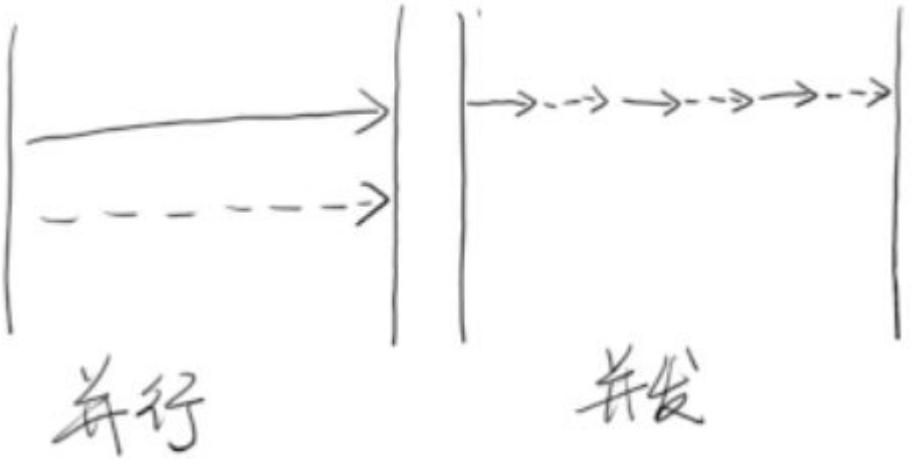
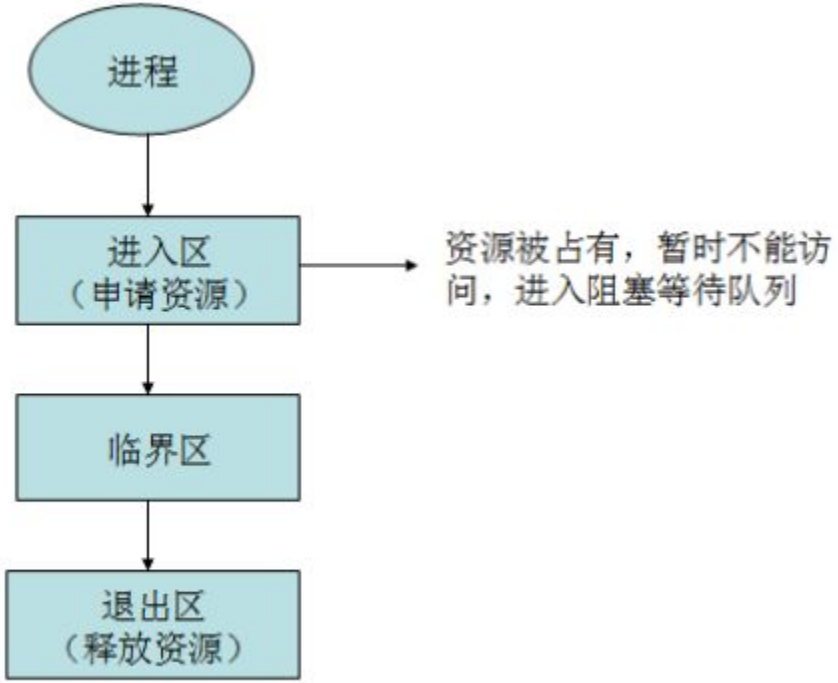
Multithreading Interview by Java

Parallel vs Concurrent



Critical Section临界区



Blocking vs Non-Blocking

阻塞:一个线程占用了临界区资源，那么其它所有需要这个资源的线程就必须在这个临界区中进行等待，等待会导致线程挂起。

非阻塞:允许多个线程同时进入临界区。

Other Terms

- Deadlock
- Livelock
- Starvation

Thread Life Cycle

- New 创建状态
- Runnable 可运行状态
- Non-Runnable (Blocked) 不可运行状态
- Running
- Terminated

