目录

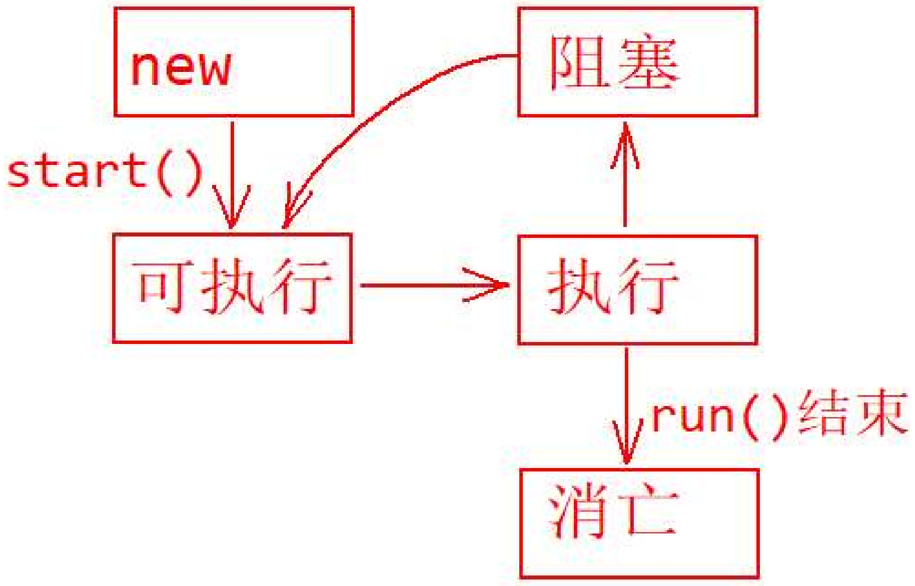
Day16. Java
1 线程
1.1 线程状态
1.2 方法
1.3 多线程共享数据冲突
1.4 线程同步 synchronized
1.5 生产者、消费者模型
1.6 等待和通知
1.7 同步监视器模型（了解）
2 作业

Day16. Java

1 线程

- 线程创建
  - 继承Thread
  - 实现Runnable

1.1 线程状态



1.2 方法

- Thread.currentThread()
  获得正在执行的线程对象

- Thread.sleep(毫秒值)
  让线程暂停指定的毫秒时长

- Thread.yield()
  让步，放弃时间片

- getName()
  setName(name)

- start()
  启动线程，线程启动后，执行run()方法

- interrupt()
  打断一个线程的暂停状态
  被打断的线程，会出现打断异常 InterruptedException

- join()
  当前线程暂停，等待被调用的线程结束

- setDaemon(true)
  把线程设置成后台线程、守护线程

  - 虚拟机回到等待所有前台线程都结束后，自动关闭
  - 而不会等待后台线程结束

- getPriority()
- setPriority(优先级)
  优先级 1 到 10 默认 5

sleep

项目：day1602_线程
类：day1602.Test1

```
package day1602;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;

public class Test1 {
    public static void main(String[] args) {
        T1 t1 = new T1();
        t1.start();

        //main线程，打断t1的暂停状态
        System.out.println("按回车");
        while(true) {
            new Scanner(System.in).nextLine();
            t1.interrupt();
        }
    }

    static class T1 extends Thread {
        @Override
        public void run() {
            SimpleDateFormat sdf =
             new SimpleDateFormat(
             "HH:mm:ss.SSS");
            long t = 0;//记录第一次的时间点

            while(true) {
                Date d = new Date();
                if(t == 0) {
                    t = d.getTime();
                }
                String s = sdf.format(d);
                System.out.println(s);
                long y = (d.getTime() - t) % 1000;
                try {
                    Thread.sleep(1000-y);
                } catch (InterruptedException e) {
                }
            }


        }
    }
}
```

Test2

```
package day1602;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;

public class Test2 {
    public static void main(String[] args) throws Exception {
        T1 t1 = new T1();
        System.out.println(
                "10秒挑战，准备好，按回车开始");
        new Scanner(System.in).nextLine();
        t1.start();

        //main线程，打断t1的暂停状态
        System.out.println("数10秒按回车");
        //while(true) {
            new Scanner(System.in).nextLine();
            t1.interrupt();
        //}
    }

    static class T1 extends Thread {
        @Override
        public void run() {
```

```
        SimpleDateFormat sdf =
         new SimpleDateFormat(
         "HH:mm:ss.SSS");
        long t = 0;//记录第一次的时间点


        while(true) {
           Date d = new Date();
           if(t == 0) {
               t = d.getTime();
           }
           //String s = sdf.format(d);
           //System.out.println(s);
           long y = (d.getTime() - t) % 1000;
           try {
               Thread.sleep(1000-y);
           } catch (InterruptedException e) {
               d = new Date();
               t = d.getTime()-t;
               // 10000, 10012, 9982
               y = (t%1000) / 10;
               t = t/1000;
               String s = y<10 ? "0"+y : ""+y;
               System.out.println(t+"."+s+" 秒");
               break;
           }
        }


     }
  }
}
```

