多线程

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

什么是线程？

- 线程(thread)是一个程序内部的一条执行路径。
- 我们之前启动程序执行后，main方法的执行其实就是一条单独的执行路径。

```
public static void main(String[] args) {
    //代码...
    for (int i = 0; i < 10; i++) {
        System.out.println(i);
    }
    //代码...
}
```

- 程序中如果只有一条执行路径，那么这个程序就是单线程的程序。

多线程是什么？

- 多线程是指从软硬件上实现多条执行流程的技术。

多线程用在哪里，有什么好处

- 再例如：消息通信、淘宝、京东系统都离不开多线程技术。

关于多线程同学们需要学会什么

多线程的创建	Thread类的常用方法	线程安全、线程同步	线程通信、线程池	定时器、线程状态等
如何在程序中实现多线程，有哪些方式，各自有什么优缺点。	线程的代表是Thread类，Thread提供了哪些线程的操作给我们呢？	多个线程同时访问一个共享的数据的时候会出现问题，如何去解决？	线程与线程间需要配合完成一些事情。 线程池是一种线程优化方案，可以用一种更好的方式使用多线程。	如何在程序中实现定时器？ 线程在执行的过程中会有很多不同的状态，理解这些状态有助于理解线程的执行原理，也有利于面试

目录

Contents

Thread类

- Java是通过java.lang.Thread 类来代表线程的。
- 按照面向对象的思想，Thread类应该提供了实现多线程的方式。

多线程的实现方案一：继承Thread类

① 定义一个子类MyThread继承线程类java.lang.Thread，重写run()方法

② 创建MyThread类的对象

③ 调用线程对象的start()方法启动线程（启动后还是执行run方法的）

方式一优缺点：

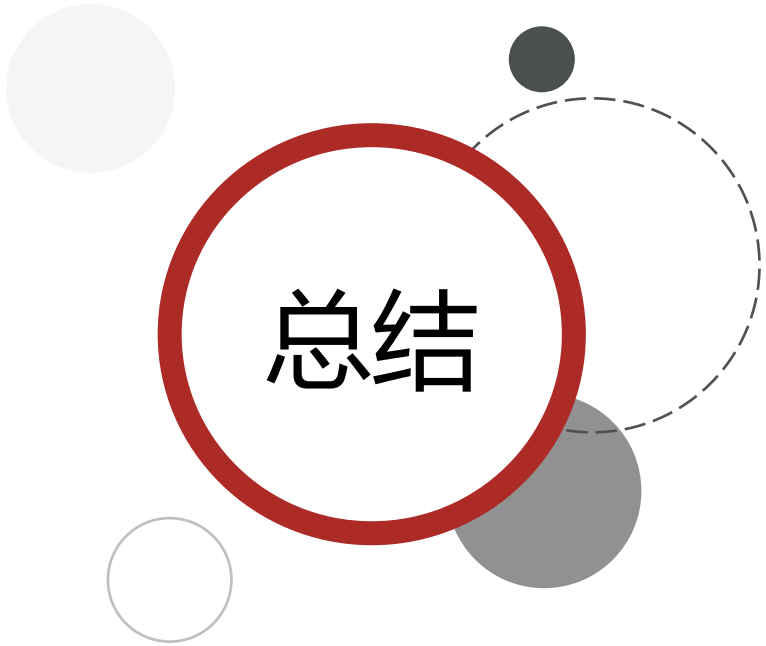
- 优点：编码简单
- 缺点：线程类已经继承Thread，无法继承其他类，不利于扩展。

1、为什么不直接调用了run方法，而是调用start启动线程。

- **直接调用run方法会当成普通方法执行，此时相当于还是单线程执行。**
- **只有调用start方法才是启动一个新的线程执行。**

2、把主线程任务放在子线程之前了。

- **这样主线程一直是先跑完的，相当于是一个单线程的效果了。**



1. 方式一是如何实现多线程的？
   - **继承Thread类**
   - **重写run方法**
   - **创建线程对象**
   - **调用start()方法启动。**
2. 优缺点是什么？
   - **优点：编码简单**
   - **缺点：存在单继承的局限性，线程类继承Thread后，不能继承其他类，不便于扩展。**

目录

Contents

多线程的实现方案二：实现Runnable接口

① 定义一个线程任务类MyRunnable实现Runnable接口，重写run()方法

② 创建MyRunnable任务对象

③ 把MyRunnable任务对象交给Thread处理。

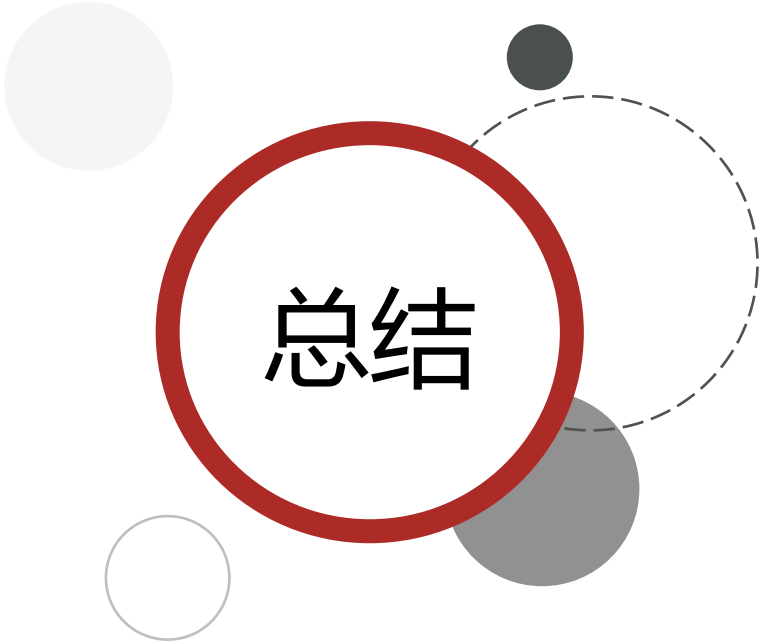
④ 调用线程对象的start()方法启动线程

Thread的构造器

构造器	说明
public Thread(String name)	可以为当前线程指定名称
public Thread(Runnable target)	封装Runnable对象成为线程对象
public Thread(Runnable target ，String name)	封装Runnable对象成为线程对象，并指定线程名称

方式二优缺点：

- 优点：线程任务类只是实现接口，可以继续继承类和实现接口，扩展性强。
- 缺点：编程多一层对象包装，如果线程有执行结果是不可以直接返回的。



1. 第二种方式是如何创建线程的？

- **定义一个线程任务类MyRunnable实现Runnable接口，重写run()方法**
- **创建MyRunnable对象**
- **把MyRunnable任务对象交给Thread线程对象处理。**
- **调用线程对象的start()方法启动线程**

2. 第二种方式的优点

- **优点：线程任务类只是实现了Runnale接口，可以继续继承和实现。**
- **缺点：如果线程有执行结果是不能直接返回的。**

多线程的实现方案二：实现Runnable接口(匿名内部类形式)

① 可以创建Runnable的匿名内部类对象。

② 交给Thread处理。

③ 调用线程对象的start()启动线程。

目录

Contents

1、前2种线程创建方式都存在一个问题：

- **他们重写的run方法均不能直接返回结果。**
- **不适合需要返回线程执行结果的业务场景。**

2、怎么解决这个问题呢？

- **JDK 5.0提供了Callable和FutureTask来实现。**
- **这种方式的优点是：可以得到线程执行的结果。**

多线程的实现方案三：利用Callable、FutureTask接口实现。

① 、得到任务对象

1. 定义类实现Callable接口，重写call方法，封装要做的事情。
2. 用FutureTask把Callable对象封装成线程任务对象。

② 、把线程任务对象交给Thread处理。

③ 、调用Thread的start方法启动线程，执行任务

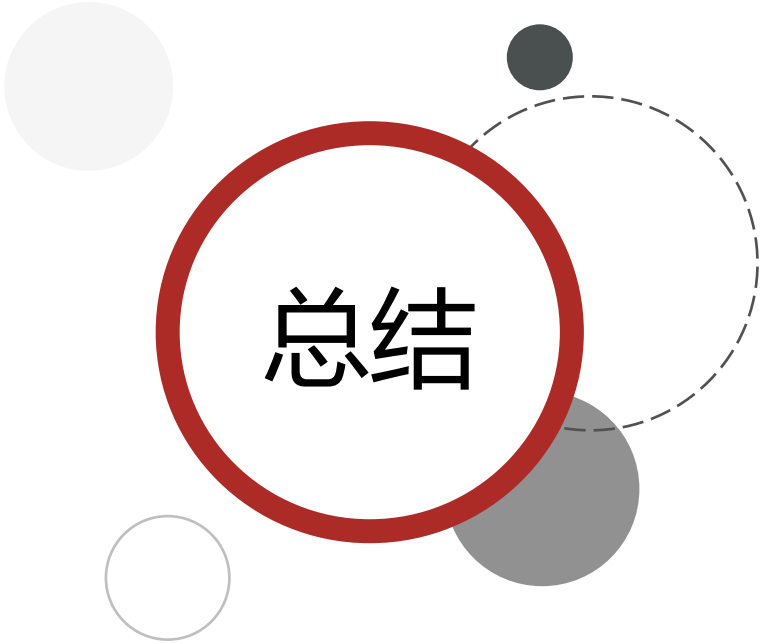
④ 、线程执行完毕后、通过FutureTask的get方法去获取任务执行的结果。

FutureTask的API

方法名称	说明
public FutureTask<>(Callable call)	把Callable对象封装成FutureTask对象。
public V get() throws Exception	获取线程执行call方法返回的结果。