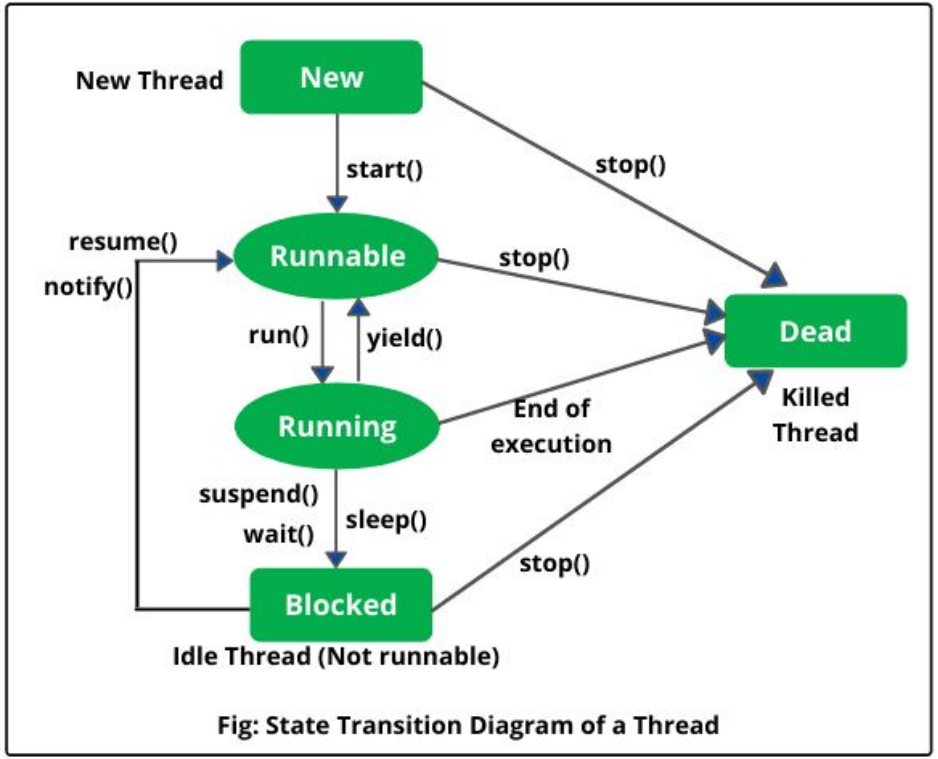


Fig: State Transition Diagram of a Thread

线程的优先级

线程优先级及设置

- 线程创建时，子继承父的优先级；
- 线程创建后，可通过调用 setPriority() 方法改变优先级；
- 线程的优先级是1-10之间的正整数。

线程的Scheduling策略

- 线程体中调用了 yield() 方法，让出了对 CPU 的占用权；
- 线程体中调用了 sleep() 方法，使线程进入睡眠状态；
- 线程由于 I/O 操作而受阻塞；
- 另一个更高优先级的线程出现；
- 在支持时间片的系统中，该线程的时间片用完。

线程的创建方式

1. Extends Thread

```java
public class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("thread run...");
    }

    public static void main(String[] args) {
        new MyThread().start();
    }
}
```

线程的创建方式

2. Implements Runnable

```java
public class MyThread implements Runnable{
    @Override
    public void run() {
        System.out.println("thread run...");
    }

    public static void main(String[] args) {
        new Thread(new MyThread()).start();
    }
}
```

线程的创建方式

3. 实现 Callable 接口，并结合 Future

```java
public class TestFuture {
    public static void main(String[] args) throws Exception {
        FutureTask<Integer> task = new FutureTask<>(new MyThread());
        new Thread(task).start();
        Integer result = task.get(); //获取线程的执行结果，阻塞式
        System.out.println(result);
    }
}

class MyThread implements Callable<Integer>{
    @Override
    public Integer call() throws Exception {
        return new Random().nextInt(100);
    }
}
```

线程的创建方式

4. 线程池

```
public class MyThread implements Runnable{
    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName() + " thread run...");
    }

    public static void main(String[] args) {
        ExecutorService executorService = Executors.newFixedThreadPool(10);
        for (int i = 0; i < 10; i++) {
            executorService.execute(new MyThread());
        }
        executorService.shutdown();
    }
}
```

