1、概念

1. 线程是什么

一个线程就是一个 "执行流". 每个线程之间都可以按照顺讯执行自己的代码. 多个线程之间 "同时" 执行着多份代码.

2. 线程出现的原因

1.首先, "并发编程" 成为 "刚需".

引入进程的目的,就是为了能够"并发编程",虽然编程已经能解决并发的问题了,但是认为,还不够理想,由于创建进程、销毁进程和进程调度都需要系统进行资源分配，如果频繁的创建和销毁，这样开销就会很大。于是就提出了一个“线程”（Thresd）概念，线程在系统上也叫做“轻量级进程”

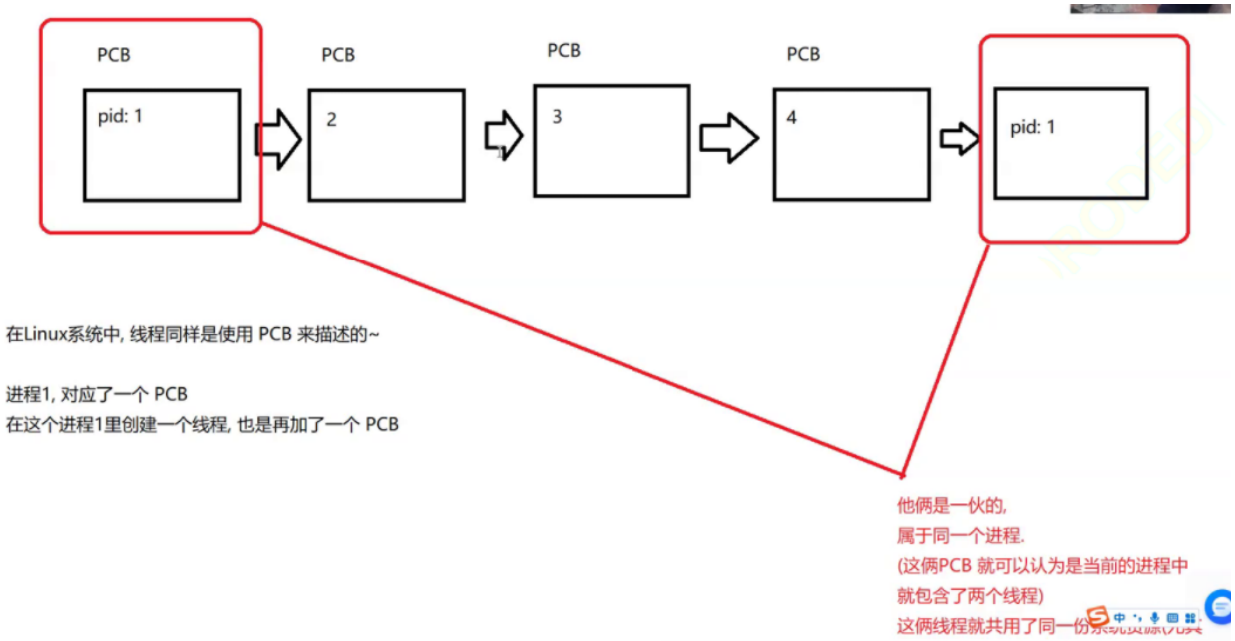
2. 其次, 虽然多进程也能实现 并发编程, 但是线程比进程更轻量.

