

# 2019年最新Java面试题及答案整理（上）

原创 Java程序员-张凯 最后发布于2019-07-15 16:25:48 阅读数 34476 ☆ 收藏

展开

## 1、面向对象的特征有哪些方面？

答：面向对象的特征主要有以下几个方面：

- **抽象**：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。
- **继承**：继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段（如果不能理解请阅读阎宏博士的《Java与模式》或《设计模式精解》中关于桥梁模式的部分）。
- **封装**：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。
- **多态性**：多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当A系统访问B系统提供的服务时，B系统有多种提供服务的方式，但一切对A系统来说都是透明的（就像电动剃须刀是A系统，它的供电系统是B系统，B系统可以使用电池供电或者用交流电，甚至还有可能是太阳能，A系统只会通过B类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：1). 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；2). 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

## 2、访问修饰符public,private,protected,以及不写（默认）时的区别？

答：



类的成员不写访问修饰符默认为default。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。Java中，外部类的修饰符只能是public或默认，类的成员（包括内部类）的修饰符可以是以上四种。

## 3、String是最基本的数据类型吗？

答：不是。Java中的基本数据类型只有8个：byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type），剩下的都是引用类型（reference type），Java 5以后引入的枚举类型也算是一种比较特殊的引用类型。

## 4、float f=3.4;是否正确？

答:不正确。3.4是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换float f =(float)3.4; 或者写成float f =3.4F;。

## 5、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？

答：对于short s1 = 1; s1 = s1 + 1;由于1是int类型，因此s1+1运算结果也是int 型，需要强制转换类型才能赋值给short型。而short s1 = 1; s1 += 1;可以正确编译，因为s1+= 1;相当于s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

示例代码如下：

- 1 前者有错，s1会自动提升为int类型，结果赋值给short类型，所以报错。
- 2 后者无错，+=这种赋值运算符隐含了强制类型转换。其实变量s1的值未被使用。
- 3

```
4 | public class Test {
5 |     public static void main(String[] args) {
6 |         System.out.println(Math.round(11.5));    // 12
7 |         System.out.println(Math.round(-11.5));    // -11
8 |
9 |         // short s1 = 1;
10 |        // s1 = s1 + 1; // Type mismatch: cannot convert from int to short 类型不匹配: 不能从int转换为short
11 |
12 |        short s1 = 1; // The value of the local variable s1 is not used 局部变量s1的值未被使用 出现了警告
13 |        s1 += 1;
14 |    }
15 | }
```

## 6、Java有没有goto?

答: goto 是Java中的保留字, 在目前版本的Java中没有使用。(根据James Gosling (Java之父) 编写的《The Java Programming Language》一书的附录中给出了一个Java关键字列表, 其中有goto和const, 但是这两个是目前无法使用的关键字, 因此有些地方将其称之为保留字, 其实保留字这个词应该有更广泛的意义, 因为熟悉C语言的程序员都知道, 在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字)

## 7、int和Integer有什么区别?

答: Java是一个近乎纯洁的面向对象编程语言, 但是为了编程的方便还是引入了基本数据类型, 但是为了能够将这些基本数据类型当成对象操作, Java为每一个基本数据类型都引入了对应的包装类型(wrapper class), int的包装类就是Integer, 从Java 5开始引入了自动装箱/拆箱机制, 使得二者可以相互转换。

Java 为每个原始类型提供了包装类型:

- 原始类型: boolean, char, byte, short, int, long, float, double
- 包装类型: Boolean, Character, Byte, Short, Integer, Long, Float, Double

示例代码如下:

```
1 | class AutoUnboxingTest {
2 |
3 |     public static void main(String[] args) {
4 |         Integer a = new Integer(3);
5 |         Integer b = 3;                // 将3自动装箱成Integer类型
6 |         int c = 3;
7 |         System.out.println(a == b);    // false 两个引用没有引用同一对象
8 |         System.out.println(a == c);    // true  a自动拆箱成int类型再和c比较
9 |     }
10 | }
```