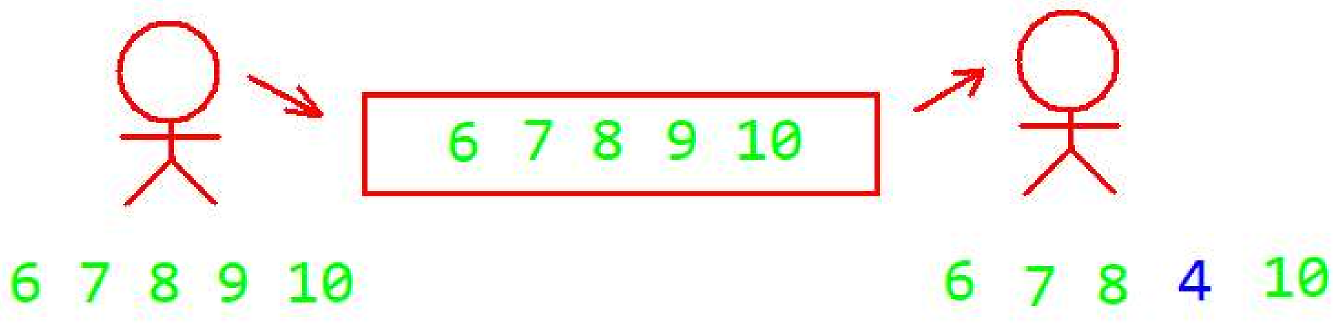
join

```
Test3

package day1602;

public class Test3 {
   public static void main(String[] args) throws Exception {
      /*
       * 1000万内有多少个质数
       */
      System.out.println("\n\n--单线程---------");
      f1();

      System.out.println("\n\n--5个线程---------");
      f2();

   }

   private static void f2() throws Exception {
      long t = System.currentTimeMillis();

      T1 t1 = new T1(0, 2000000);
      T1 t2 = new T1(2000000,4000000);
      T1 t3 = new T1(4000000,6000000);
      T1 t4 = new T1(6000000,8000000);
      T1 t5 = new T1(8000000,10000000);
      t1.start();
      t2.start();
      t3.start();
      t4.start();
      t5.start();
      t1.join();
      t2.join();
      t3.join();
      t4.join();
      t5.join();
      int n = t1.count+
            t2.count+
            t3.count+
            t4.count+
            t5.count;

      t = System.currentTimeMillis()-t;
      System.out.println(t);

      System.out.println(n);
   }

   private static void f1() throws Exception {
      long t = System.currentTimeMillis();

      T1 t1 = new T1(0, 10000000);
      t1.start();
      t1.join();//main暂停，等待t1结束

      t = System.currentTimeMillis()-t;
      System.out.println(t);

      System.out.println(t1.count);

   }

   static class T1 extends Thread {
      int start;
      int end;
      int count;
      public T1(int start, int end) {
         if(start<=2) {
            start = 3;
            count = 1;
         }
         this.start = start;
         this.end = end;
      }

      @Override
      public void run() {
         for(int i=start;i<end;i++) {
            if(isPrime(i)) {
               count++;
            }
         }
```

```
        }

        private boolean isPrime(int i) {
            double max = 1+Math.sqrt(i);
            for(int j=2;j<max;j++) {
                if(i%j == 0) return false;
            }

            return true;
        }
    }
}
```

1.3 多线程共享数据冲突

- 一个线程修改数据
- 另一个线程访问数据，可能访问到修改了一半的数据，称为“脏数据”



数据访问冲突

Test4

```
package day1602;

import java.util.Arrays;

public class Test4 {
    static char[] a = {
            '-','-','-','-','-'
    };

    public static void main(String[] args) {
        Thread t1 = new Thread() {
            @Override
            public void run() {
                char c = '*';
                while(true) {
                    for (int i = 0; i < a.length; i++) {
                        a[i] = c;
                    }
                    c = (c=='*'?'-':'*');
                }
            }
        };

        Thread t2 = new Thread() {
           @Override
           public void run() {
              while(true) {
                  System.out.println(
                   Arrays.toString(a));
              }
           }
        };
        t1.start();
        t2.start();
     }
  }
```