线程的创建方式

http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/lang/Thread.java

一种是继承 Thread 类，一种是实现 Runnable接口

新手注意点

- 不管是继承 Thread 类还是实现 Runable 接口，业务逻辑是写在 run 方法里面，线程启动的时候是执行 start() 方法；
- 开启新的线程，不影响主线程的代码执行顺序也不会阻塞主线程的执行；
- 新的线程和主线程的代码执行顺序是不能够保证先后的；
- 对于多线程程序，从微观上来讲某一时刻只有一个线程在工作，多线程目的是让 CPU 忙起来；
- 通过查看 Thread 的源码可以看到，Thread 类是实现了 Runnable 接口的，所以这两种本质上来讲是一个

为什么要用Thread Pool

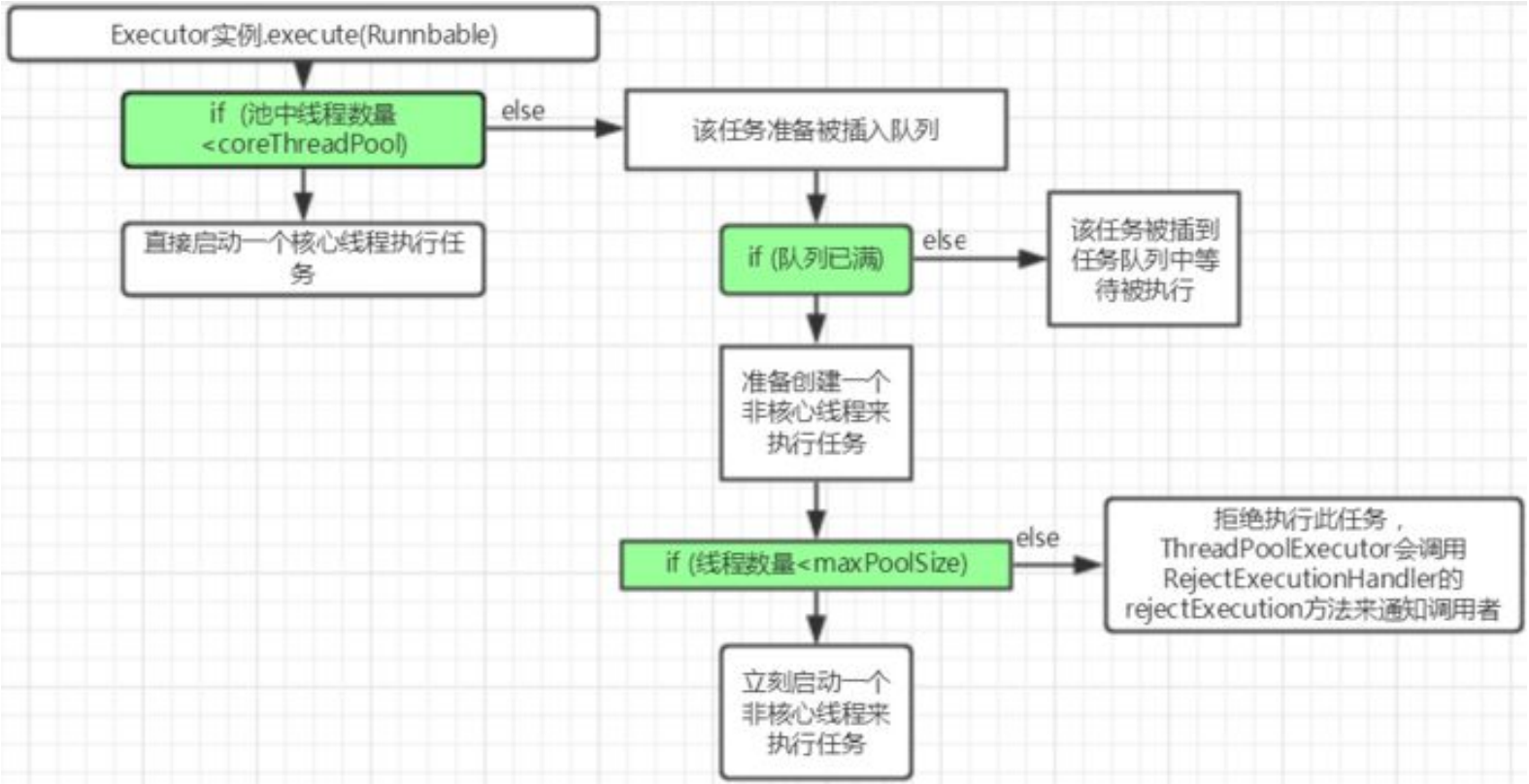
- 方便管理和调度，调优，监控
- 创建T1，执行T2，销毁T3 - T1 + T3 >> T2
- 高峰低谷的效率