- 创建线程比创建进程更快
- 销毁线程比销毁进程更快
- 调度线程比调度进程更快

注：创建线程，并没有去申请资源；销毁线程，也没有释放资源。让线程产生在进程内部，共用之前的资源

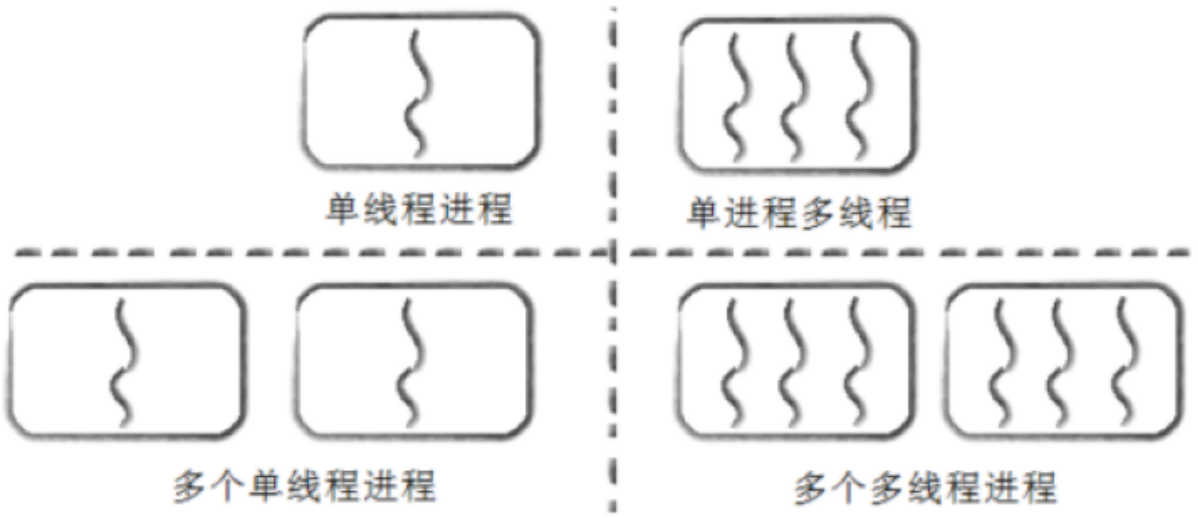
3. 进程和线程的区别和联系（经典面试题）

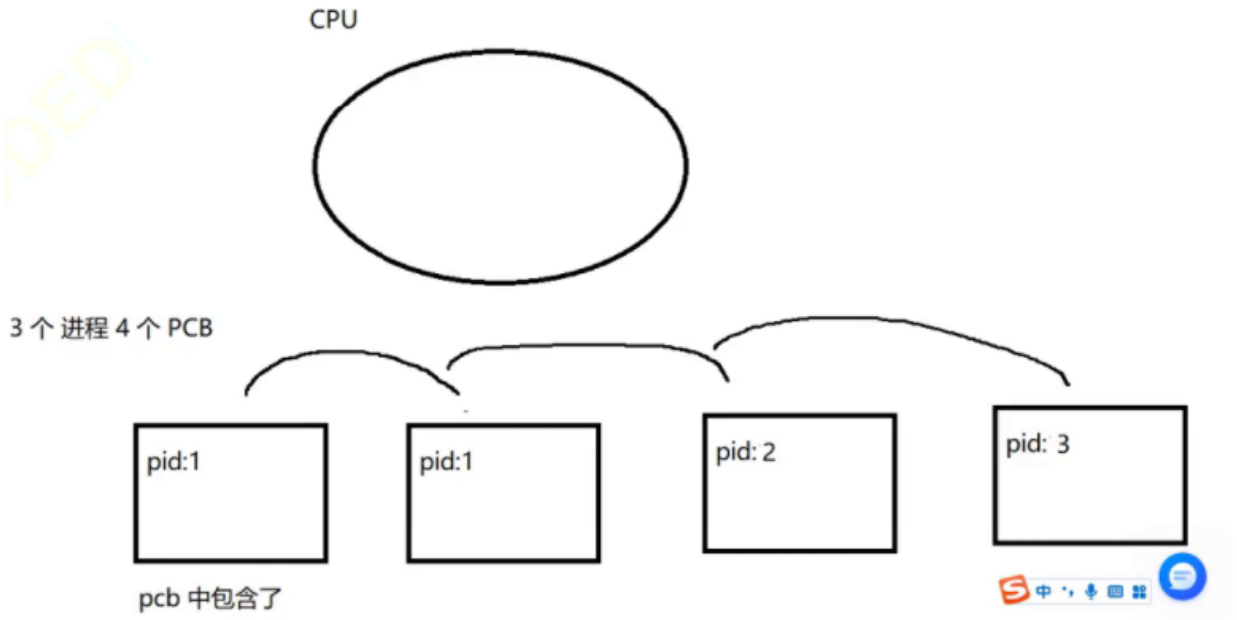
- 进程是包含线程的. 每个进程至少有一个线程存在，即主线程



当创建一个进程的时候，就创建一个PCB出来，同时这个PCB也就可以视为是当前进程中已经包含一个线程了（一个进程中至少得有一个线程）

- 进程和进程之间不共享内存空间. 同一个进程的线程之间共享同一个内存空间.
- 进程是系统分配资源的最小单位，线程是系统调度的最小单位。





一个进程里面如果某个线程抛出了异常，并且没有合理catch住的话，可能会导致整个进程都异常退出，其他的线程也就玩完了。因此，多线程程序的编写，其实就提出了一个更高的要求，一共要保证线程的稳定。

4. Java线程和操作系统线程的区别

线程是操作系统中的概念. 操作系统内核实现了线程这样的机制, 并且对用户层提供了一些 API 供用户使用(例如 Linux 的 pthread 库).

Java 标准库中 Thread 类可以视为是对操作系统提供的 API 进行了进一步的抽象和封装.