1.4 线程同步 synchronized

- java任何对象，都有一个同步锁
- synchronized 关键字，要求执行的线程，必须获得锁，才能执行
- synchronized(对象) {
      ...
  }

  争夺指定对象的锁

- synchronized void f() {
      ...
  }

  争夺当前对象 this 的锁

- static synchronized void f() {
      ...
  }

  争夺“类对象”的锁

- 为了保证线程安全，必须降低效率，牺牲性能

同步

```
package day1602;

import java.util.Arrays;

public class Test4 {
   static char[] a = {
         '-','-','-','-','-'
   };

   public static void main(String[] args) {
      Thread t1 = new Thread() {
         @Override
         public void run() {
            char c = '*';
            while(true) {
                 synchronized (a) {
                   for (int i = 0; i < a.length; i++) {
                      a[i] = c;
                   }
                 }
                c = (c=='*'?'-':'*');
            }
         }
      };

      Thread t2 = new Thread() {
         @Override
         public void run() {
            while(true) {
                 synchronized (a) {
                   System.out.println(
                         Arrays.toString(a));
                 }
            }
         }
      };
      t1.start();
      t2.start();
   }
}
```

Test5

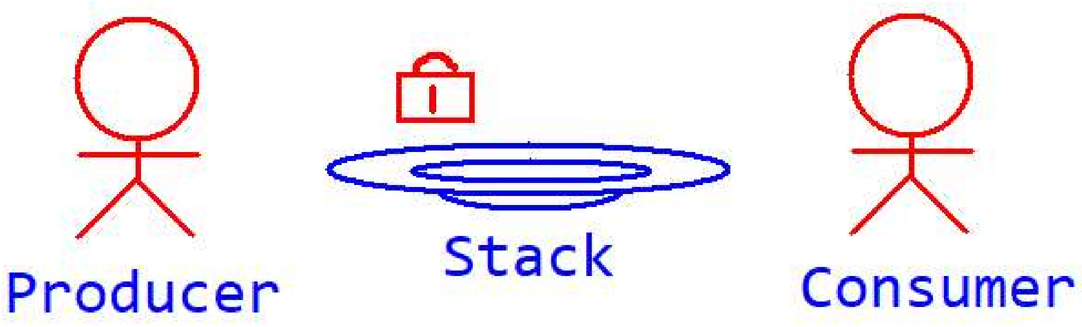
```
package day1602;

public class Test5 {
   public static void main(String[] args) {
      R1 r1 = new R1();
      Thread t1 = new Thread(r1);
      t1.start();

      //main线程
      while(true) {
         int i = r1.get();
         if(i%2 == 1) {
            System.out.println(i);
            System.exit(0);//退出虚拟机
         }
      }
   }

   static class R1 implements Runnable {
      static int i;
      //争夺当前对象，this，的锁
      public synchronized void add() {
         i++;
         i++;
      }
      public synchronized int get() {
         return i;
      }
      @Override
      public void run() {
         while(true) {
            add();
         }
      }
   }
}
```

Test6

```
package day1602;

public class Test6 {
   public static void main(String[] args) {
      R1 r1 = new R1();
      Thread t1 = new Thread(r1);
      t1.start();

      //main线程
      R1 r2 = new R1();
      while(true) {
         int i = r2.get();
         if(i%2 == 1) {
            System.out.println(i);
            System.exit(0);//退出虚拟机
         }
      }
   }

   static class R1 implements Runnable {
      static int i;
      //争夺当前对象，this，的锁
      public static synchronized void add() {
         i++;
         i++;
      }
      public static synchronized int get() {
         return i;
      }
      @Override
      public void run() {
         while(true) {
            add();
         }
      }
   }
}
```

1.5 生产者、消费者模型

- 线程之间传递数据的一种方式



生产者消费者

Stack

```
package day1602;

public class Stack {
   private char[] a = new char[5];
   private int index;

   public void push(char c) {
      if(isFull()) {
         return;
      }
      a[index] = c;
      index++;
   }
   public char pop() {
      if(isEmpty()) {
         return ' ';//用空格表示没有数据
      }
      index--;
      char c = a[index];
      return c;
   }
   public boolean isEmpty() {
      return index==0;
   }
   public boolean isFull() {
      return index==5;
   }
}
```

