史上最全的 Java 新手问题汇总

Java 是目前最流行的编程语言之一——它可以用来编写 Windows 程序或者是 Web 应用，移动应用，网络程序，消费电子产品，机顶盒设备，它无处不在。

有超过 30 亿的设备是运行在 Java 之上的。根据 Oracle 的统计数据，光是使用中的 Java Card 就有有 50 亿。

超过 900 万程序员选择使用 Java 进行开发，它是最受开发人员欢迎的语言，同时也是最流行的开发平台。

本文为那些准 Java 程序员们准备了一系列广为流传的 Java 最佳编程实践

优先返回空集合而非 null

如果程序要返回一个不包含任何值的集合，确保返回的是空集合而不是 null。这能节省大量的”if else”检查。

```
public class getLocationName {

    return (null==cityName ? "": cityName);

}
```

谨慎操作字符串

如果两个字符串在 for 循环中使用+操作符进行拼接，那么每次循环都会产生一个新的字符串对象。这不仅浪费内存空间同时还会影响性能。类似的，如果初始化字符串对象，尽量不要使用构造方法，而应该直接初始化。比方说：

```
//Slower Instantiation

String bad = new String("Yet another string object");



//Faster Instantiation

String good = "Yet another string object"
```

避免无用对象

创建对象是 Java 中最昂贵的操作之一。因此最好在有需要的时候再进行对象的创建/初始化。如下：

```
import java.util.ArrayList;

import java.util.List;



public class Employees {



    private List Employees;
```

```
    public List getEmployees() {


        //initialize only when required

        if(null == Employees) {

            Employees = new ArrayList();

        }

        return Employees;

    }

}
```

数组与 ArrayList 之争

开发人员经常会发现很难在数组和 ArrayList 间做选择。它们二者互有优劣。如何选择应该视情况而定。

```
import java.util.ArrayList;


public class arrayVsArrayList {
```

加楠楠 QQ1692250804 获取跟多学习资料

```
    public static void main(String[] args) {

        int[] myArray = new int[6];

        myArray[7]= 10; // ArraysOutOfBoundException



        //Declaration of ArrayList. Add and Remove of elements is easy.

        ArrayList<Integer> myArrayList = new ArrayList<>();

        myArrayList.add(1);

        myArrayList.add(2);

        myArrayList.add(3);

        myArrayList.add(4);

        myArrayList.add(5);

        myArrayList.remove(0);



        for(int i = 0; i < myArrayList.size(); i++) {

        System.out.println("Element: " + myArrayList.get(i));

        }
```

加楠楠 QQ1692250804 获取跟多学习资料

```
        //Multi-dimensional Array

        int[][][] multiArray = new int [3][3][3];

    }

}
```

- 数组是定长的，而 ArrayList 是变长的。由于数组长度是固定的，因此在声明数组时就已经分配好内存了。而数组的操作则会更快一些。另一方面，如果我们不知道数据的大小，那么过多的数据便会导致 ArrayOutOfBoundException，而少了又会浪费存储空间。
- ArrayList 在增删元素方面要比数组简单。
- 数组可以是多维的，但 ArrayList 只能是一维的。
- try 块的 finally 块没有被执行

看下下面这段代码：

```
public class shutDownHooksDemo {

    public static void main(String[] args) {

        for(int i=0;i<5;i++)

        {

            try {
```

```
                if(i==4) {

                    System.out.println("Inside Try Block.Exiting without execut
ing Finally block.");

                    System.exit(0);

                }

            }

            finally {

                System.out.println("Inside Finally Block.");

            }

        }

    }

}
```

从代码来看，貌似 finally 块中的 println 语句应该会被执行 5 次。但当程序运行后，你会发现 finally 块只执行了 4 次。第 5 次迭代的时候会触发 exit 函数的调用，于是这第 5 次的 finally 便永远也触发不到了。原因便是——System.exit 会挂起所有线程的执行，包括当前线程。即便是 try 语句后的 finally 块，只要是执行了 exit，便也无力回天了。