ThreadPoolExecutor

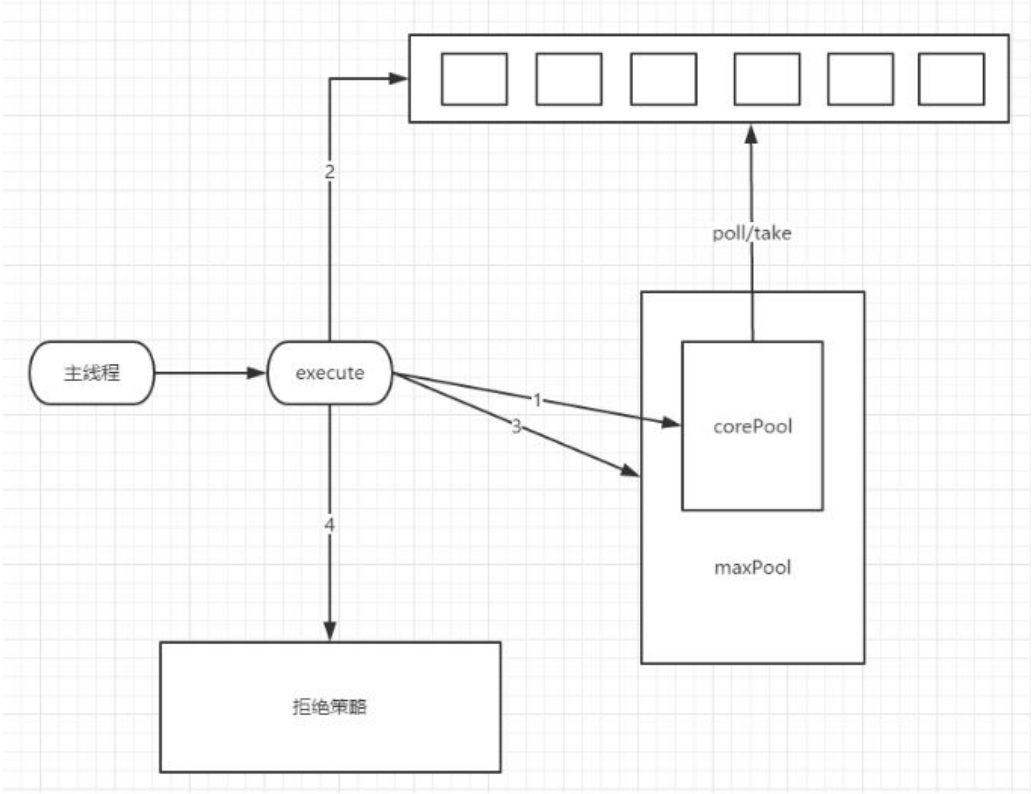
```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue);
```

各个参数一览 + threadFactory, handler

线程池工作流程图


Executor实例.execute(Runnbable)
if (池中线程数量<coreThreadPool)
else
该任务准备被插入队列
直接启动一个核心线程执行任务
if (队列已满)
else
该任务被插到任务队列中等待被执行
准备创建一个非核心线程来执行任务
if (线程数量<maxPoolSize)
else
拒绝执行此任务，ThreadPoolExecutor会调用RejectExecutionHandler的rejectExecution方法来通知调用者
立刻启动一个非核心线程来执行任务