方式三优缺点：

- 优点：线程任务类只是实现接口，可以继续继承类和实现接口，扩展性强。
- **可以在线程执行完毕后去获取线程执行的结果。**
- 缺点：编码复杂一点。



1. 3种方式对比

方式	优点	缺点
继承Thread类	编程比较简单，可以直接使用Thread类中的方法	扩展性较差，不能再继承其他的类，不能返回线程执行的结果
实现Runnable接口	扩展性强，实现该接口的同时还可以继承其他的类。	编程相对复杂，不能返回线程执行的结果
实现Callable接口	扩展性强，实现该接口的同时还可以继承其他的类。可以得到线程执行的结果	编程相对复杂

目录

Contents

Thread常用API说明

- Thread常用方法：获取线程名称getName()、设置名称setName()、获取当前线程对象currentThread()。
- 至于Thread类提供的诸如：yield、join、interrupt、不推荐的方法 stop 、守护线程、线程优先级等线程的控制方法，在开发中很少使用，这些方法会在高级篇以及后续需要用到的时候再为大家讲解。

1. 当有很多线程在执行的时候，我们怎么去区分这些线程呢？

- **此时需要使用Thread的常用方法：getName()、setName()、currentThread()等。**

Thread获取和设置线程名称

方法名称	说明
`String getName ()`	获取当前线程的名称，默认线程名称是Thread-索引
`void setName (String name)`	将此线程的名称更改为指定的名称，通过构造器也可以设置线程名称

Thread类获得当前线程的对象

方法名称	说明
public static Thread currentThread() :	返回对当前正在执行的线程对象的引用

注意

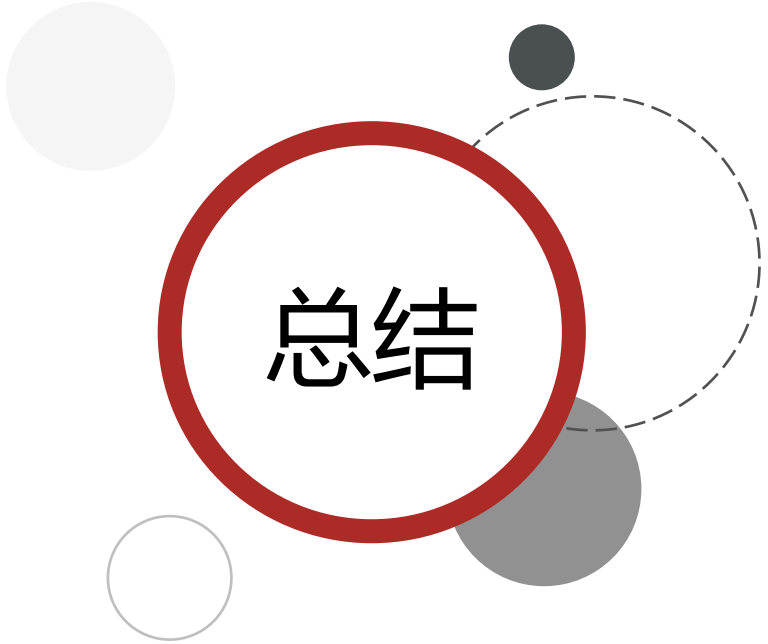
- 1、此方法是Thread类的静态方法，可以直接使用Thread类调用。
- 2、这个方法是在哪个线程执行中调用的，就会得到哪个线程对象。

Thread的构造器

方法名称	说明
public Thread(String name)	可以为当前线程指定名称
public Thread(Runnable target)	封装Runnable对象成为线程对象
public Thread(Runnable target , String name)	封装Runnable对象成为线程对象，并指定线程名称

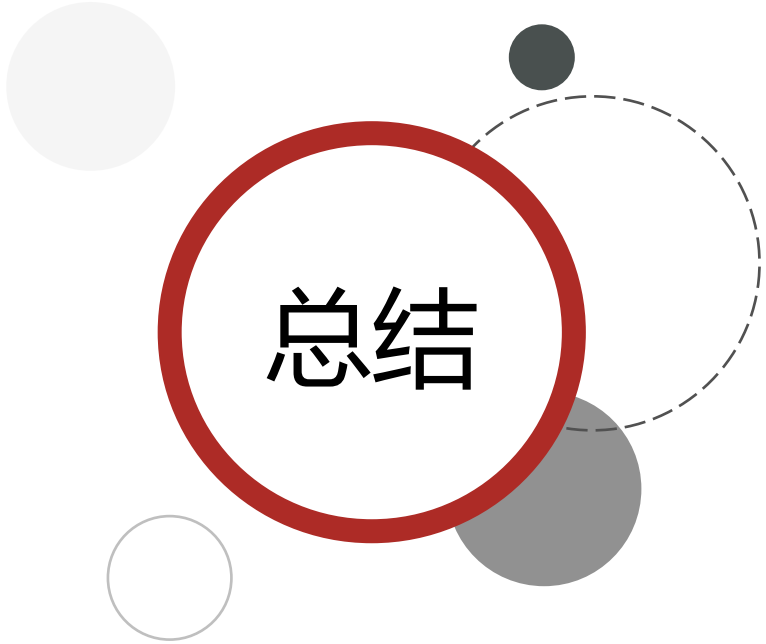
Thread类的线程休眠方法

方法名称	说明
public static void sleep(long time)	让当前线程休眠指定的时间后再继续执行，单位为毫秒。



1. Thread常用方法、构造器

方法名称	说明
String getName ()	获取当前线程的名称，默认线程名称是Thread-索引
void setName (String name)	设置线程名称
public static Thread currentThread()：	返回对当前正在执行的线程对象的引用
public static void sleep(long time)	让线程休眠指定的时间，单位为毫秒。
public void run()	线程任务方法
public void start()	线程启动方法



2. Thread常用方法、构造器

构造器	说明
public Thread(String name)	可以为当前线程指定名称
public Thread(Runnable target)	把Runnable对象交给线程对象
public Thread(Runnable target ，String name)	把Runnable对象交给线程对象，并指定线程名称