在调用 System.exit 时，JVM 会在关闭前执行两个结束任务：

加楠楠 QQ1692250804 获取跟多学习资料

首先，它会执行完所有通过 Runtime.addShutdownHook 注册进来的终止的钩子程序。这一点很关键，因为它会释放 JVM 外部的资源。

接下来的便是 Finalizer 了。可能是 System.runFinalizersOnExit 也可能是 Runtime.runFinalizersOnExit。finalizer 的使用已经被废弃有很长一段时间了。finalizer 可以在存活对象上进行调用，即便是这些对象仍在被其它线程所使用。而这会导致不可预期的结果甚至是死锁。

```
public class shutDownHooksDemo {



    public static void main(String[] args) {

            for(int i=0;i<5;i++)

            {

                    final int final_i = i;

                    try {

                            Runtime.getRuntime().addShutdownHook(

                                    new Thread() {

                                    public void run() {

                                    if(final_i==4) {
```

```
                                        System.out.println("Inside Try Block.Ex
iting without executing Finally block.");
                                        System.exit(0);
                                    }
                                }
                                });
                    }
                    finally {
                        System.out.println("Inside Finally Block.");
                    }

            }
    }
}
```

判断奇数

看下这几行代码，看看它们是否能用来准确地判断一个数是奇数？

```
public boolean oddOrNot(int num) {

    return num % 2 == 1;

}
```

看似是对的，但是每执行四便会有一个错误的结果（用数据说话）。考虑到负奇数的情况，它除以 2 的结果就不会是 1。因此，返回值是 false，而这样是不对的。

代码可以修改成这样：

```
public boolean oddOrNot(int num) {

    return (num & 1) != 0;

}
```

这么写不光是负奇数的问题解决了，并且还是经过充分优化过的。因为算术运算和逻辑运行要比乘除运算更高效，计算的结果也会更快。

单引号与双引号的区别

```
public class Haha {

    public static void main(String args[]) {

    System.out.print("H" + "a");
```

```
    System.out.print('H' + 'a');

    }

}
```

看起来这段代码会返回”Haha”,但实际返回的是 Ha169。原因就是用了双引号的时候，字符会被当作字符串处理，而如果是单引号的话，字符值会通过一个叫做基础类型拓宽的操作来转换成整型值。然后再将值相加得到 169。

一些防止内存泄露的小技巧

内存泄露会导致软件的性能降级。由于 Java 是自动管理内存的，因此开发人员并没有太多办法介入。不过还是有一些方法能够用来防止内存泄露的。

- 查询完数据后立即释放数据库连接
- 尽可能使用 finally 块
- 释放静态变量中的实例
- 避免死锁

死锁出现的原因有很多。避免死锁不是一句话就能解决的。通常来说，当某个同步对象在等待另一个同步对象所拥有的资源上的锁时，便会产生死锁。

试着运行下下面的程序。它会告诉你什么是死锁。这个死锁是由于两个线程都在等待对方所拥有的资源，因此会产生死锁。它们会一直等待，没有谁会先放手。

加楠楠 QQ1692250804 获取跟多学习资料

```
public class DeadlockDemo {

   public static Object addLock = new Object();

   public static Object subLock = new Object();



   public static void main(String args[]) {



      MyAdditionThread add = new MyAdditionThread();

      MySubtractionThread sub = new MySubtractionThread();

      add.start();

      sub.start();

   }

private static class MyAdditionThread extends Thread {

      public void run() {

         synchronized (addLock) {

        int a = 10, b = 3;

        int c = a + b;

            System.out.println("Addition Thread: " + c);
```

```
            System.out.println("Holding First Lock...");

            try { Thread.sleep(10); }

            catch (InterruptedException e) {}

            System.out.println("Addition Thread: Waiting for AddLock...");

            synchronized (subLock) {

              System.out.println("Threads: Holding Add and Sub Locks...");

            }

        }

    }

  }

  private static class MySubtractionThread extends Thread {

    public void run() {

       synchronized (subLock) {

      int a = 10, b = 3;

      int c = a - b;

            System.out.println("Subtraction Thread: " + c);
```

```
            System.out.println("Holding Second Lock...");

            try { Thread.sleep(10); }

            catch (InterruptedException e) {}

            System.out.println("Subtraction  Thread: Waiting for SubLock...");

            synchronized (addLock) {

              System.out.println("Threads: Holding Add and Sub Locks...");

            }

        }

    }

}
```

输出：

```
Addition Thread: 13

Subtraction Thread: 7

Holding First Lock...

Holding Second Lock...
```

加楠楠 QQ1692250804 获取跟多学习资料

```
Addition Thread: Waiting for AddLock...

Subtraction  Thread: Waiting for SubLock...
```