Producer

```
package day1602;

import java.util.Random;
```

```
public class Producer extends Thread {
   private Stack stack;
   public Producer(Stack stack) {
      this.stack = stack;
   }
   @Override
   public void run() {
      while(true) {
         // 'a'+[0, 26)
         char c =
          (char) ('a'+new Random().nextInt(26));
         synchronized (stack) {
            stack.push(c);
            System.out.println("压入 <-- "+c);
         }
      }
   }
}
```

Consumer

```
package day1602;

public class Consumer extends Thread {
   private Stack stack;
   public Consumer(Stack stack) {
      this.stack = stack;
   }
   @Override
   public void run() {
      while(true) {
         synchronized (stack) {
            char c = stack.pop();
            System.out.println("弹出 --> "+ c );
         }
      }
   }

}
```

Test7

```
package day1602;

public class Test7 {
   public static void main(String[] args) {
      Stack stack = new Stack();
      Producer p = new Producer(stack);
      Consumer c = new Consumer(stack);

      p.start();
      c.start();
```

```
    }
}
```

1.6 等待和通知

- 当前线程，在指定的对象上等待

```
stack.wait()
```

- 在指定的对象上发送通知

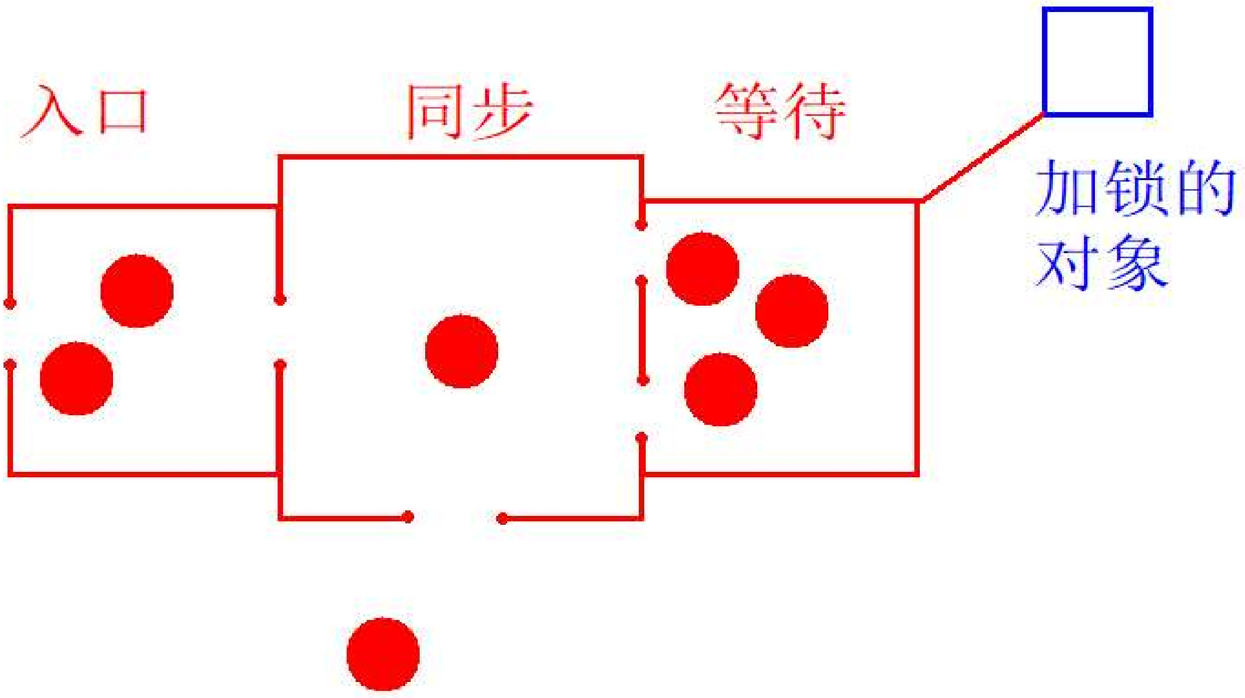
```
stack.notify()      //通知所有线程中的一个
stack.notifyAll()   //通知全部线程
```

- Object 中定义的方法
  - wait()
  - notify()
  - notifyAll()

- 必须在同步代码块内才能调用
- 必须在加锁的对象上等待
- wait()外面，通常是一个循环条件检查

1.7 同步监视器模型（了解）

- 执行到synchronized，会在加锁的对象上，关联一个同步监视器对象



2 作业

- 重写生产者消费者
  - `Stack`
  - `Producer`
  - `Consumer`
  - `Test7`