目录

Contents

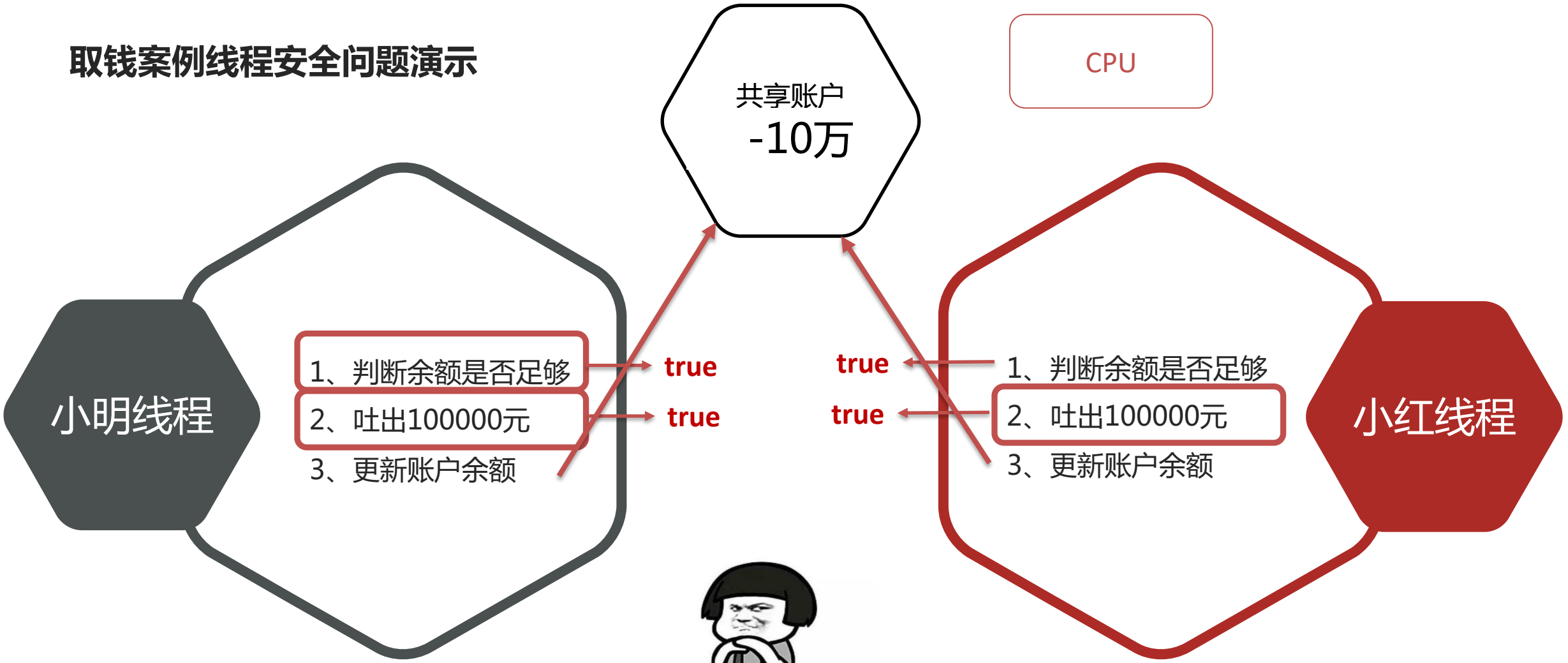
线程安全问题

- 多个线程同时操作同一个共享资源的时候可能会出现业务安全问题，称为线程安全问题。

取钱模型演示

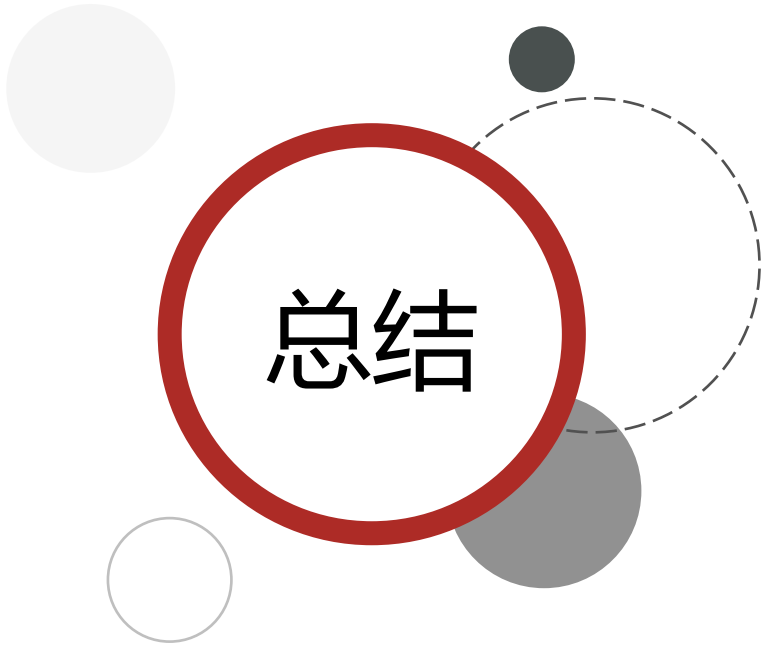
- 需求：小明和小红是一对夫妻，他们有一个共同的账户，余额是10万元。
- 如果小明和小红同时来取钱，而且2人都要取钱10万元，可能出现什么问题呢？

取钱案例线程安全问题演示



我有个大胆的想法

结果：2人都取钱10万，银行亏了10万。



1. 线程安全问题出现的原因？

- **存在多线程并发**
- **同时访问共享资源**
- **存在修改共享资源**

目录

Contents

案例

取钱业务

需求：

- 小明和小红是一对夫妻，他们有一个共同的账户，余额是10万元，模拟2人同时去取钱10万。

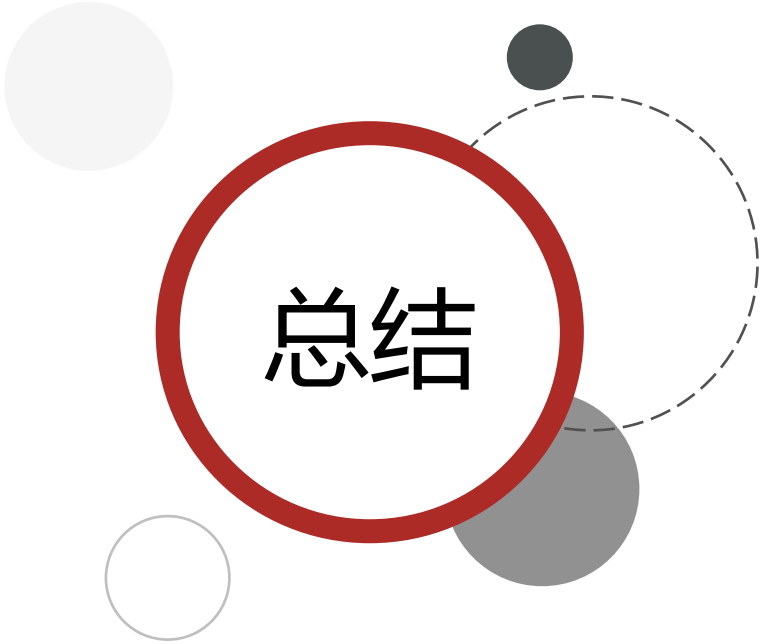
分析：

①：需要提供一个账户类，创建一个账户对象代表2个人的共享账户。

②：需要定义一个线程类，线程类可以处理账户对象。

③：创建2个线程对象，传入同一个账户对象。

④：启动2个线程，去同一个账户对象中取钱10万。



1. 线程安全问题发生的原因是什么？

- **多个线程同时访问同一个共享资源且存在修改该资源。**

目录

Contents

线程同步

- 为了解决线程安全问题。

1、取钱案例出现问题的原因？

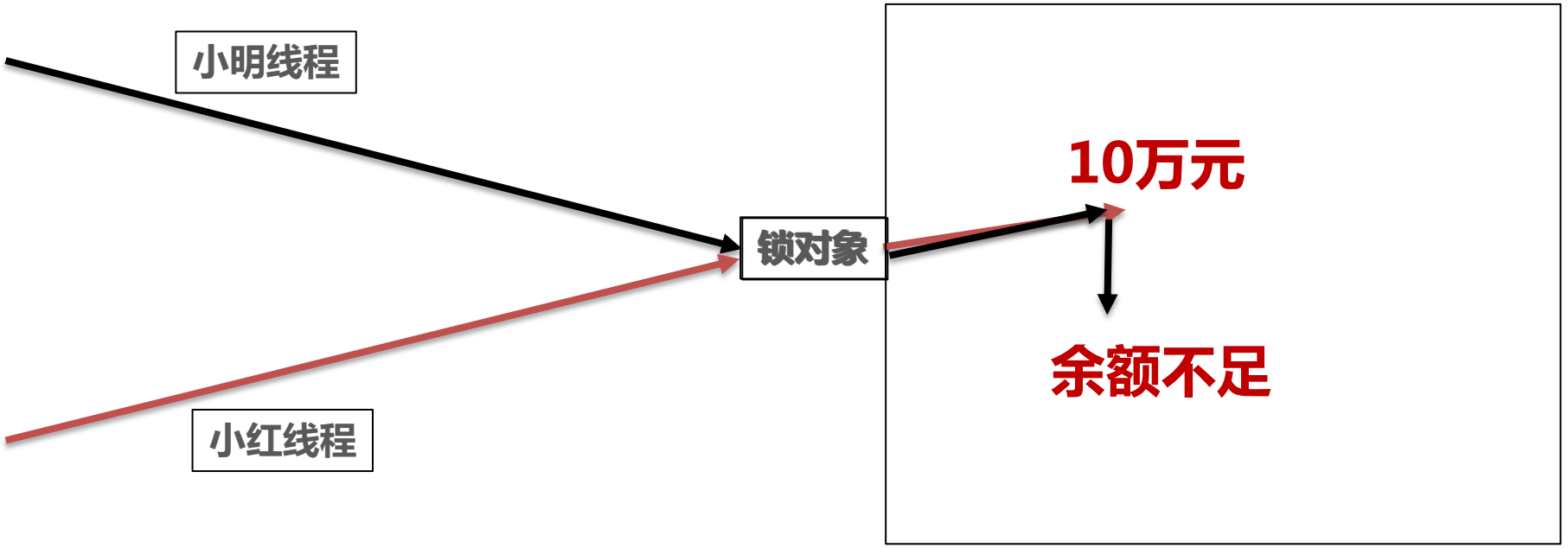
- **多个线程同时执行，发现账户都是够钱的。**

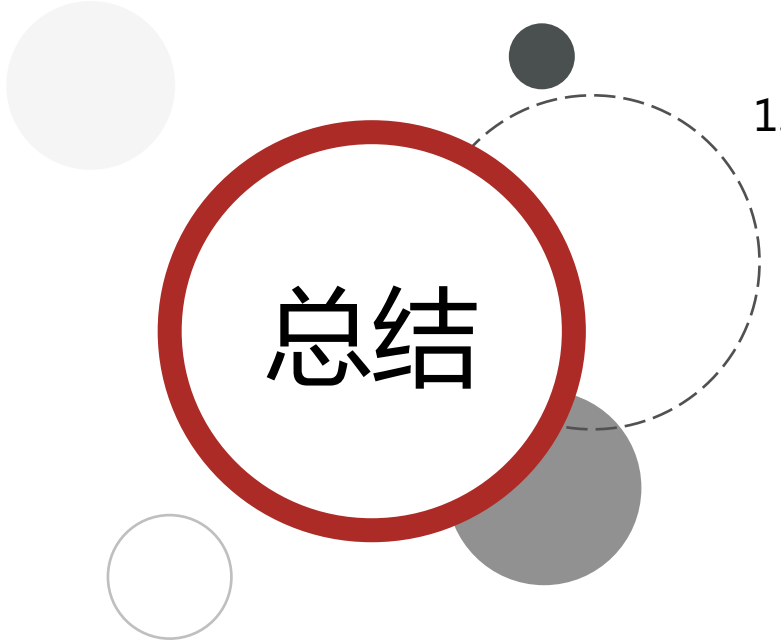
2、如何才能保证线程安全呢？

- **让多个线程实现先后依次访问共享资源，这样就解决了安全问题**

线程同步的核心思想

- **加锁，**把共享资源进行上锁，每次只能一个线程进入访问完毕以后解锁，然后其他线程才能进来。





1. 线程同步解决安全问题的思想是什么？

- **加锁：让多个线程实现先后依次访问共享资源，这样就解决了安全问题。**

目录

Contents

同步代码块

- 作用：把出现线程安全问题的核心代码给上锁。
- 原理：每次只能一个线程进入，执行完毕后自动解锁，其他线程才可以进来执行。

```
synchronized(同步锁对象) {
        操作共享资源的代码(核心代码)
}
```