但如果调用的顺序变一下的话，死锁的问题就解决了。

```
public class DeadlockSolutionDemo {

   public static Object addLock = new Object();

   public static Object subLock = new Object();



   public static void main(String args[]) {



      MyAdditionThread add = new MyAdditionThread();

      MySubtractionThread sub = new MySubtractionThread();

      add.start();

      sub.start();

   }



private static class MyAdditionThread extends Thread {

      public void run() {
```

```
        synchronized (addLock) {

       int a = 10, b = 3;

       int c = a + b;

           System.out.println("Addition Thread: " + c);

           System.out.println("Holding First Lock...");

           try { Thread.sleep(10); }

           catch (InterruptedException e) {}

           System.out.println("Addition Thread: Waiting for AddLock...");

           synchronized (subLock) {

              System.out.println("Threads: Holding Add and Sub Locks...");

           }

        }

     }

   }


  private static class MySubtractionThread extends Thread {
```

加楠楠 QQ1692250804 获取跟多学习资料

```
    public void run() {

       synchronized (addLock) {

      int a = 10, b = 3;

      int c = a - b;

          System.out.println("Subtraction Thread: " + c);

          System.out.println("Holding Second Lock...");

          try { Thread.sleep(10); }

          catch (InterruptedException e) {}

          System.out.println("Subtraction  Thread: Waiting for SubLock...");

          synchronized (subLock) {

             System.out.println("Threads: Holding Add and Sub Locks...");

          }

       }

    }

  }

}
```

加楠楠 QQ1692250804 获取跟多学习资料

输出：

```
Addition Thread: 13

Holding First Lock...

Addition Thread: Waiting for AddLock...

Threads: Holding Add and Sub Locks...

Subtraction Thread: 7

Holding Second Lock...

Subtraction  Thread: Waiting for SubLock...

Threads: Holding Add and Sub Locks...
```

替 Java 省点内存

某些 Java 程序是 CPU 密集型的，但它们会需要大量的内存。这类程序通常运行得很缓慢，因为它们对内存的需求很大。为了能提升这类应用的性能，可得给它们多留点内存。因此，假设我们有一台拥有 10G 内存的 Tomcat 服务器。在这台机器上，我们可以用如下的这条命令来分配内存:

```
export JAVA OPTS="$JAVA OPTS -Xms5000m -Xmx6000m -XX:PermSize=1024m -XX:MaxP

ermSize=2048m"
```

加楠楠 QQ1692250804 获取跟多学习资料

- Xms = 最小内存分配
- Xmx = 最大内存分配
- XX:PermSize = JVM 启动时的初始大小
- XX:MaxPermSize = JVM 启动后可分配的最大空间
- 如何计算 Java 中操作的耗时

在 Java 中进行操作计时有两个标准的方法：System.currentTimeMillis()和 System.nanoTime()。问题就在于，什么情况下该用哪个。从本质上来讲，他们的作用都是一样的，但有以下几点不同：

1. System.currentTimeMillis()的精度在千分之一秒到千分之 15 秒之间(取决于系统) 而 System.nanoTime()则能到纳秒级。
2. System.currentTimeMillis 读操作耗时在数个 CPU 时钟左右。而 System.nanoTime()则需要上百个。
3. System.currentTimeMillis 对应的是绝对时间(1970 年 1 月 1 日所经历的毫秒数), 而 System.nanoTime()则不与任何时间点相关。
4. Float 还是 double

数据类型	所用字节	有效位数
float	4	7
double	8	15

在对精度要求高的场景下，double 类型相对 float 要更流行一些，理由如下：

加楠楠 QQ1692250804 获取跟多学习资料

大多数处理器在处理 float 和 double 上所需的时间都是差不多的。而计算时间一样的前提下，double 类型却能提供更高的精度。

幂运算

Java 是通过异或操作来进行幂运算的。Java 对于幂运算有两种处理方式：

乘积：

```
double square = double a * double a;                          // Optimized

double cube = double a * double a * double a;                  // Non-optimized

double cube = double a * double square;                       // Optimized

double quad = double a * double a * double a * double a;          // Non-optimi
zed

double quad = double square * double square;                  // Optimized
```

pow 方法：在无法使用乘积的情况下可以使用 pow 方法。

```
double cube = Math.pow(base, exponent);
```

不到万不得已不要使用 Math.pow。比方说，当指数是小数的时候。因为 Math.pow 要比乘积慢 300-600 倍左右。

加楠楠 QQ1692250804 获取跟多学习资料

如何处理空指针异常

空指针异常是 Java 中很常见的异常。当你尝试调用一个 null 对象上的方法时便会抛出这个异常。比如：

```
int noOfStudents = school.listStudents().count;
```