最近还遇到一个面试题, 也是和自动装箱和拆箱有点关系的, 代码如下所示:

```
1 | public class Test03 {
2 |
3 |     public static void main(String[] args) {
4 |         Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;
5 |
6 |         System.out.println(f1 == f2);
7 |         System.out.println(f3 == f4);
8 |     }
9 | }
```

如果不明就里很容易认为两个输出要么都是true要么都是false。首先需要注意的是f1、f2、f3、f4四个变量都是Integer对象引用, 所以下

面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个Integer对象赋一个int值的时候，会调用Integer类的静态方法valueOf，如果看看valueOf的源代码就知道发生了什么。

```

1 public static Integer valueOf(int i) {
2     if (i >= IntegerCache.low && i <= IntegerCache.high)
3         return IntegerCache.cache[i + (-IntegerCache.low)];
4     return new Integer(i);
5 }

```

IntegerCache是Integer的内部类，其代码如下所示：

```

1 /**
2  * Cache to support the object identity semantics of autoboxing for values between
3  * -128 and 127 (inclusive) as required by JLS.
4  *
5  * The cache is initialized on first usage. The size of the cache
6  * may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.
7  * During VM initialization, java.lang.Integer.IntegerCache.high property
8  * may be set and saved in the private system properties in the
9  * sun.misc.VM class.
10 */
11
12 private static class IntegerCache {
13     static final int low = -128;
14     static final int high;
15     static final Integer cache[];
16
17     static {
18         // high value may be configured by property
19         int h = 127;
20         String integerCacheHighPropValue =
21             sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
22         if (integerCacheHighPropValue != null) {
23             try {
24                 int i = parseInt(integerCacheHighPropValue);
25                 i = Math.max(i, 127);
26                 // Maximum array size is Integer.MAX_VALUE
27                 h = Math.min(i, Integer.MAX_VALUE - (-low) - 1);
28             } catch (NumberFormatException nfe) {
29                 // If the property cannot be parsed into an int, ignore it.
30             }
31         }
32         high = h;
33
34         cache = new Integer[(high - low) + 1];
35         int j = low;
36         for(int k = 0; k < cache.length; k++)
37             cache[k] = new Integer(j++);
38
39         // range [-128, 127] must be interned (JLS7 5.1.7)
40         assert IntegerCache.high >= 127;
41     }
42
43     private IntegerCache() {}
44 }

```

简单的说，如果整型字面量的值在-128到127之间，那么不会new新的Integer对象，而是直接引用常量池中的Integer对象，所以上面的面试题中f1==f2的结果是true，而f3==f4的结果是false。

**提醒：**越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。

## 8、&和&&的区别？

答：&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true整个表达式的值才是true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是null而且不是空字符串，应当写为：username != null && !username.equals("")，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的equals比较，否则会产生NullPointerException异常。注意：逻辑或运算符（||）和短路或运算符（|||）的差别也是如此。

**补充：**如果你熟悉JavaScript，那你可能更能感受到短路运算的强大，想成为JavaScript的高手就先从玩转短路运算开始吧。

## 9、解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法。

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用JVM中的栈空间；而通过new关键字和构造器创建的对象则放在堆空间，堆是垃圾收集器管理的主要区域，由于现在的垃圾收集器都采用分代收集算法，所以堆空间还可以细分为新生代和老生代，再具体一点可以分为Eden、Survivor（又可分为From Survivor和To Survivor）、Tenured；方法区和堆都是各个线程共享的内存区域，用于存储已经被JVM加载的类信息、常量、静态变量、JIT编译器编译后的代码等数据；程序中的字面量（literal）如直接书写的100、“hello”和常量都是放在常量池中，常量池是方法区的一部分，。栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间，栈和堆的大小都可以通过JVM的启动参数来进行调整，栈空间用光了会引发StackOverflowError，而堆和常量池空间不足则会引发OutOfMemoryError。

```
String str = new String("hello");
```

上面的语句中变量str放在栈上，用new创建出来的字符串对象放在堆上，而“hello”这个字面量是放在方法区的。

补充

换等

补充

新的

原创

粉丝

获赞

评论

访问

242

2287

1492

246

60万+

等级: 博客 6

周排名: 228

积分: 7464

总排名: 4019

勋章:   