线程池中各个参数的工作

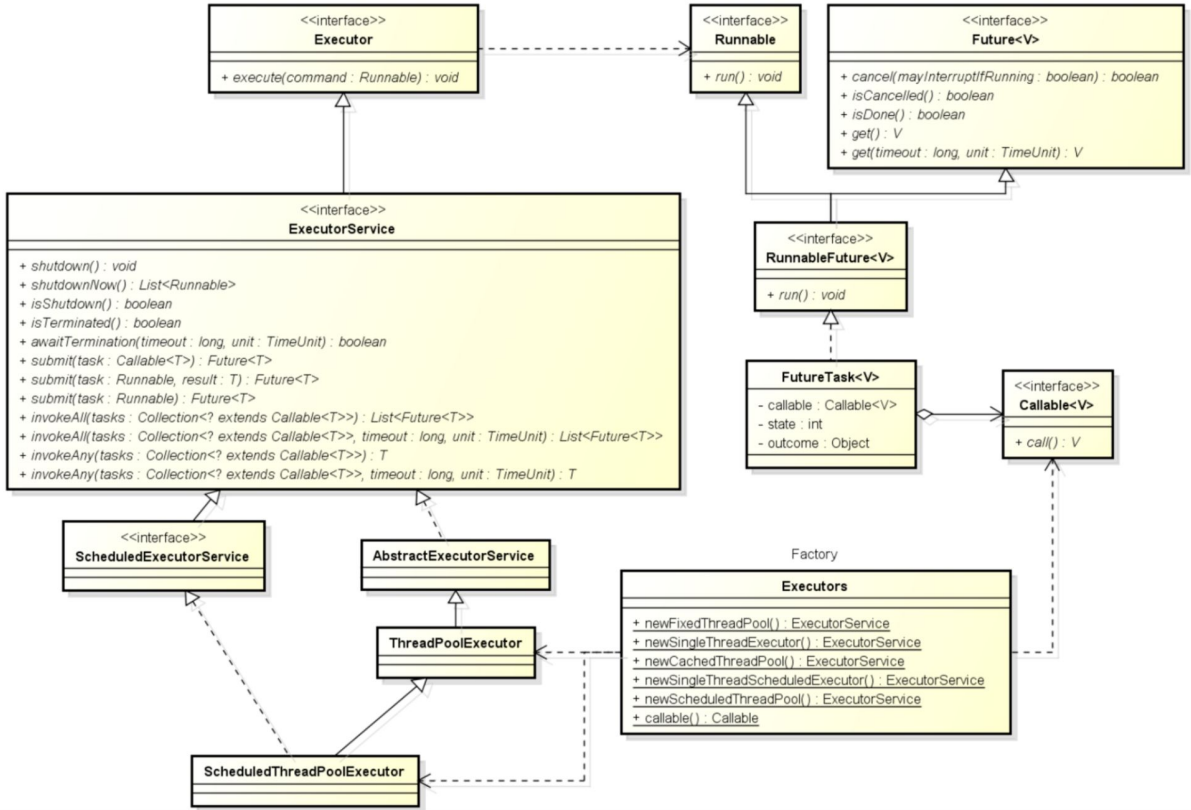


Handler拒绝策略

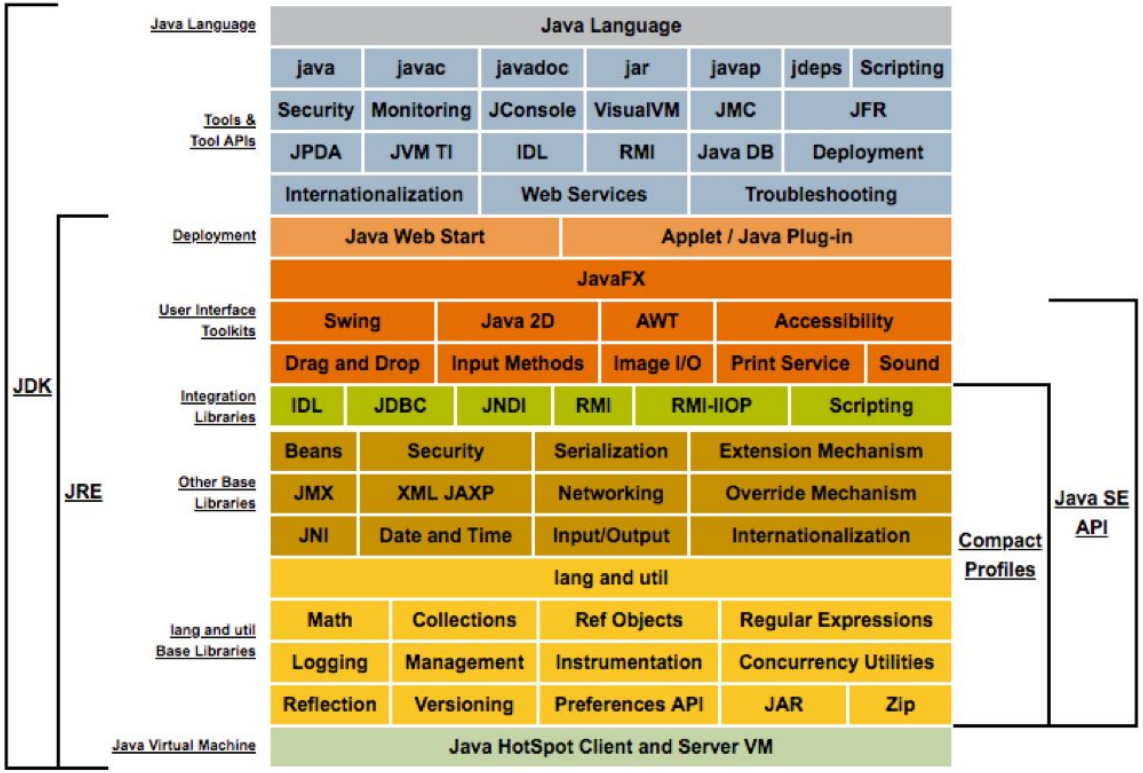
1. ThreadPoolExecutor.AbortPolicy：丢弃任务并抛出 RejectedExecutionException 异常；
2. ThreadPoolExecutor.DiscardPolicy：也是丢弃任务，但是不抛出异常；