在上述例子中，school 为空或者 listStudents()为空都可能会抛出了 NullPointerException。因此最好检查下对象是否为空以避免类似情况。

```
private int getListOfStudents(File[] files) {

    if (files == null)

      throw new NullPointerException("File list cannot be null");

   }
```

JSON 编码

JSON 是数据存储及传输的一种协议。与 XML 相比，它更易于使用。由于它非常轻量级以及自身的一些特性，现在 JSON 在网络上已经是越来越流行了。常见的数据结构都可以编码成 JSON 然后在各个网页间自由地传输。不过在开始编码前，你得先安装一个 JSON 解析器。在下面的例子中，我们将使用 json.simple 库来完成这项工作 (https://code.google.com/p/json-simple/)。

下面是编码成 JSON 串的一个简单的例子。

加楠楠 QQ1692250804 获取跟多学习资料

```
import org.json.simple.JSONObject;

import org.json.simple.JSONArray;



public class JsonEncodeDemo {



   public static void main(String[] args) {



      JSONObject obj = new JSONObject();

      obj.put("Novel Name", "Godaan");

      obj.put("Author", "Munshi Premchand");



      JSONArray novelDetails = new JSONArray();

      novelDetails.add("Language: Hindi");

      novelDetails.add("Year of Publication: 1936");

      novelDetails.add("Publisher: Lokmanya Press");



      obj.put("Novel Details", novelDetails);
```

加楠楠 QQ1692250804 获取跟多学习资料

```
        System.out.print(obj);

    }

}
```

输出：

```
{"Novel Name":"Godaan","Novel Details":["Language: Hindi","Year of Publicati

on: 1936","Publisher: Lokmanya Press"],"Author":"Munshi Premchand"}
```

JSON 解析

开发人员要想解析 JSON 串，首先你得知道它的格式。下面例子有助于你来理解这一点：

```
import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.Iterator;



import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;
```

```
import org.json.simple.parser.ParseException;



public class JsonParseTest {



    private static final String filePath = "//home//user//Documents//jsonDemo
File.json";



    public static void main(String[] args) {



        try {

            // read the json file

            FileReader reader = new FileReader(filePath);

            JSONParser jsonParser = new JSONParser();

            JSONObject jsonObject = (JSONObject)jsonParser.parse(reader);



            // get a number from the JSON object

            Long id =  (Long) jsonObject.get("id");

            System.out.println("The id is: " + id);
```

```
        // get a String from the JSON object

        String   type = (String) jsonObject.get("type");

        System.out.println("The type is: " + type);


        // get a String from the JSON object

        String   name = (String) jsonObject.get("name");

        System.out.println("The name is: " + name);


        // get a number from the JSON object

        Double ppu =  (Double) jsonObject.get("ppu");

        System.out.println("The PPU is: " + ppu);


        // get an array from the JSON object

        System.out.println("Batters:");

        JSONArray batterArray= (JSONArray) jsonObject.get("batters");

        Iterator i = batterArray.iterator();
```

加楠楠 QQ1692250804 获取跟多学习资料

```
            // take each value from the json array separately

            while (i.hasNext()) {

                JSONObject innerObj = (JSONObject) i.next();

                System.out.println("ID "+ innerObj.get("id") +

                        " type " + innerObj.get("type"));

            }



            // get an array from the JSON object

            System.out.println("Topping:");

            JSONArray toppingArray= (JSONArray) jsonObject.get("topping");

            Iterator j = toppingArray.iterator();

            // take each value from the json array separately

            while (j.hasNext()) {

                JSONObject innerObj = (JSONObject) j.next();

                System.out.println("ID "+ innerObj.get("id") +

                        " type " + innerObj.get("type"));
```

加楠楠 QQ1692250804 获取跟多学习资料

```
            }



        } catch (FileNotFoundException ex) {

            ex.printStackTrace();

        } catch (IOException ex) {

            ex.printStackTrace();

        } catch (ParseException ex) {

            ex.printStackTrace();

        } catch (NullPointerException ex) {

            ex.printStackTrace();

        }



    }



}
```

jsonDemoFile.json

```
{
```

```
    "id": 0001,

    "type": "donut",

    "name": "Cake",

    "ppu": 0.55,

    "batters":

        [

            { "id": 1001, "type": "Regular" },

            { "id": 1002, "type": "Chocolate" },

            { "id": 1003, "type": "Blueberry" },

            { "id": 1004, "type": "Devil's Food" }

        ],

    "topping":

        [

            { "id": 5001, "type": "None" },

            { "id": 5002, "type": "Glazed" },

            { "id": 5005, "type": "Sugar" },
```