锁对象要求

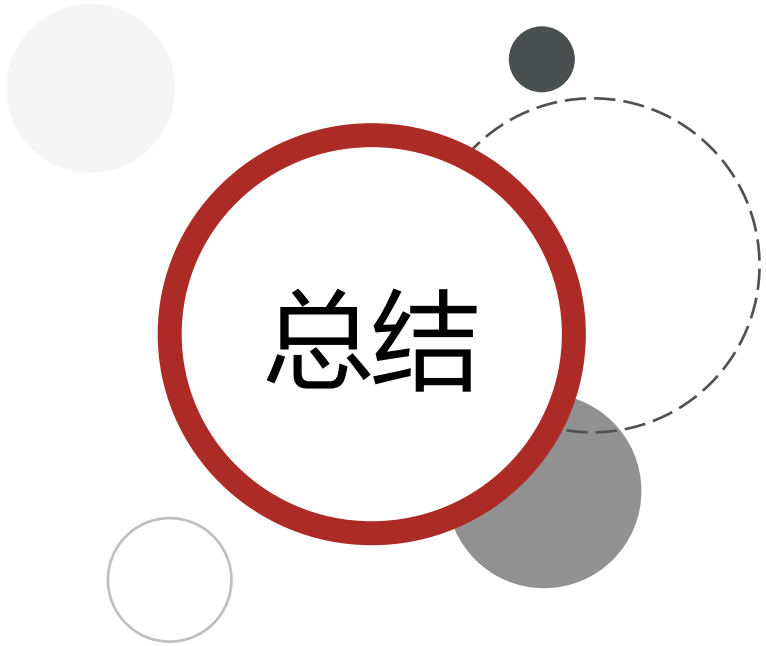
- 理论上：锁对象只要对于当前同时执行的线程来说是同一个对象即可。

锁对象用任意唯一的对象好不好呢?

- 不好，会影响其他无关线程的执行。

锁对象的规范要求

- 规范上：建议使用共享资源作为锁对象。
- 对于实例方法建议使用this作为锁对象。
- 对于静态方法建议使用字节码（类名.class）对象作为锁对象。



1. 同步代码块是如何实现线程安全的？
   - **对出现问题的核心代码使用synchronized进行加锁**
   - **每次只能一个线程占锁进入访问**
2. 同步代码块的同步锁对象有什么要求？
   - **对于实例方法建议使用this作为锁对象。**
   - **对于静态方法建议使用字节码（类名.class）对象作为锁对象。**

目录

Contents

同步方法

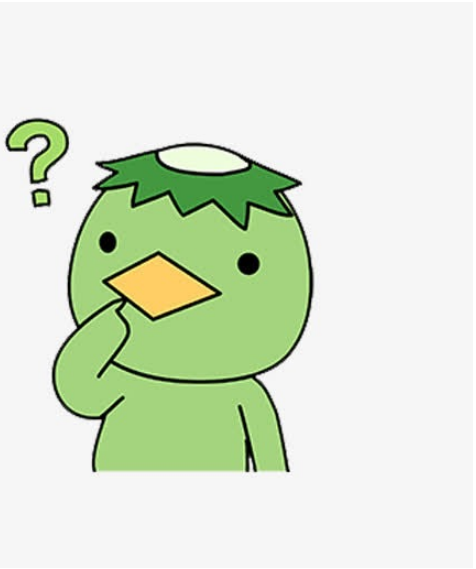
- 作用：把出现线程安全问题的核心方法给上锁。
- 原理：每次只能一个线程进入，执行完毕以后自动解锁，其他线程才可以进来执行。

格式

```
修饰符 synchronized 返回值类型 方法名称(形参列表) {
        操作共享资源的代码
}
```

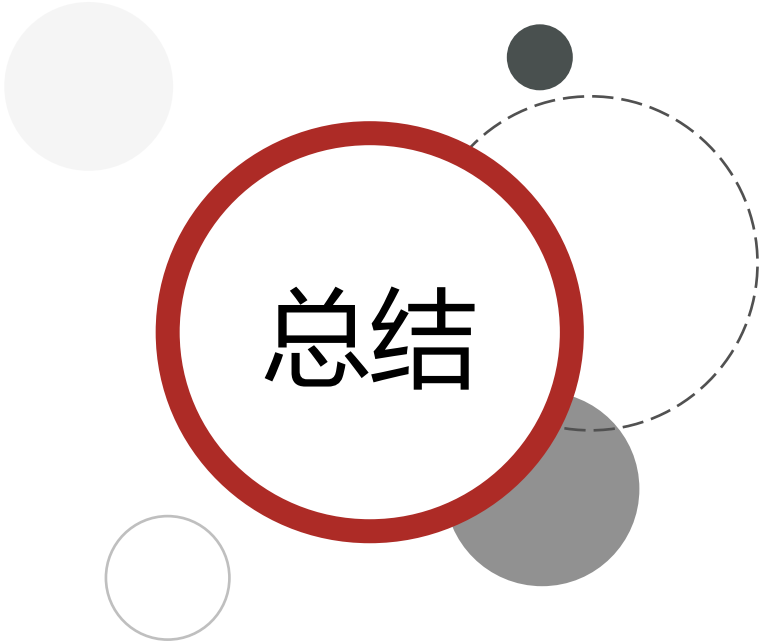
同步方法底层原理

- 同步方法其实底层也是有隐式锁对象的，只是锁的范围是整个方法代码。
- 如果方法是实例方法：同步方法默认用this作为的锁对象。但是代码要高度面向对象！
- 如果方法是静态方法：同步方法默认用类名.class作为的锁对象。



1、是同步代码块好还是同步方法好一点？

- **同步代码块锁的范围更小，同步方法锁的范围更大。**



1. 同步方法是如何保证线程安全的？
   - **对出现问题的核心方法使用synchronized修饰**
   - **每次只能一个线程占锁进入访问**

2. 同步方法的同步锁对象的原理？
   - **对于实例方法默认使用this作为锁对象。**
   - **对于静态方法默认使用类名.class对象作为锁对象。**

目录

Contents

Lock锁

- 为了更清晰的表达如何加锁和释放锁，JDK5以后提供了一个新的锁对象Lock，更加灵活、方便。
- Lock实现提供比使用synchronized方法和语句可以获得更广泛的锁定操作。
- Lock是接口不能直接实例化，这里采用它的实现类ReentrantLock来构建Lock锁对象。

方法名称	说明
public ReentrantLock()	获得Lock锁的实现类对象

Lock的API

方法名称	说明
void lock()	获得锁
void unlock()	释放锁

目录

Contents

什么是线程通信、如何实现？

- 所谓线程通信就是线程间相互发送数据，线程间共享一个资源即可实现线程通信。

线程通信常见形式

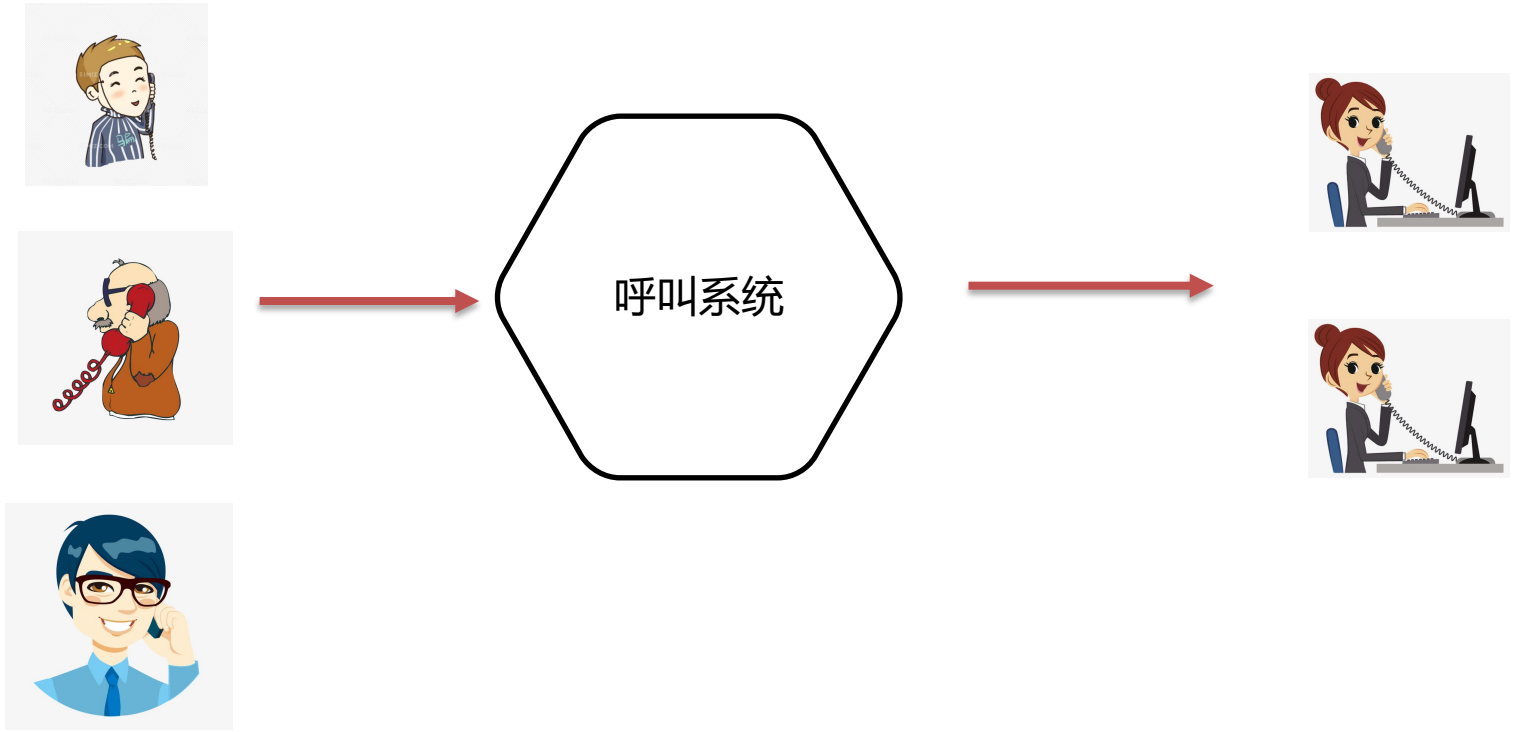
- 通过共享一个数据的方式实现。
- 根据共享数据的情况决定自己该怎么做，以及通知其他线程怎么做。