2、第一个多线程程序

```
1  public class ThreadDemo {
2  private static class MyThread extends Thread{
3  @Override
```

```
4   public void run() {
5   Random random = new Random();
6   while(true){
7   // 打印线程名称
8   System.out.println(Thread.currentThread().getName());
9   try{
10   // 随机停止运行 0~9 秒
11   Thread.sleep(random.nextInt(10));
12   }catch(InterruptedException e){
13   e.printStackTrace();
14   }
15   }
16   }
17   }
18   public static void main(String[] args) throws InterruptedExcep
tion {
19   Thread t1 = new MyThread();
20   Thread t2 = new MyThread();
21   Thread t3 = new MyThread();
22
23   t1.start();
24   t2.start();
25   t3.start();
26
27   Random random = new Random();
28   while(true){
29   // 打印线程名称
30   System.out.println(Thread.currentThread().getName());
31   try{
32   // 随机停止运行 0~9 秒
33   Thread.sleep(random.nextInt(10));
34   }catch(InterruptedException e){
35   e.printStackTrace();
36   }
37   }
```

```
38
39  }
40 }
```

3、创建线程

1.继承 Thread 类

```
1  class MyThread1 extends Thread {
2   @Override
3   public void run() {
4   while(true){
5   System.out.println("Hello Thread!");
6   try {
7   Thread.sleep(1000);
8   } catch (InterruptedException e) {
9   e.printStackTrace();
10   }
11   }
12   }
13  }
14
15  public class ThreadDemo1 {
16   public static void main(String[] args) {
17   Thread t = new MyThread1();
18   t.start();
19   // t.run();
20   while(true){
21   System.out.println("Hello main!");
22   try {
23   Thread.sleep(1000);
24   } catch (InterruptedException e) {
25   e.printStackTrace();
26   }
27   }
28   }
29  }
```

2. 实现 Runnable 接口

```
class MyRunnable implements Runnable{
 @Override
 public void run() {
 while(true){
 System.out.println("Hello Thread!");
 try {
 Thread.sleep(1000);
 } catch (InterruptedException e) {
 e.printStackTrace();
  }
  }
  }
}

public class ThreadDemo2 {
  public static void main(String[] args) {
  Thread T = new Thread(new MyRunnable());
  T.start();
  }
}
```

3. 匿名内部类创建 Thread 子类对象

```
public class ThreadDemo3 {
 public static void main(String[] args) {
 // 这个语法是匿名内部类
 // 相当于创建了一个匿名的类，这个类是继承了 Thread
 // 此处 new 的实例，其实是 new 了这个新的子类的实例，
 Thread t = new Thread(){
 @Override
 public void run() {
 while(true){
  System.out.println("hello Thread!");
  try {
  Thread.sleep(1000);
  } catch (InterruptedException e) {
```

```
 e.printStackTrace();
 }
 }
 }
 };
 t.start();
 }
}
```

4.匿名内部类创建 Runnable 子类对象

```
public class ThreadDemo4 {
 public static void main(String[] args) {
 Thread t = new Thread(new Runnable() {
 @Override
 public void run() {
 while(true){
 System.out.println("hello Thread!");
 try {
 Thread.sleep(1000);
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 }
 }
 });
 t.start();
 }
}
```

5.lambda 表达式创建 Runnable 子类对象

```
public class ThreadDemo5 {
 public static void main(String[] args) {
 Thread t = new Thread(()->{
 while(true){
 System.out.println("hello Thread!");
 try {
 Thread.sleep(1000);
```

```
8   } catch (InterruptedException e) {
9   e.printStackTrace();
10   }
11   }
12   }) ;
13   t.start();
14   }
15  }
```

4、多线程的优势-增加运行速度

使用 System.nanoTime() 可以记录当前系统的 纳秒 级时间戳.

serial() 串行的完成一系列运算. concurrency()使用两个线程并行的完成同样的运算.

```
1  public class ThreadAdvantage {
2   private static final long count = 10_0000_0000;
3
4   private static void concurrency(){
5   long begin = System.nanoTime();
6
7   // 利用一个线程计算 a 的值
8   Thread thread = new Thread(new Runnable() {
9   @Override
10   public void run() {
11   int a = 0;
12   for(long i = 1; i < count ; i ++){
13   a--;
14   }
15   }
16   });
17   thread.start();
18   // 主线程内计算 b 的值
19   int b = 0;
20   for(long i = 1; i < count ; i ++){
21   b--;
22   }
```

```
23
24  // 等待 thread 线程运行结束
25  try {
26  thread.join();
27  } catch (InterruptedException e) {
28  e.printStackTrace();
29  }
30
31  long end = System.nanoTime();
32  double ms = (end - begin)* 1.0 / 1000 / 1000;
33  System.out.println("并发:" + ms);
34  }
35
36  private static void serial(){
37  // 全部在主线程内计算 a、b 的值
38  long begin = System.nanoTime();
39
40  int a = 0;
41  for(long i = 1; i < count ; i ++){
42  a--;
43  }
44  int b = 0;
45  for(long i = 1; i < count ; i ++){
46  b--;
47  }
48
49  long end = System.nanoTime();
50  double ms = (end - begin) * 1.0 / 1000 / 1000;
51  System.out.println("串行:" + ms);
52  }
53
54  public static void main(String[] args) {
55  // 使用并发方式
56  concurrency(); // 并发:653.6996
57  // 使用串行方式
```

```
58   serial(); // 串行:1243.4612
59   }
60 }
```