线程同步

并发、并行

使用并发编程的目的？为了充分利用计算机的资源，提高性能，企业以盈利为目的。

并发：多个线程访问同一个共享资源，前提是计算机是单核 CPU，多个线程不是同时在访问，而是交替进行，只是因为 CPU 运行速度太快，看起来是同时在运行。

并行：多核 CPU，多个线程是真正的同时在运行，各自占用不同的 CPU，相互之间没有影响，也不会争夺资源。

Java 默认线程有两个，main（主线程），GC（垃圾回收机制）

synchronized 关键字实现线程同步，让在访问同一个资源的多个线程排队去完成业务，避免出现数据错乱的情况。

死锁 DeadLock

前提：一个线程完成业务需要同时访问两个资源。

死锁：多个线程同时在完成业务，出现争抢资源的情况。

资源类

```
package com.southwind.demo1;

public class DeadLockRunnable implements Runnable {
  //编号
  public int num;
  //资源
  private static Chopsticks chopsticks1 = new Chopsticks();
  private static Chopsticks chopsticks2 = new Chopsticks();

  /**
   * num = 1 拿到 chopsticks1，等待 chopsticks2
   * num = 2 拿到 chopsticks2，等待 chopsticks1
   */
  @Override
  public void run() {
    // TODO Auto-generated method stub
    if(num == 1) {
      System.out.println(Thread.currentThread().getName()+"拿到了chopsticks1，等待获取chopsticks2");
      synchronized (chopsticks1) {
        try {
          Thread.sleep(100);
        } catch (InterruptedException e) {
          // TODO Auto-generated catch block
```

```
            e.printStackTrace();
          }
          synchronized (chopsticks2) {
            System.out.println(Thread.currentThread().getName()+"用餐完毕！");
          }
        }
      }
      if(num == 2) {
        System.out.println(Thread.currentThread().getName()+"拿到了chopsticks2，等待获取chopsticks1");
        synchronized (chopsticks2) {
          try {
            Thread.sleep(100);
          } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
          }
          synchronized (chopsticks1) {
            System.out.println(Thread.currentThread().getName()+"用餐完毕！");
          }
        }
      }
    }

}
```

```
package com.southwind.demo1;

public class DeadLockTest {
  public static void main(String[] args) {
    DeadLockRunnable deadLockRunnable1 = new DeadLockRunnable();
    deadLockRunnable1.num = 1;
    DeadLockRunnable deadLockRunnable2 = new DeadLockRunnable();
    deadLockRunnable2.num = 2;
    new Thread(deadLockRunnable1,"张三").start();
    new Thread(deadLockRunnable2,"李四").start();
  }
}
```

如何破解死锁

不要让多线程并发访问

```
package com.southwind.demo1;

public class DeadLockTest {
  public static void main(String[] args) {
    DeadLockRunnable deadLockRunnable1 = new DeadLockRunnable();
```

```
    deadLockRunnable1.num = 1;
    DeadLockRunnable deadLockRunnable2 = new DeadLockRunnable();
    deadLockRunnable2.num = 2;
    new Thread(deadLockRunnable1,"张三").start();
    try {
      Thread.sleep(2000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    new Thread(deadLockRunnable2,"李四").start();
  }
}
```

使用 lambda 表达式简化代码开发

```
package com.southwind.demo2;

public class Test3 {
  public static void main(String[] args) {
    //lambda表达式
    new Thread(()->{
      for(int i=0;i<100;i++) {
        System.out.println("++++++++++Runnable");
      }
    }) .start();
  }
}
```

```
package com.southwind.demo2;

public class Test3 {
  public static void main(String[] args) {
    new Thread(()->{for(int i=0;i<100;i++)
System.out.println("+++++++Runnable");}).start();
    new Thread(()->{for(int i=0;i<100;i++) System.out.println("----
Runnable");}).start();
    new Thread(()->{for(int i=0;i<100;i++)
System.out.println("++++======++Runnable");}).start();
  }
}
```

Lock

JUC

java.util.concurrent

Lock 是一个接口，用来实现线程同步的，功能与 synchronized 一样。

Lock 使用频率最高的实现类是 ReentrantLock（重入锁），可以重复上锁。

```
package com.southwind.demo3;

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

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

class Account implements Runnable{

  private static int num;
  private Lock lock = new ReentrantLock();

  @Override
  public void run() {
    // TODO Auto-generated method stub
    lock.lock();
    num++;
    System.out.println(Thread.currentThread().getName()+"是当前的第"+num+"位访
客");
    lock.unlock();
  }

}
```

实现资源和 Runnable 接口的解耦合。

```
package com.southwind.demo3;

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

public class Test2 {
  public static void main(String[] args) {
    Account account = new Account();
    new Thread(()->{
      account.count();
    },"A").start();
    new Thread(()->{
      account.count();
```

```
    },"B").start();
  }
}

class Account {
  private int num;
  private Lock lock = new ReentrantLock();

  public void count() {
    lock.lock();
    num++;
    System.out.println(Thread.currentThread().getName()+"是第"+num+"位访客");
    lock.unlock();
  }

}
```