加楠楠 QQ1692250804 获取跟多学习资料

```
          { "id": 5007, "type": "Powdered Sugar" },
          { "id": 5006, "type": "Chocolate with Sprinkles" },
          { "id": 5003, "type": "Chocolate" },
          { "id": 5004, "type": "Maple" }
       ]
}
The id is: 1
The type is: donut
The name is: Cake
The PPU is: 0.55
Batters:
ID 1001 type Regular
ID 1002 type Chocolate
ID 1003 type Blueberry
ID 1004 type Devil's Food
Topping:
```

加楠楠 QQ1692250804 获取跟多学习资料

```
ID 5001 type None

ID 5002 type Glazed

ID 5005 type Sugar

ID 5007 type Powdered Sugar

ID 5006 type Chocolate with Sprinkles

ID 5003 type Chocolate

ID 5004 type Maple
```

简单字符串查找

Java 提供了一个库函数叫做 indexOf()。这个方法可以用在 String 对象上，它返回的是要查找的字符串所在的位置序号。如果查找不到则会返回-1。

列出目录下的文件

你可以用下面的代码来列出目录下的文件。这个程序会遍历某个目录下的所有子目录及文件，并存储到一个数组里，然后通过遍历数组来列出所有文件。

```
import java.io.*;



public class ListContents {
```

```
    public static void main(String[] args) {

        File file = new File("//home//user//Documents/");

        String[] files = file.list();



        System.out.println("Listing contents of " + file.getPath());

        for(int i=0 ; i < files.length ; i++)

        {

            System.out.println(files[i]);

        }

    }

}
```

一个简单的 IO 程序

Java 提供了 FileInputStream 以及 FileOutputStream 类来进行文件的读写操作。

FileInputStream 的构造方法会接收输入文件的路径作为入参然后创建出一个文件的输入流。同样的，FileOutputStream 的构造方法也会接收一个文件路径作为入参然后创建出文件的输出流。在处理完文件之后，一个很重要的操作就是要记得”close”掉这些流。

加楠楠 QQ1692250804 获取跟多学习资料

```
import java.io.*;



public class myIODemo {

   public static void main(String args[]) throws IOException {

      FileInputStream in = null;

      FileOutputStream out = null;



      try {

         in = new FileInputStream("//home//user//Documents//InputFile.txt
");

         out = new FileOutputStream("//home//user//Documents//OutputFile.tx
t");



         int c;

         while((c = in.read()) != -1) {

            out.write(c);

         }

      } finally {
```

```
            if(in != null) {
                in.close();
            }
            if(out != null) {
                out.close();
            }
        }
    }
}
```

在 Java 中执行某个 shell 命令

Java 提供了 Runtime 类来执行 shell 命令。由于这些是外部的命令，因此异常处理就显得异常重要。在下面的例子中，我们将通过一个简单的例子来演示一下。我们会在 shell 命令行中打开一个 pdf 文件。

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
```

```
public class ShellCommandExec {


   public static void main(String[] args) {

      String gnomeOpenCommand = "gnome-open //home//user//Documents//MyDoc.
pdf";


      try {

         Runtime rt = Runtime.getRuntime();

         Process processObj = rt.exec(gnomeOpenCommand);


         InputStream stdin = processObj.getErrorStream();

         InputStreamReader isr = new InputStreamReader(stdin);

         BufferedReader br = new BufferedReader(isr);


         String myoutput = "";


         while ((myoutput=br.readLine()) != null) {
```

加楠楠 QQ1692250804 获取跟多学习资料

```
                myoutput = myoutput+"/n";

            }

            System.out.println(myoutput);

        }

        catch (Exception e) {

            e.printStackTrace();