线程通信实际应用场景

- 生产者与消费者模型：生产者线程负责生产数据，消费者线程负责消费生产者产生的数据。
- 要求：生产者线程生产完数据后唤醒消费者，然后等待自己，消费者消费完该数据后唤醒生产者，然后等待自己。

线程通信案例模拟

- 模拟客服系统，系统可以不断的接入电话 和 分发给客服。



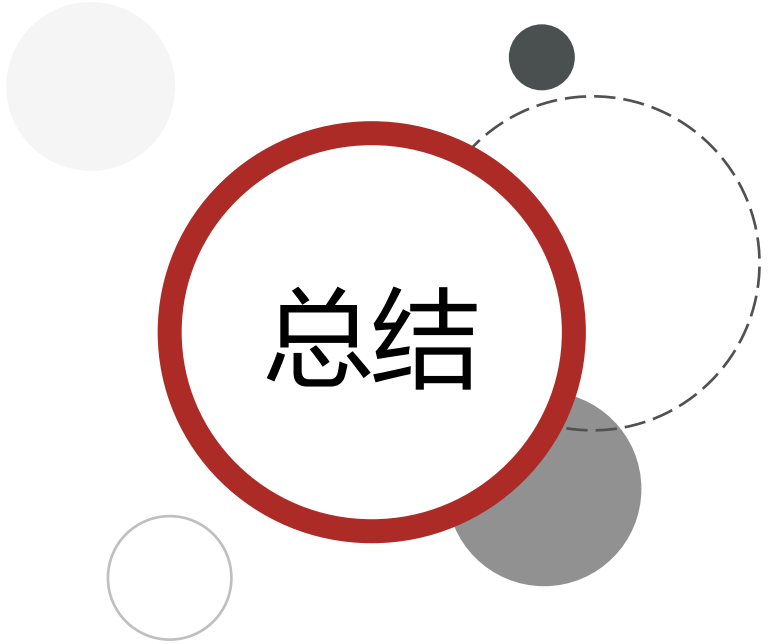
- 线程通信的前提：线程通信通常是在多个线程操作同一个共享资源的时候需要进行通信，且要保证线程安全。

Object类的等待和唤醒方法：

方法名称	说明
void wait ()	让当前线程等待并释放所占锁，直到另一个线程调用notify()方法或 notifyAll()方法
void notify ()	唤醒正在等待的单个线程
void notifyAll ()	唤醒正在等待的所有线程

注意

- 上述方法应该使用当前同步锁对象进行调用。



1. 线程通信的三个常见方法

方法名称	说明
void wait ()	当前线程等待，直到另一个线程调用notify() 或 notifyAll()唤醒自己
void notify ()	唤醒正在等待对象监视器(锁对象)的单个线程
void notifyAll ()	唤醒正在等待对象监视器(锁对象)的所有线程

注意

- 上述方法应该使用当前同步锁对象进行调用。

目录

Contents

什么是线程池？

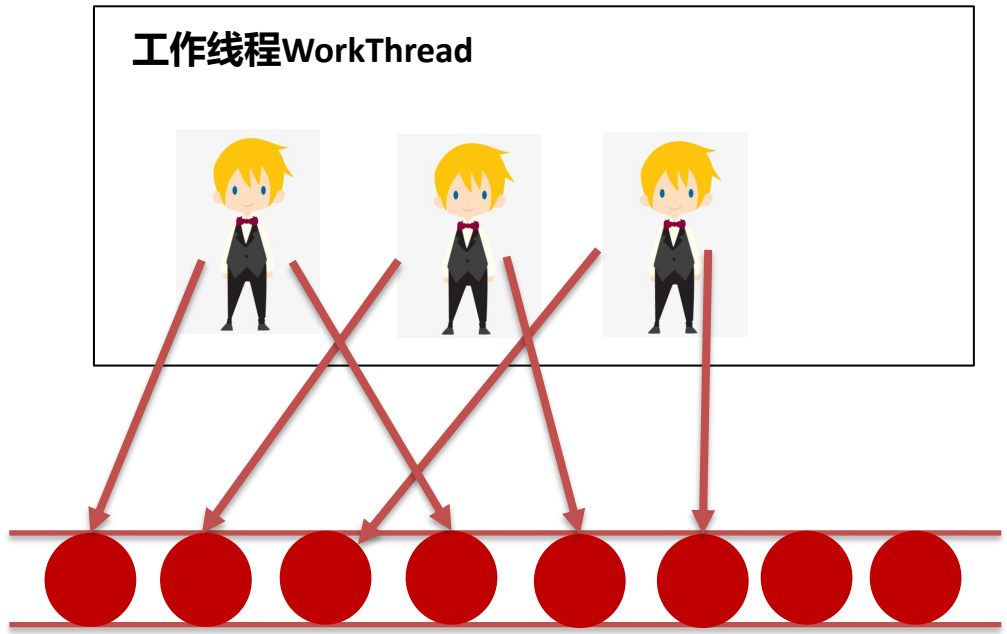
- 线程池就是一个可以复用线程的技术。

不使用线程池的问题

- 如果用户每发起一个请求，后台就创建一个新线程来处理，下次新任务来了又要创建新线程，而创建新线程的开销是很大的，这样会严重影响系统的性能。

线程池的工作原理

线程池



任务接口

- **Runnable**
- **Callable**

目录

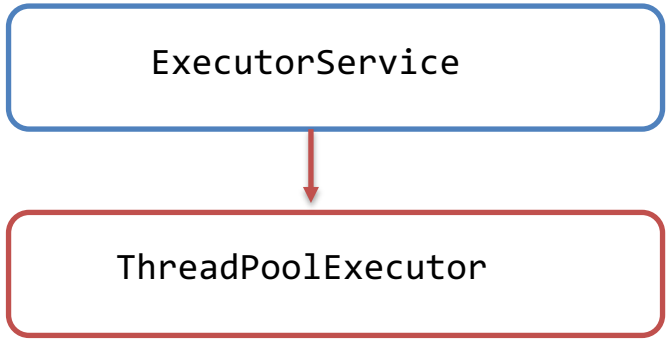
Contents

谁代表线程池？

- JDK 5.0起提供了代表线程池的接口：ExecutorService

如何得到线程池对象

- 方式一：使用ExecutorService的实现类ThreadPoolExecutor自创建一个线程池对象



- 方式二：使用Executors（线程池的工具类）调用方法返回不同特点的线程池对象

ThreadPoolExecutor构造器的参数说明

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory,
                          RejectedExecutionHandler handler)
```

参数一：指定线程池的线程数量（核心线程）： corePoolSize → 不能小于0

参数二：指定线程池可支持的最大线程数： maximumPoolSize → 最大数量 >= 核心线程数量

参数三：指定临时线程的最大存活时间： keepAliveTime → 不能小于0

参数四：指定存活时间的单位(秒、分、时、天)： unit → 时间单位

参数五：指定任务队列： workQueue → 不能为null

参数六：指定用哪个线程工厂创建线程： threadFactory → 不能为null

参数七：指定线程忙，任务满的时候，新任务来了怎么办： handler → 不能为null

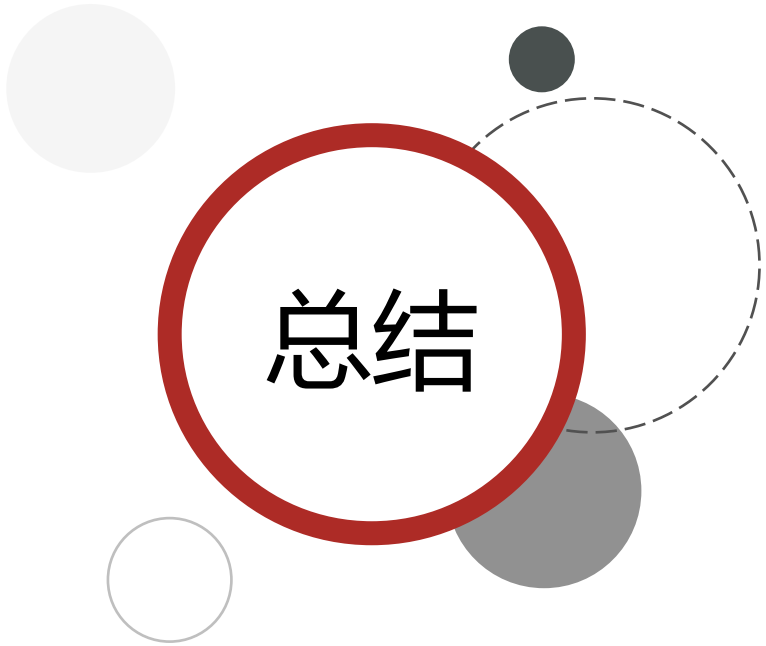
线程池常见面试题

临时线程什么时候创建啊？

- 新任务提交时发现核心线程都在忙，任务队列也满了，并且还可以创建临时线程，此时才会创建临时线程。

什么时候会开始拒绝任务？

- 核心线程和临时线程都在忙，任务队列也满了，新的任务过来的时候才会开始任务拒绝。



1. 谁代表线程池？
   - **ExecutorService接口**
2. ThreadPoolExecutor实现线程池对象的七个参数是什么意思
   - **使用线程池的实现类ThreadPoolExecutor**

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory,
                          RejectedExecutionHandler handler)
```