看看下面

Stri

时常量

和jdk1.7

关注

私信

```
1 String s1 = new StringBuilder("go").append("od").toString();
2 System.out.println(s1.intern() == s1);
3 String s2 = new StringBuilder("ja").append("va").toString();
4 System.out.println(s2.intern() == s2);
```

开始)中，由于JIT编译器的发展和“逃逸分析”技术的逐渐成熟，栈上分配、标量替换已经变得不那么绝对了。

!有动态性，Java语言并不要求常量一定只有编译期间才能产生，运行期间也可以有这样的。

以前和以后的运行结果是否一致。

如果字符串常量池中已经包含了一个等于此String对象的字符串，则返回代表池(运行string对象包含的字符串添加到常量池中并且返回此String对象的引用。此方法在jdk1.6

这段代码在jdk1.6中运行，会得到两个false，而在jdk1.7中运行会得到一个true一个false。产生差异的原因是：在jdk1.6中，intern()方法会把首次遇到的字符串实例复制到永久代中，返回的也是永久代中这个字符串实例的引用，而用StringBuilder创建的字符串实例在Java堆上，所以必然不是同一个引用，将返回false。而jdk1.7中的intern()实现不会再复制实例，只是在常量池中记录首次出现的实例引用，因此intern()返回的引用和由StringBuilder创建的那个字符串实例是同一个。对比str2返回false是因为“java”这个字符串在执行StringBuilder.toString()之前已经出现过，字符串常量池中已经有它的引用了，不符合首次出现的原则，而“good”这个字符串则是首次出现的，因此返回true。

现在的疑问是“java”这个字符串在常量池中什么时候存在了？

量池中的？那为什么常驻在常量池中呢？Java虚拟机什么时候加载了“java”这个字符？

下：

ic initializer.  
 emClass method to complete  
 separated from clinit.  
 the VM, see the constraints  
 Class method.

$\Delta \text{CO}_2 = \text{CO}_2 - \text{CO}_2^{\text{ref}}$  在此公式中油田的  $\text{CO}_2$  和  $\text{CO}_2^{\text{ref}}$  分别为油田最大产量



```
_name = "java";
_version = "1.7.0_51";
_runtime_name = "Java(TM) SE Runtime Environment";
_runtime_version = "1.7.0_51-b13";

// 指定的常量值做默认初始化，所以"java"被加载到了字符串常量池中，修改上面代码使字符串值为
// "java_51"
sb.append("_51").toString();

return sb.toString();
}
```

赏

“永久代”，但两者本质上并不等价，仅仅是因为HotSpot虚拟机的设计团队选择把永久代实现方法区而已，但现在看来使用永久代实现方法区并不是一个好主意，因为在HotSpot中，已经把原本放在永久代中的字符串常量池移除——摘自《深入理解Java虚拟机》

·11.5)的返回值是-11。四舍五入的原理是在参数上加0.5然后进行下取整。

### 四舍五入(参数为double的自学)

的值大;  
的值小;

否能作用在long上，是否能作用在String上？

te、short、char、int；从Java 5开始，Java中引入了枚举类型，expr也可以是enum类，但是长整型（long）在目前所有的版本中都是不可以的。

位相当于除以2的3次方)。

可能会看到如下所示的代码，其实我们不太理解为什么要使用这样的乘法运算来产生掩码，为什么通常选择31这个数？前两个问题的答案你可以自己百度一下，选择31是因

为可  
移5位

关于我们 招聘 广告服务 网站地图  
京ICP备19004658号 经营性网站备案信息  
公安备案号 11010502030143  
©1999-2020 北京创新乐知网络技术有限公司  
网络110报警服务  
北京互联网违法和不良信息举报中心  
中国互联网举报中心 家长监护 版权申诉