        }

    }

}
```

使用正则

正则表达式的结构摘录如下（来源: Oracle 官网）

字符

x	字符 x
/	反斜杠
/0n	8 进制值为 0n 的字符(0<=n<=7)

加楠楠 QQ1692250804 获取跟多学习资料

/0nn	
/0mnn	8 进制值为 0mnn 的字符(0 <= m <= 3, 0<=n<=7)
/xhh	16 进制值为 0xhh 的字符
/uhhhh	16 进制值为 0xhhhh 的字符
/x{h…h}	16 进制值为 0xh…h 的字符(Character.MINCODEPOINT <= 0xh…h <= Character.MAXCODEPOINT)
/t	制表符(‘/u0009’)
/n	换行符(‘/u000A’)
/r	回车(‘/u000D’)
/f	分页符(‘/u000C’)
/a	警告符(‘/u0007’)
/e	ESC(‘/u001B’)
/cx	ctrl+x

字符分类

[abc]	a, b 或 c
[^abc]	abc 以外的任意字符
[a-zA-Z]	a 到 z 以及 A 到 Z

加楠楠 QQ1692250804 获取跟多学习资料

[a-d[m-p]]	a 到 d 或者 m 到 p[a-dm-p]则是取并集
[a-z&&[def]]	d,e 或 f(交集)
[ad-z]	
[a-z&&[^bc]]	a 到 z 但不包括 b 和 c
[a-z&&[^m-p]]	a 到 z 但不包括 mp:也就是[a-lq-z]

预定义字符

.	任意字符，有可能包括换行符
/d	0 到 9 的数字
/D	0 到 9 以外的字符
/s	空格符[/t/n/x0B/f/r]
/S	非空格符[^/s]
/w	字母[a-zA-Z_0-9]
/W	非字母[^/w]

边界匹配

^	行首
$	行末
/b	单词边界

加楠楠 QQ1692250804 获取跟多学习资料

/A	输入的起始位置
/G	前一个匹配的末尾
/Z	输入的结束位置，仅用于最后的结束符
/z	输入的结束位置

```
import java.util.regex.Matcher;

import java.util.regex.Pattern;



public class RegexMatches

{

   private static String pattern =  "^[_A-Za-z0-9-]+(//.[_A-Za-z0-9-]+)*@[A-

Za-z0-9]+(//.[A-Za-z0-9]+)*(//.[A-Za-z]{2,})$";

   private static Pattern mypattern = Pattern.compile(pattern);



   public static void main( String args[] ){



      String valEmail1 = "testemail@domain.com";

      String invalEmail1 = "....@domain.com";
```

```
        String invalEmail2 = ".$$%%@domain.com";

        String valEmail2 = "test.email@domain.com";



        System.out.println("Is Email ID1 valid? "+validateEMailID(valEmail
1));

        System.out.println("Is Email ID1 valid? "+validateEMailID(invalEmail
1));

        System.out.println("Is Email ID1 valid? "+validateEMailID(invalEmail
2));

        System.out.println("Is Email ID1 valid? "+validateEMailID(valEmail
2));



    }



    public static boolean validateEMailID(String emailID) {

        Matcher mtch = mypattern.matcher(emailID);

        if(mtch.matches()){

            return true;
```

```
        }

        return false;

    }

}
```

Java Swing 的简单示例

有了 Java 的 swing ,你便可以编写 GUI 应用了。Java 所提供的 javax 包中就包含了 swing。使用 swing 来编写 GUI 程序首先需要继承下 JFrame。然后在里面添加 Box，然后便可以往里面添加诸如按钮，多选按钮，文本框等控件了。这些 Box 是放在 Container 的最外层的。

```
import java.awt.*;

import javax.swing.*;



public class SwingsDemo extends JFrame

{

    public SwingsDemo()

    {
```

```
        String path = "//home//user//Documents//images";

        Container contentPane = getContentPane();

        contentPane.setLayout(new FlowLayout());



        Box myHorizontalBox = Box. createHorizontalBox();

        Box myVerticleBox = Box. createVerticalBox();



        myHorizontalBox.add(new JButton("My Button 1"));

        myHorizontalBox.add(new JButton("My Button 2"));

        myHorizontalBox.add(new JButton("My Button 3"));



        myVerticleBox.add(new JButton(new ImageIcon(path + "//Image1.jpg")));


        myVerticleBox.add(new JButton(new ImageIcon(path + "//Image2.jpg")));


        myVerticleBox.add(new JButton(new ImageIcon(path + "//Image3.jpg")));


        contentPane.add(myHorizontalBox);
```

加楠楠 QQ1692250804 获取跟多学习资料

```
        contentPane.add(myVerticleBox);



        pack();

        setVisible(true);

    }



    public static void main(String args[]) {

        new SwingsDemo();

    }

}
```

使用 Java 播放音频

在 Java 中，播放音频是一个很常见的需求，尤其是在游戏开发里面。

下面这个 DEMO 演示了如何在 Java 中播放音频。

```
import java.io.*;

import java.net.URL;

import javax.sound.sampled.*;
```

```
import javax.swing.*;



// To play sound using Clip, the process need to be alive.