目录

Contents

ThreadPoolExecutor创建线程池对象示例

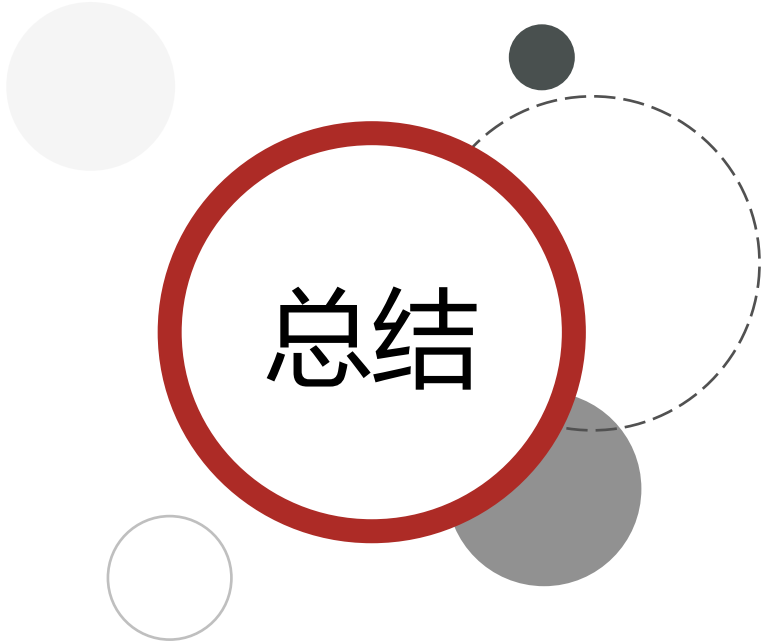
```
ExecutorService pools = new ThreadPoolExecutor(3, 5
        , 8 , TimeUnit.SECONDS, new ArrayBlockingQueue<>(6),
            Executors.defaultThreadFactory() , new ThreadPoolExecutor.AbortPolicy());
```

ExecutorService的常用方法

方法名称	说明
void execute(Runnable command)	执行任务/命令，没有返回值，一般用来执行 Runnable 任务
Future<T> submit(Callable<T> task)	执行任务，返回未来任务对象获取线程结果，一般拿来执行 Callable 任务
void shutdown()	等任务执行完毕后关闭线程池
List<Runnable> shutdownNow()	立刻关闭，停止正在执行的任务，并返回队列中未执行的任务

新任务拒绝策略

策略	详解
ThreadPoolExecutor.AbortPolicy	丢弃任务并抛出RejectedExecutionException异常。**是默认的策略**
ThreadPoolExecutor.DiscardPolicy：	丢弃任务，但是不抛出异常 这是不推荐的做法
ThreadPoolExecutor.DiscardOldestPolicy	抛弃队列中等待最久的任务 然后把当前任务加入队列中
ThreadPoolExecutor.CallerRunsPolicy	由主线程负责调用任务的run()方法从而绕过线程池直接执行



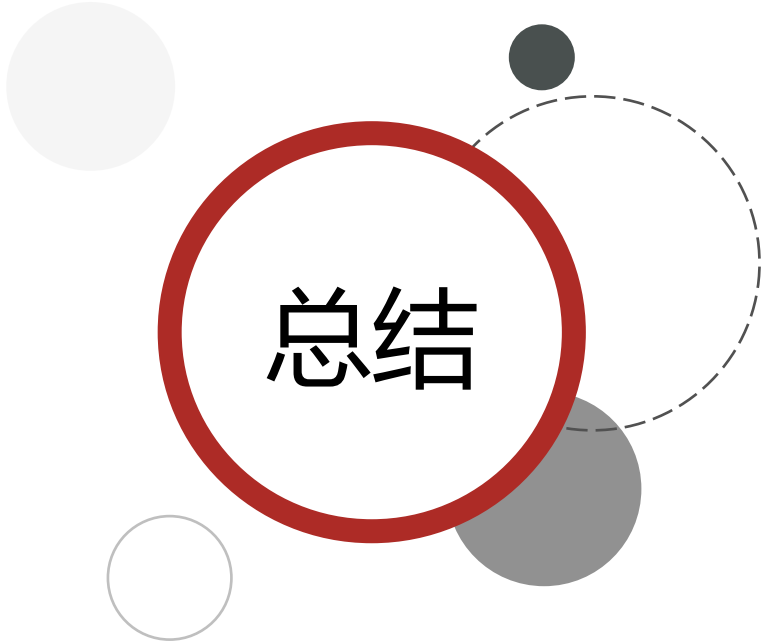
1. 线程池如何处理Runnable任务
   - **使用ExecutorService的方法：**
   - **void execute(Runnable target)**

目录

Contents

ExecutorService的常用方法

方法名称	说明
void execute(Runnable command)	执行任务/命令，没有返回值，一般用来执行 Runnable 任务
Future<T> submit(Callable<T> task)	执行Callable任务，返回未来任务对象获取线程结果
void shutdown()	等任务执行完毕后关闭线程池
List<Runnable> shutdownNow()	立刻关闭，停止正在执行的任务，并返回队列中未执行的任务



1. 线程池如何处理Callable任务，并得到任务执行完后返回的结果。
   - **使用ExecutorService的方法：**
   - **Future<T> submit(Callable<T> command)**

目录

Contents

Executors得到线程池对象的常用方法

- Executors：线程池的工具类通过调用方法返回不同类型的线程池对象。

方法名称	说明
public static ExecutorService newCachedThreadPool()	线程数量随着任务增加而增加，如果线程任务执行完毕且空闲了一段时间则会被回收掉。
public static ExecutorService newFixedThreadPool (int nThreads)	创建固定线程数量的线程池，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程替代它。
public static ExecutorService newSingleThreadExecutor ()	创建只有一个线程的线程池对象，如果该线程出现异常而结束，那么线程池会补充一个新线程。
public static ScheduledExecutorService newScheduledThreadPool (int corePoolSize)	创建一个线程池，可以实现在给定的延迟后运行任务，或者定期执行任务。

注意：Executors的底层其实也是基于线程池的实现类ThreadPoolExecutor创建线程池对象的。

Executors使用可能存在的陷阱

- 大型并发系统环境中使用Executors如果不注意可能会出现系统风险。

方法名称	存在问题
`public static ExecutorService newFixedThreadPool (int nThreads)`	允许请求的任务队列长度是Integer.MAX_VALUE，可能出现OOM错误（ java.lang.OutOfMemoryError ）
`public static ExecutorService newSingleThreadExecutor()`	
`public static ExecutorService newCachedThreadPool()`	创建的线程数量最大上限是Integer.MAX_VALUE，线程数可能会随着任务1:1增长，也可能出现OOM错误（ java.lang.OutOfMemoryError ）
`public static ScheduledExecutorService newScheduledThreadPool (int corePoolSize)`	

Executors使用可能存在的陷阱

- 大型并发系统环境中使用Executors如果不注意可能会出现系统风险。

阿里巴巴 Java 开发手册

4．【强制】线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。

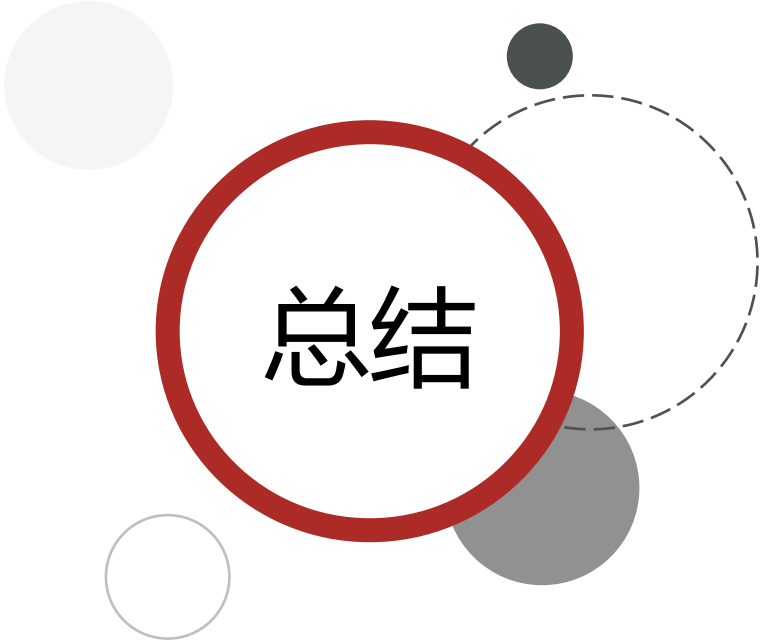
说明：Executors 返回的线程池对象的弊端如下：

1）FixedThreadPool 和 SingleThreadPool:

允许的请求队列长度为 Integer.MAX_VALUE，可能会堆积大量的请求，从而导致 OOM。

2）CachedThreadPool 和 ScheduledThreadPool:

允许的创建线程数量为 Integer.MAX_VALUE，可能会创建大量的线程，从而导致 OOM。



1. Executors工具类底层是基于什么方式实现的线程池对象？
   - **线程池ExecutorService的实现类：ThreadPoolExecutor**

2. Executors是否适合做大型互联网场景的线程池方案？
   - **不合适。**
   - **建议使用ThreadPoolExecutor来指定线程池参数，这样可以明确线程池的运行规则，规避资源耗尽的风险。**

目录

Contents

定时器

- 定时器是一种控制任务延时调用，或者周期调用的技术。
- 作用：闹钟、定时邮件发送。

定时器的实现方式

- 方式一：Timer
- 方式二： ScheduledExecutorService

Timer定时器

构造器	说明
public Timer()	创建Timer定时器对象

方法	说明
public void schedule (TimerTask task, long delay, long period)	开启一个定时器，按照计划处理TimerTask任务

Timer定时器的特点和存在的问题

1、Timer是单线程，处理多个任务按照顺序执行，存在延时与设置定时器的时间有出入。

2、可能因为其中的某个任务的异常使Timer线程死掉，从而影响后续任务执行。

ScheduledExecutorService定时器

- ScheduledExecutorService是 jdk1.5中引入了并发包，目的是为了弥补Timer的缺陷, ScheduledExecutorService内部为线程池。

Executors的方法	说明
`public static ScheduledExecutorService newScheduledThreadPool(int corePoolSize)`	得到线程池对象

ScheduledExecutorService的方法	说明
`public ScheduledFuture<?> scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)`	周期调度方法

ScheduledExecutorService的优点

1、基于线程池，某个任务的执行情况不会影响其他定时任务的执行。

目录

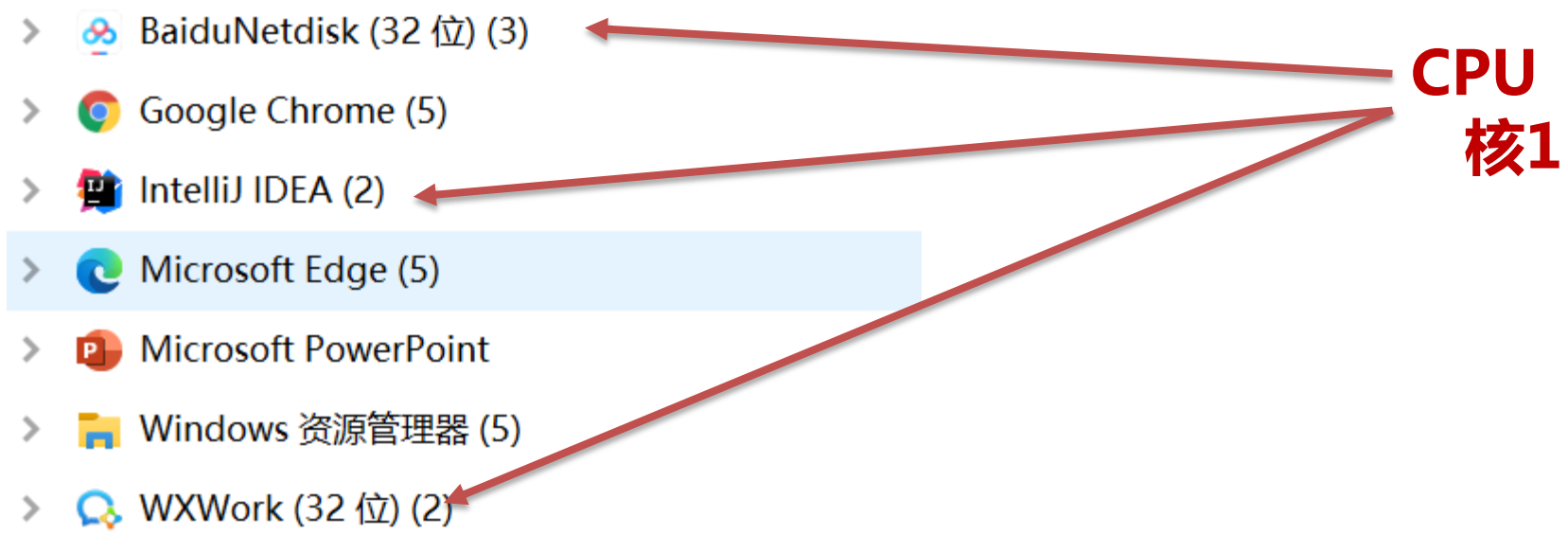
Contents

并发与并行

- 正在运行的程序（软件）就是一个独立的进程， 线程是属于进程的，多个线程其实是并发与并行同时进行的。

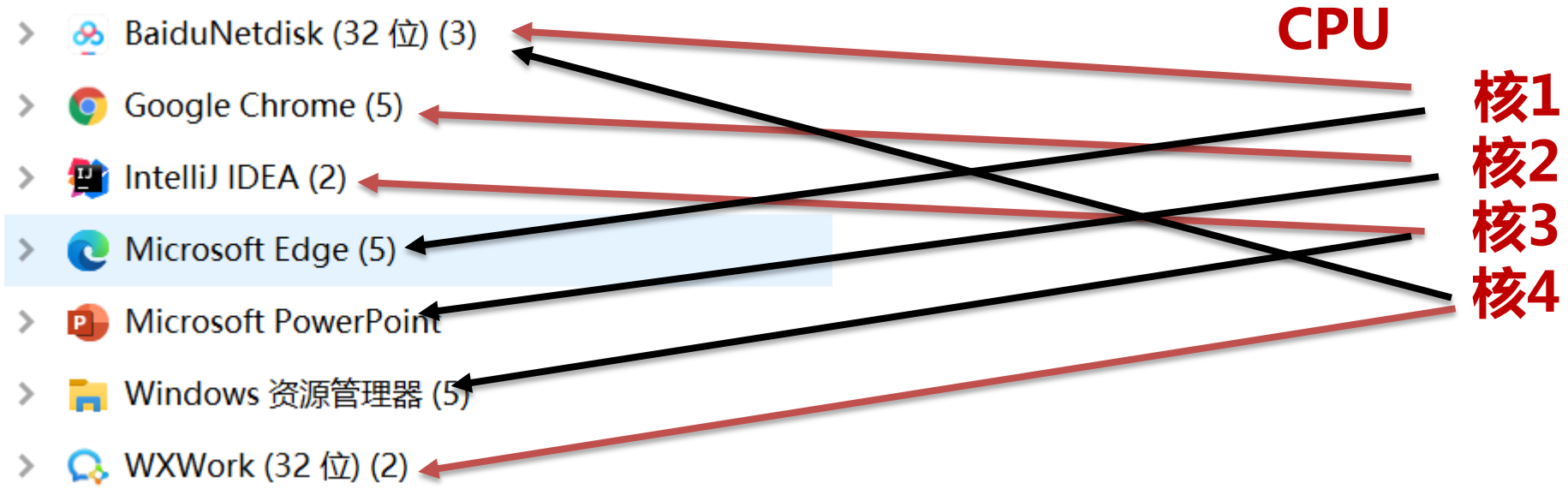
并发的理解：

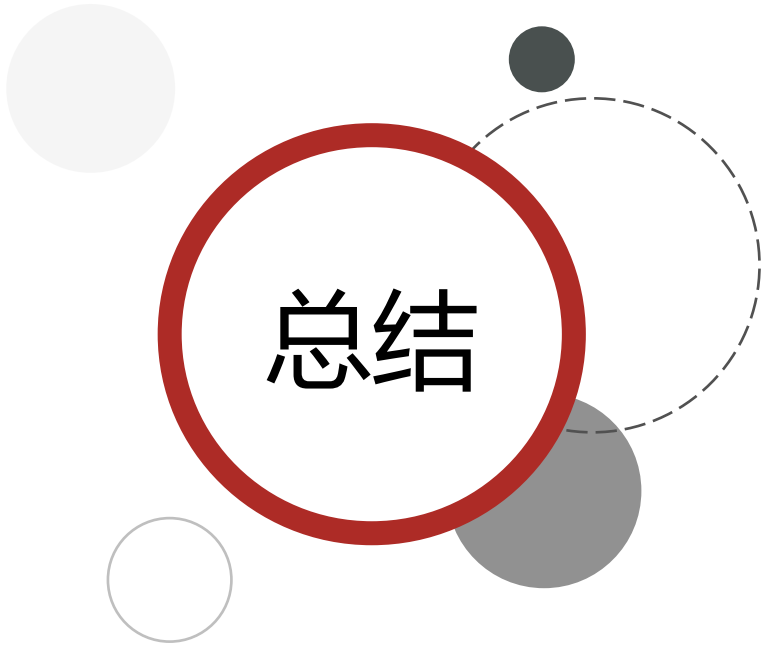
- CPU同时处理线程的数量有限。
- CPU会轮询为系统的每个线程服务，由于CPU切换的速度很快，给我们的感觉这些线程在同时执行，这就是并发。



