重入锁

JUC 勾优C

java.util.concurrent

Java 并发编程工具包，Java 官方提供的一套专门用来处理并发编程的工具集合（接口+类）

并发：单核 CPU，多个线程“同时”运行，实际是交替执行，只不过速度太快，看起来是同时执行。

两个厨师一口锅

并行：多核 CPU，真正的多个线程同时运行。

两个厨师两口锅

重入锁是 JUC 使用频率非常高的一个类 ReentrantLock

ReentrantLock 就是对 synchronized 的升级，目的也是为了实现线程同步。

- ReentrantLock 是一个类，synchronized 是一个关键字。
- ReentrantLock 是 JDK 实现，synchronized 是 JVM 实现。
- synchronized 可以自动释放锁，ReentrantLock 需要手动释放。

ReentrantLock 是 Lock 接口的实现类。

公平锁和非公平锁的区别

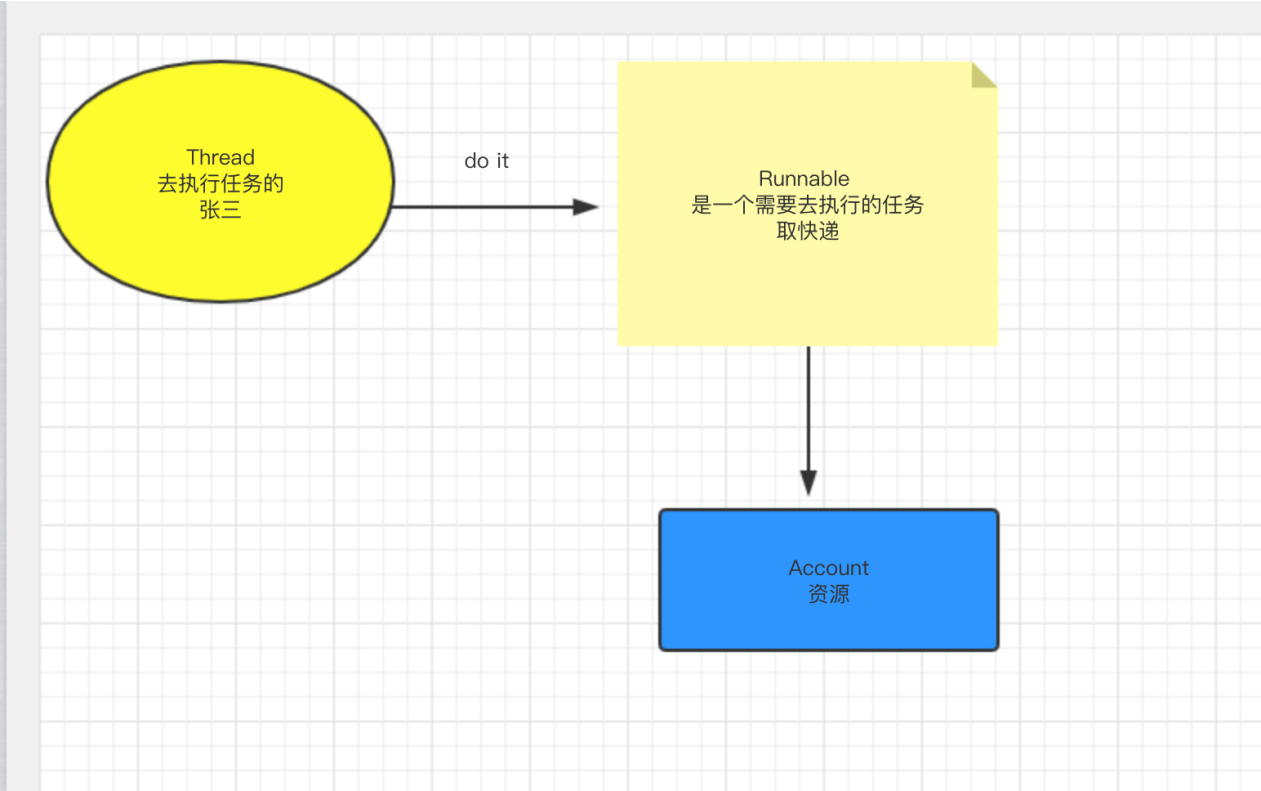
公平锁：线程同步时，多个线程排队，依次执行

非公平锁：线程同步时，可以插队

线程的实现有两种方式

- 继承 Thread
- 实现 Runnable

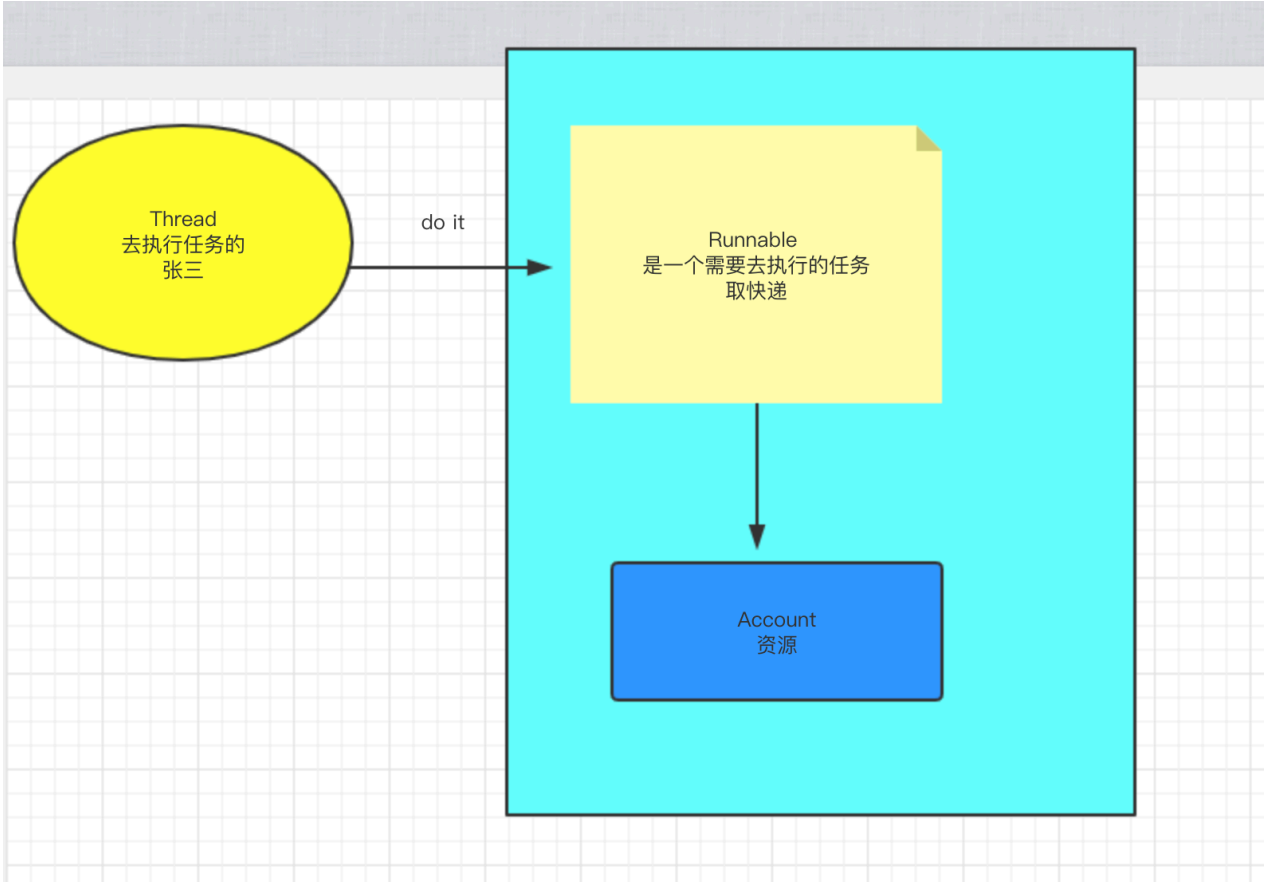
实现 Runnable 的耦合度更低




```
package com.southwind.demo;

import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.ReentrantLock;

public class Test {
  public static void main(String[] args) {
    Account account = new Account();
    new Thread(()->{
      account.count();
    },"A") .start();
    new Thread(()->{
      account.count();
    },"B") .start();
  }
}

/**
 * 将资源和 Runnable 进行解耦合
 * @author southwind
 *
 */
class Account{
  private static int num;
  public void count() {
    num++;
    try {
      TimeUnit.MILLISECONDS.sleep(1000);
    } catch (InterruptedException e) {
```

```
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    System.out.println(Thread.currentThread().getName()+"是当前的第"+num+"位访客");
  }
}
```

```
package com.southwind.demo;

import java.util.concurrent.TimeUnit;

public class Test2 {
  public static void main(String[] args) {
    Account2 account = new Account2();
    new Thread(account,"A").start();
    new Thread(account,"B").start();
  }
}

class Account2 implements Runnable{

  private static int num;

  @Override
  public void run() {
    // TODO Auto-generated method stub
    num++;
    try {
      TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    System.out.println(Thread.currentThread().getName()+"是当前的第"+num+"位访客");
  }

}
```