// Hence, we use a Swing application.

public class playSoundDemo extends JFrame {



   // Constructor

   public playSoundDemo() {

      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      this.setTitle("Play Sound Demo");

      this.setSize(300, 200);

      this.setVisible(true);



      try {

         URL url = this.getClass().getResource("MyAudio.wav");

         AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);

         Clip clip = AudioSystem.getClip();
```

```
        clip.open(audioIn);

        clip.start();

      } catch (UnsupportedAudioFileException e) {

        e.printStackTrace();

      } catch (IOException e) {

        e.printStackTrace();

      } catch (LineUnavailableException e) {

        e.printStackTrace();

      }

    }



  public static void main(String[] args) {

      new playSoundDemo();

    }

}
```

导出 PDF 文件

将表格导出成 pdf 也是一个比较常见的需求。通过 itextpdf，导出 pdf 也不是什么难事。

```
import java.io.FileOutputStream;

import com.itextpdf.text.Document;

import com.itextpdf.text.Paragraph;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;



public class DrawPdf {



    public static void main(String[] args) throws Exception {

        Document document = new Document();

        PdfWriter.getInstance(document, new FileOutputStream("Employee.pdf
"));

        document.open();



        Paragraph para = new Paragraph("Employee Table");
```

```
        para.setSpacingAfter(20);

        document.add(para);



        PdfPTable table = new PdfPTable(3);

        PdfPCell cell = new PdfPCell(new Paragraph("First Name"));



        table.addCell(cell);

        table.addCell("Last Name");

        table.addCell("Gender");

        table.addCell("Ram");

        table.addCell("Kumar");

        table.addCell("Male");

        table.addCell("Lakshmi");

        table.addCell("Devi");

        table.addCell("Female");



        document.add(table);
```

加楠楠 QQ1692250804 获取跟多学习资料

```
        document.close();

    }

}
```

邮件发送

在 Java 中发送邮件也很简单。你只需装一下 Java Mail 这个 jar 包，放到你的类路径里即可。在下面的代码中，我们设置了几个基础属性，然后便可以发送邮件了：

```
import java.util.*;

import javax.mail.*;

import javax.mail.internet.*;



public class SendEmail

{

   public static void main(String [] args)

   {

        String to = "recipient@gmail.com";
```

```
        String from = "sender@gmail.com";

        String host = "localhost";



        Properties properties = System.getProperties();

        properties.setProperty("mail.smtp.host", host);

        Session session = Session.getDefaultInstance(properties);



        try{

            MimeMessage message = new MimeMessage(session);

            message.setFrom(new InternetAddress(from));



            message.addRecipient(Message.RecipientType.TO,new InternetAddress
(to));



            message.setSubject("My Email Subject");

            message.setText("My Message Body");

            Transport.send(message);

            System.out.println("Sent successfully!");
```

```
        }

        catch (MessagingException ex) {

            ex.printStackTrace();

        }

    }

}
```

计算时间

许多程序都需要精确的时间计量。Java 提供了一个 System 的静态方法来支持这一功能：

currentTimeMillis()：返回当前时间自新纪元时间以来的毫秒值，long 类型。

```
long startTime = System.currentTimeMillis();

long estimatedTime = System.currentTimeMillis() - startTime;
```

nanoTime()：返回系统计时器当前的精确时间，纳秒值，这也是 long 类型。nanoTime() 主要是用于计算相对时间而非绝对时间。

```
long startTime = System.nanoTime();

long estimatedTime = System.nanoTime() - startTime;
```

图片缩放

图片缩放可以通过 AffineTransform 来完成。首先要生成一个输入图片的图片缓冲，然后通过它来渲染出缩放后的图片。

```
import java.awt.Graphics2D;

import java.awt.geom.AffineTransform;

import java.awt.image.BufferedImage;

import java.io.File;

import javax.imageio.ImageIO;



public class RescaleImage {

  public static void main(String[] args) throws Exception {

    BufferedImage imgSource = ImageIO.read(new File("images//Image3.jpg"));

    BufferedImage imgDestination = new BufferedImage(100, 100, BufferedImage.
TYPE_INT_RGB);

    Graphics2D g = imgDestination.createGraphics();

    AffineTransform affinetransformation = AffineTransform.getScaleInstance
(2, 2);
```

```
    g.drawRenderedImage(imgSource, affinetransformation);

    ImageIO.write(imgDestination, "JPG", new File("outImage.jpg"));