并行的理解：

- 在同一个时刻上，同时有多个线程在被CPU处理并执行。





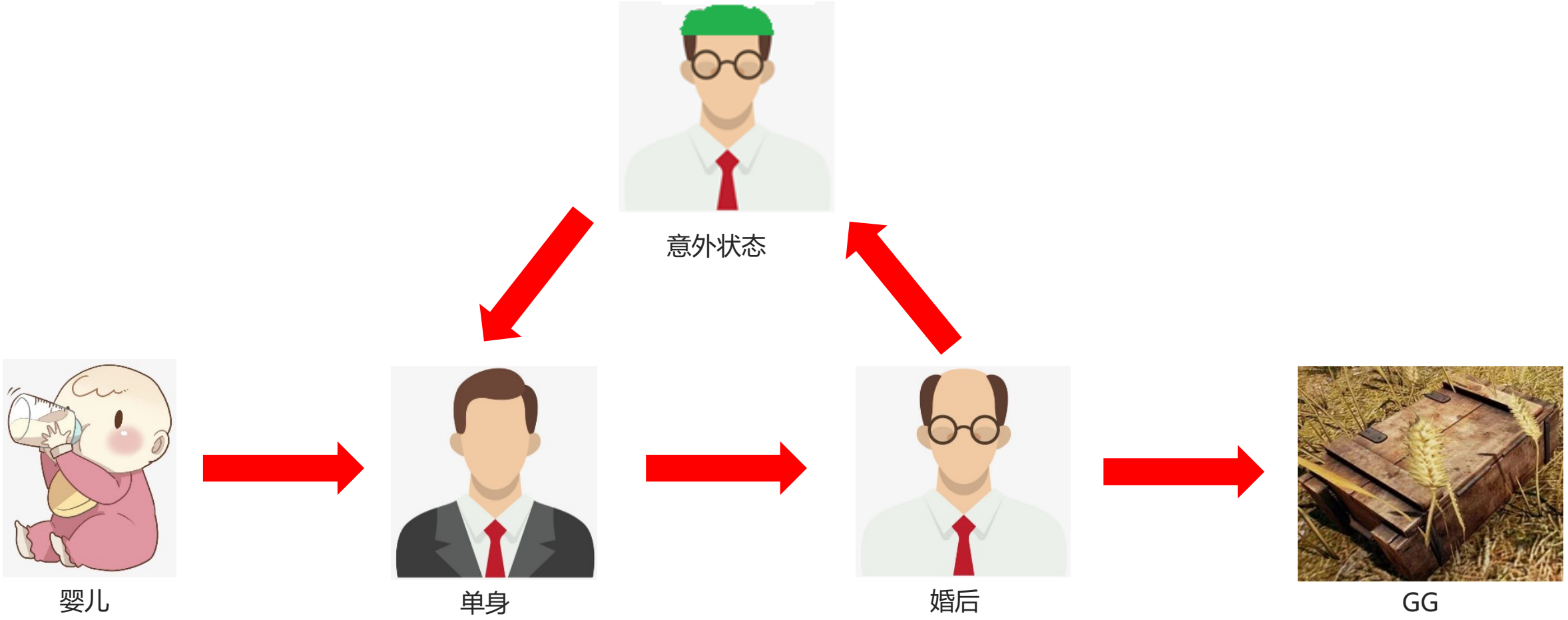
1. 简单说说并发和并行的含义

- **并发：CPU分时轮询的执行线程。**
- **并行：同一个时刻同时在执行。**

目录

Contents

状态



线程的状态

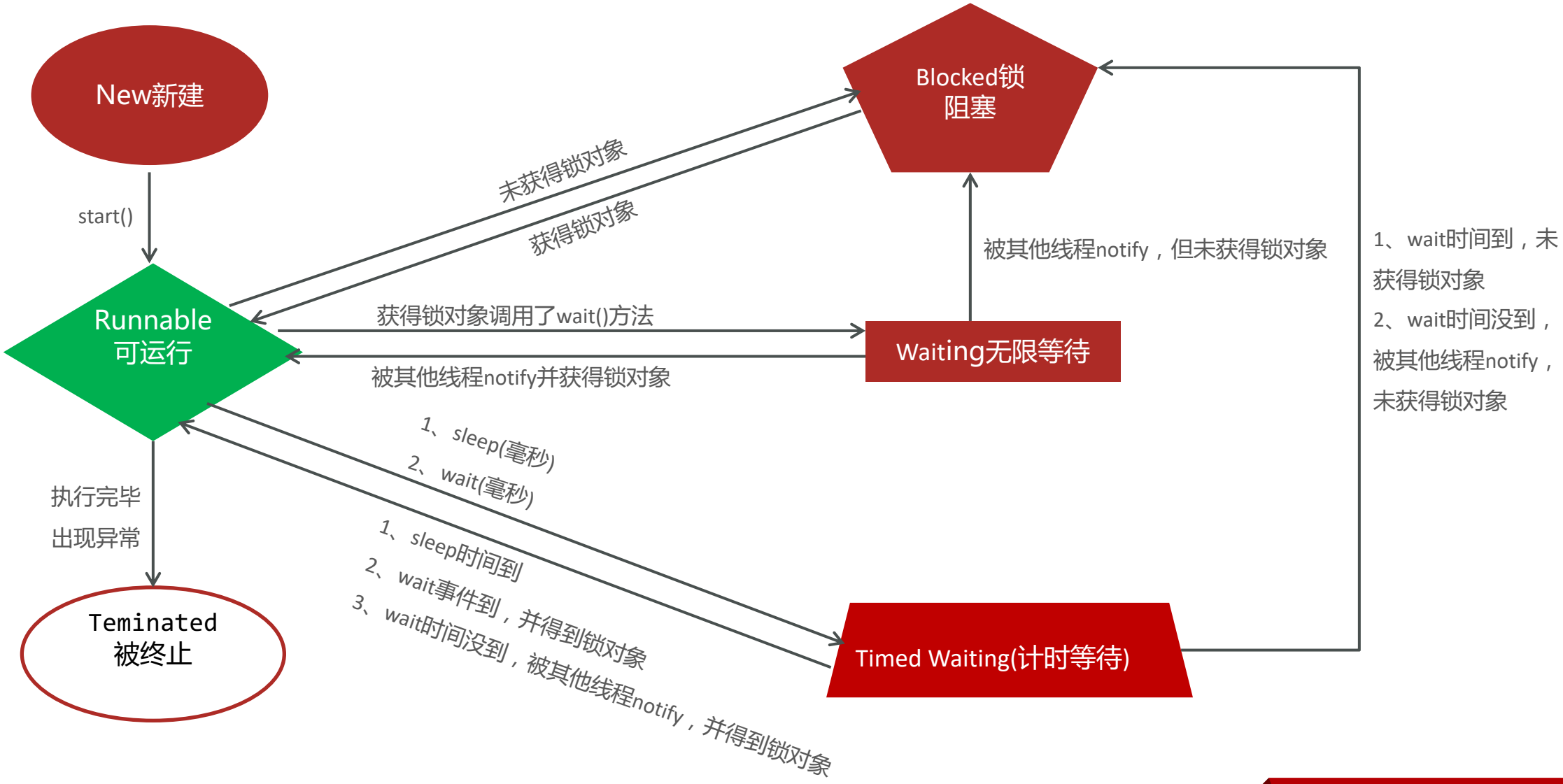
- 线程的状态：也就是线程从生到死的过程，以及中间经历的各种状态及状态转换。
- 理解线程的状态有利于提升并发编程的理解能力。

Java线程的状态

- Java总共定义了6种状态
- 6种状态都定义在Thread类的内部枚举类中。

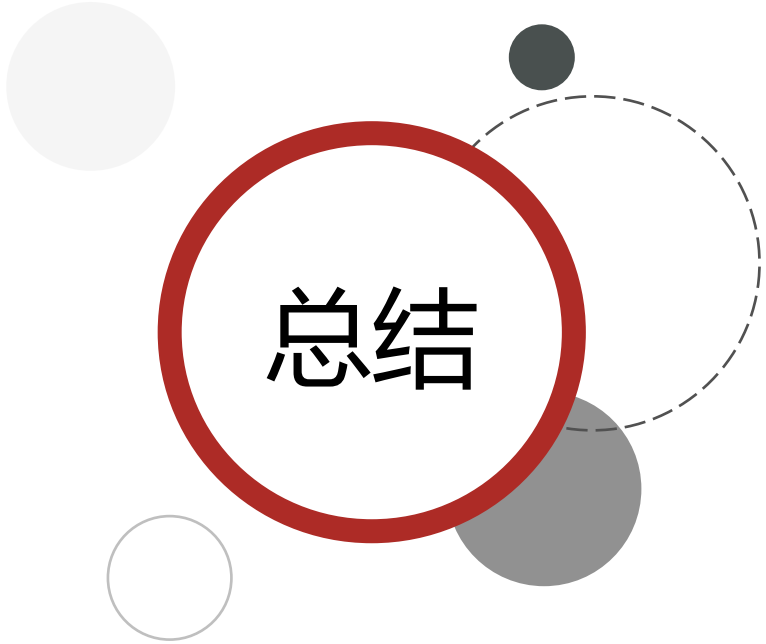
```
public class Thread{
      ...
      public enum State {
            NEW,
            RUNNABLE,
            BLOCKED,
            WAITING,
            TIMED_WAITING,
            TERMINATED;
      }
      ...
}
```

线程的6种状态互相转换



线程的6种状态总结

线程状态	描述
NEW(新建)	线程刚被创建，但是并未启动。
Runnable(可运行)	线程已经调用了start()等待CPU调度
Blocked(锁阻塞)	线程在执行的时候未竞争到锁对象，则该线程进入Blocked状态；。
Waiting(无限等待)	一个线程进入Waiting状态，另一个线程调用notify或者notifyAll方法才能够唤醒
Timed Waiting(计时等待)	同waiting状态，有几个方法有超时参数，调用他们将进入Timed Waiting状态。带有超时参数的常用方法有Thread.sleep 、Object.wait。
Teminated(被终止)	因为run方法正常退出而死亡，或者因为没有捕获的异常终止了run方法而死亡。



线程的六种状态：

新建状态（ NEW ）	→	创建线程对象
就绪状态（ RUNNABLE ）	→	start方法
阻塞状态（ BLOCKED ）	→	无法获得锁对象
等待状态（ WAITING ）	→	wait方法
计时等待（ TIMED_WAITING ）	→	sleep方法
结束状态（ TERMINATED ）	→	全部代码运行完毕

传智教育旗下高端IT教育品牌