Tips

```
class Account{
  private static Integer num = 0;
  private static Integer id = 0;
  public void count() {
    synchronized (num) {
      num++;
      try {
        TimeUnit.MILLISECONDS.sleep(1000);
      } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
      System.out.println(Thread.currentThread().getName()+"是当前的第"+num+"位访客");
    }
  }
}
```

如果锁定 num 不能同步，锁定 id 可以同步，原因是什么?

synchronized 必须锁定唯一的元素才可以实现同步

num 的值每次都在变，所以 num 所指向的引用一直在变，所以不是唯一的元素，肯定无法实现同步。

id 的值永远不变，所以是唯一的元素，可以实现同步。

ReentrantLock

```
package com.southwind.demo;

import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.ReentrantLock;

public class Test3 {
  public static void main(String[] args) {
    Account3 account = new Account3();
    new Thread(()->{
      account.count();
    },"A").start();
    new Thread(()->{
      account.count();
    },"B").start();
  }
}

class Account3{
  private static int num;
  private ReentrantLock reentrantLock = new ReentrantLock();

  public void count() {
    //上锁
    reentrantLock.lock();
    reentrantLock.lock();
    num++;
    try {
      TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    System.out.println(Thread.currentThread().getName()+"是当前的第"+num+"位访
客");
    //解锁
    reentrantLock.unlock();
    reentrantLock.unlock();
  }
}
```

- Lock 上锁和解锁都需要开发者手动完成。
- 可以重复上锁，上几把锁就需要解几把锁。

ReentrantLock 除了可以重入之外，还有一个可以中断的特点，可中断是指某个线程在等待获取锁的过程中可以主动过终止线程。

```java
package com.southwind.demo;

import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.ReentrantLock;

public class Test5 {
  public static void main(String[] args) {
    StopLock stopLock = new StopLock();
    Thread t1 = new Thread(()->{
      stopLock.service();
    },"A");
    Thread t2 =new Thread(()->{
      stopLock.service();
    },"B");
    t1.start();
    t2.start();
    try {
      TimeUnit.SECONDS.sleep(1);
      t2.interrupt();
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}

class StopLock{

  private ReentrantLock reentrantLock = new ReentrantLock();

  public void service() {

    try {
      reentrantLock.lockInterruptibly();

      System.out.println(Thread.currentThread().getName()+"get lock");
      try {
        TimeUnit.SECONDS.sleep(5);
      } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }

    } catch (InterruptedException e1) {
      // TODO Auto-generated catch block
      e1.printStackTrace();
    } finally {
      reentrantLock.unlock();
    }
```

```
    }
}
```