  }

}
```

捕获鼠标动作

实现了 MouseMotionListner 接口后，便可以捕获鼠标事件了。 当鼠标进入到某个特定区域时便会触发 MouseMoved 事件，你便能捕获到这个移动的动作了。通过一个例子来看下：

```
import java.awt.event.*;

import javax.swing.*;



public class MouseCaptureDemo extends JFrame implements MouseMotionListener

{

    public JLabel mouseHoverStatus;



    public static void main(String args[])

    {
```

```
        new MouseCaptureDemo();

    }


    MouseCaptureDemo()

    {

        setSize(500, 500);

        setTitle("Frame displaying Coordinates of Mouse Motion");


        mouseHoverStatus = new JLabel("No Mouse Hover Detected.", JLabel.CENTE
R);

        add(mouseHoverStatus);

        addMouseMotionListener(this);

        setVisible(true);

    }


    public void mouseMoved(MouseEvent e)

    {
```

加楠楠 QQ1692250804 获取跟多学习资料

```
        mouseHoverStatus.setText("Mouse Cursor Coordinates => X:"+e.getX()+"
| Y:"+e.getY());

    }



    public void mouseDragged(MouseEvent e)

    {}

}
```

FileOutputStream Vs. FileWriter

在 Java 中有两种写文件的方式：FileOutputStream 与 FileWriter。开发人员经常会在它们之间犹豫不决。下面这个例子能帮忙你更好地理解在不同的场景下应该选择何种方案。首先我们来看一下实现：

使用 FileOutputStream：

```
File foutput = new File(file location string);

FileOutputStream fos = new FileOutputStream(foutput);

BufferedWriter output = new BufferedWriter(new OutputStreamWriter(fos));

output.write("Buffered Content");
```

使用 FileWriter：

```
FileWriter fstream = new FileWriter(file_location_string);

BufferedWriter output = new BufferedWriter(fstream);

output.write("Buffered Content");
```

根据 Java 的接口规范：

FileOutputStream 是用于写入原始字节流比如图片流数据。如果是要写入字符流，则应该考虑使用 FileWriter。

这样就很清楚了，写图片应该使用 FileOutputStream 而写文本则应该选择 FileWriter。

附加建议

集合的使用

Java 提供了许多集合类——比如，Vector，Stack，Hashtable 等。所以鼓励开发人员尽可能地使用这些集合类有如下原因：

- 使用集合使得代码的可重用度更高。
- 集合类使得代码的结构更良好，更易于理解与维护。
- 最重要的是这些集合类都经过充分的测试，代码质量很高。

1-50-500 规则

在大型软件系统中，代码的可维护性是件很有挑战的工作。新加入的开发人员经常会抱怨这些情况：单片代码（Monolithic Code）,意大利面式代码（spaghetti code，常用于描述捆绑在一起并且低内聚的类和方法）。保持代码的整洁与可维护有一条很简单的规则：

- 10：包内的类不超过 10 个
- 50：方法的代码行数不超过 50
- 500：类的代码行数不超过 500

1. SOLID 设计准则
2. SOLID 是 Robert Martin 提出的一套设计准则的简称。根据他的准则：

一个类应当有仅只有一个任务/职责。执行多个任务的类会让人觉得困惑。

单一职责原则	
开闭原则	开发人员应当优先考虑扩展现有的软件功能，而不是是修改它。
里氏替换原则	子类必须能够替换掉他们的父类型
接口隔离原则	和单一职责原则类似，但它特指的是接口层。每个接口都应当只负责一项任务。
依赖反转原	依赖抽象而不是具体实现。也就是说每个模块都应当通过一个抽象层与其它模块进行

则	解耦。

设计模式的使用

设计模式能帮助开发人员更好地在软件中应用软件的设计准则。它还为开发人员提供了跨语言的通用平台。设计模式中的标准术语能让开发人员更容易进行沟通。

关于文档

不要上来就开始写代码。制定计划，准备，编写文档，检查然后再去实现。首先，先把需求记下来。然后去准备设计文档。合理地去假设举证。互相 review 方案然后进行确认。

使用 equals 而非==

==是用来比较对象引用的，它会检查两个操作数指向的是不是同一个对象（不是相同的对象，而是同一个对象）。而”equals”则比较的是两个字符串是不是相同（假设是字符串对象）。

避免使用浮点数

只有当确实有必要的时候才使用浮点数。比方说，使用浮点数来表示卢比或者派萨就很容易产生问题——这种情况应当使用 BigDecimal。而浮点数更多地是用于测量。