3. ThreadPoolExecutor.DiscardOldestPolicy：丢弃队列最前面的任务，然后重新尝试执行任务（重复此过程）；
4. ThreadPoolExecutor.CallerRunsPolicy：由调用线程处理该任务。

Important classes/interfaces

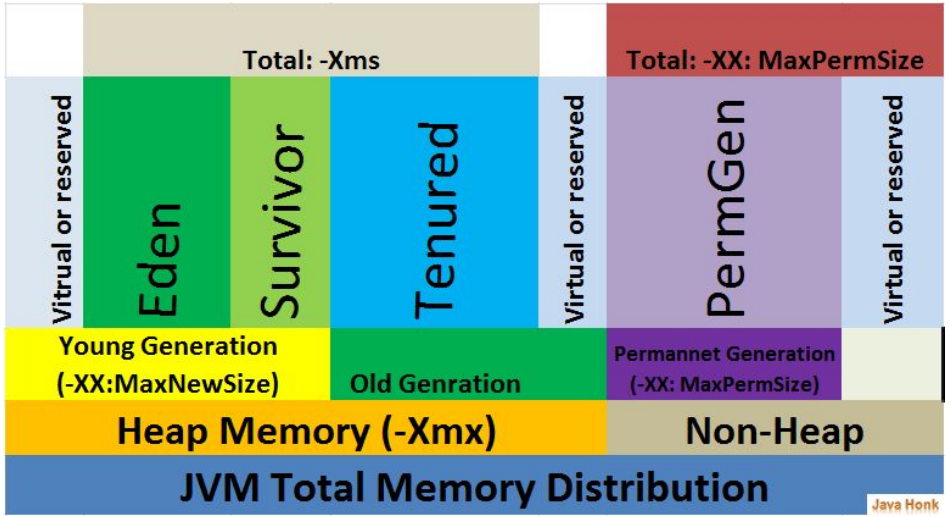
- Executors
- ExecutorService
- ScheduledExecutorService

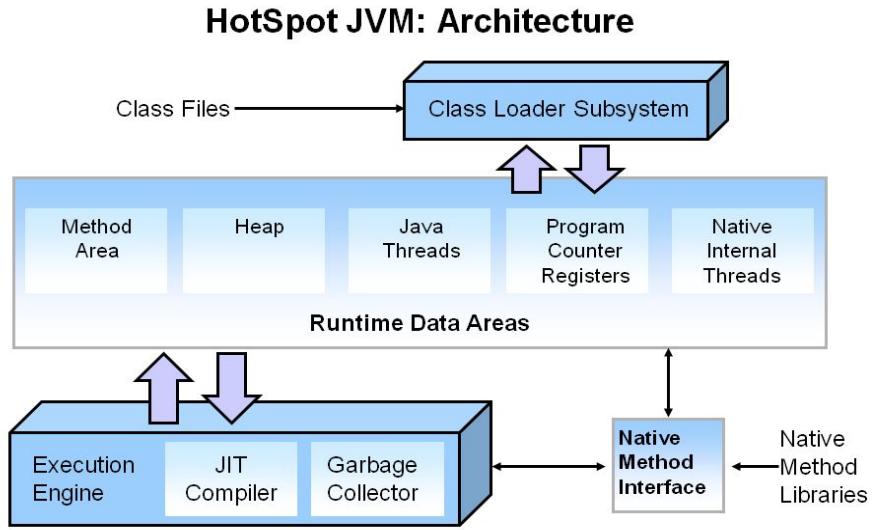


Java内存模型与线程

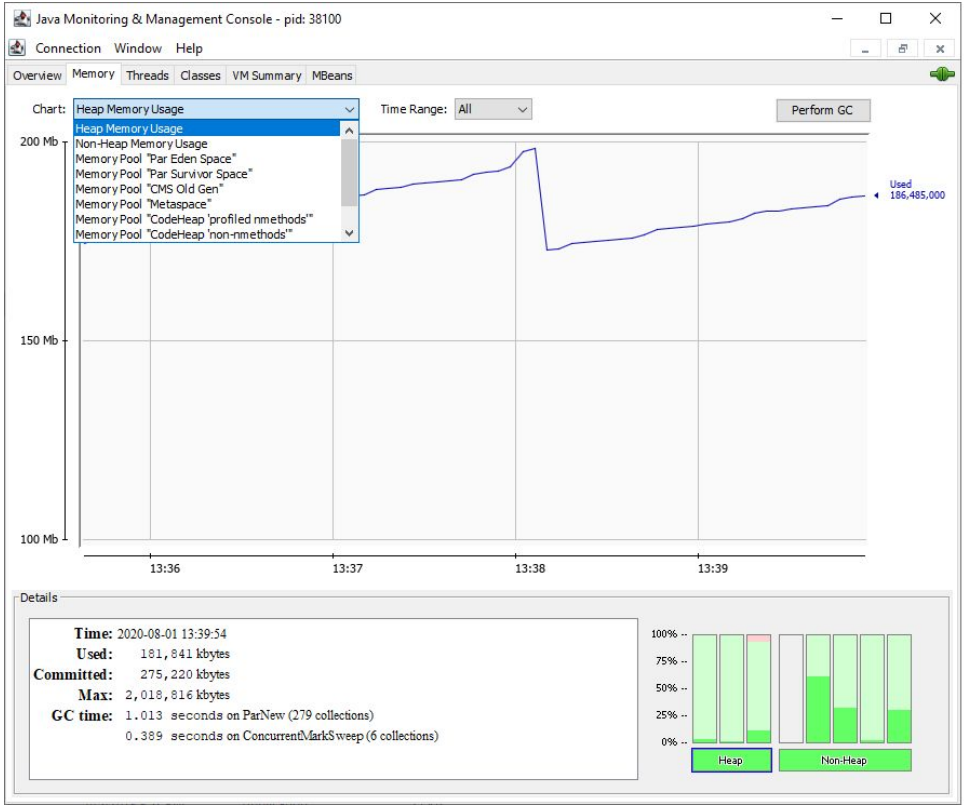


Java内存模型与线程

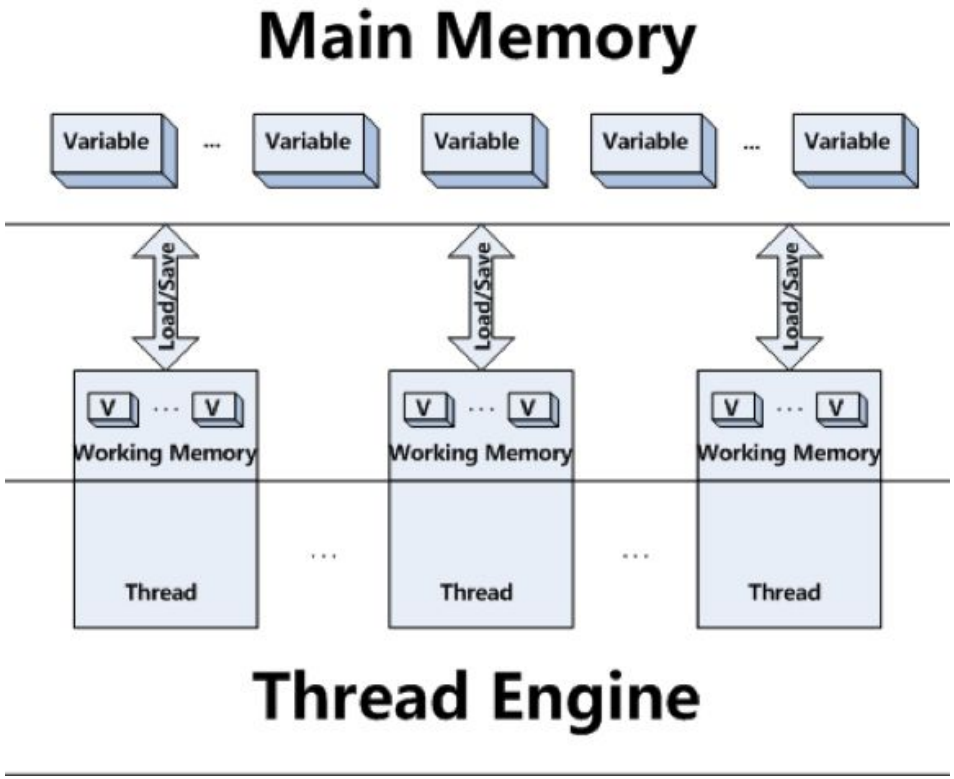




Java内存模型与线程



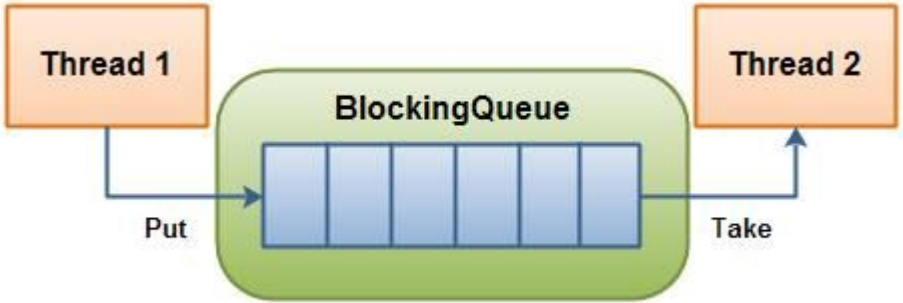
Java内存模型与线程



保证可见性

- volatile
- synchronized
- final

Queue



阻塞队列: ArrayBlockingQueue

非阻塞队列: CocurrentLinkedQueue

并发编程模型

- Producer/Consumer
- Leader/Follower
- Disruptor

多线程常规问题debug

- top/jps, jstack
- Example:

  死锁，Deadlock

  执行中，Runnable

  等待资源，Waiting on condition

  等待获取监视器，Waiting on monitor entry

  暂停，Suspended

  对象等待中，Object.wait() 或 TIMED_WAITING

  阻塞，Blocked

  停止，Parked