更好的性能。说到这里你可能已经想到了：31 \* num 等价于(num <= 5) - num，左乘以31，现在的VM都能自动完成这个优化。



```
1 public int hashCode() {
2     final int prime = 31;
3     int result = 1;
4     result = prime * result + areaCode;
5     result = prime * result
6         + ((lineNumber == null) ? 0 : lineNumber.hashCode());
7     result = prime * result + ((prefix == null) ? 0 : prefix.hashCode());
8     return result;
9 }
10
11 @Override
12 public boolean equals(Object obj) {
13     if (this == obj)
14         return true;
15     if (obj == null)
16         return false;
17     if (getClass() != obj.getClass())
18         return false;
19     PhoneNumber other = (PhoneNumber) obj;
20     if (areaCode != other.areaCode)
21         return false;
22     if (lineNumber == null) {
23         if (other.lineNumber != null)
24             return false;
25     } else if (!lineNumber.equals(other.lineNumber))
26         return false;
27     if (prefix == null) {
28         if (other.prefix != null)
29             return false;
30     } else if (!prefix.equals(other.prefix))
31         return false;
32     return true;
33 }
34 }
```

### 13、数组有没有length()方法？String有没有length()方法？

答：数组没有length()方法，有length的属性。

String有length()方法。在JavaScript中，获得字符串的长度是通过length属性得到的，这一点容易和Java混淆。

### 14、在Java中，如何跳出当前的多重嵌套循环？

答：在最外层循环前加一个标记如A，然后用break A;可以跳出多重循环。（Java中支持带标签的break和continue语句，作用有点类似于C和C++中的goto语句，但是就像要避免使用goto一样，应该避免使用带标签的break和continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好，为什么会让程序变得不优雅呢？一个程序跳来跳去，太灵活了，我们不能够控制了，就不好了）



## 15、构造器（constructor）是否可被重写（override）？

答：构造器不能被继承，因此不能被重写，但可以被重载。

## 16、两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对？

答：不对，如果两个对象x和y满足x.equals(y) == true，它们的哈希码（hash code）应当相同。Java对于equals方法和hashCode方法是这样规定的：

- (1) 如果两个对象相同（equals方法返回true），那么它们的hashCode值一定要相同；
- (2) 如果两个对象的hashCode相同，它们并不一定相同。

当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在Set集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

补充：关于equals和hashCode方法，很多Java程序都知道，但很多人也就是仅仅知道而已，在Joshua Bloch的大作《Effective Java》（很多软件公司，《Effective Java》、《Java编程思想》以及《重构：改善既有代码质量》是Java程序员必看书籍，如果你还没看过，那就赶紧去亚马逊买一本吧）中是这样介绍equals方法的：首先equals方法必须满足自反性（x.equals(x)必须返回true）、对称性（x.equals(y)返回true时，y.equals(x)也必须返回true）、传递性（x.equals(y)和y.equals(z)都返回true时，x.equals(z)也必须返回true）和一致性（当x和y引用的对象信息没有被修改时，多次调用x.equals(y)应该得到同样的返回值），而且对于任何非null值的引用x，x.equals(null)必须返回false。

实现高质量的equals方法的诀窍包括：

1. 使用==操作符检查"参数是否为这个对象的引用"；
2. 使用instanceof操作符检查"参数是否为正确的类型"；
3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；
4. 编写完equals方法后，问自己它是否满足对称性、传递性、一致性；
5. 重写equals时总是要重写hashCode；
6. 不要将equals方法参数中的Object对象替换为其他的类型，在重写时不要忘掉@Override注解。

## 17、是否可以继承String类？

答：String 类是final类，不可以被继承。

补充：继承String本身就是一个错误的行为，对String类型最好的重用方式是关联关系（Has-A）和依赖关系（Use-A）而不是继承关系（Is-A）。

## 18、当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递？

答：是值传递。Java语言的方法调用只支持参数的值传递。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用（地址值）。对象的属性可以在被调用过程中被改变，但对对象引用的改变是不会影响到调用者的。C++和C#中可以通过传引用或传输出参数来改变传入的参数的值。在C#中可以编写如下所示的代码，但是在Java中却做不到。

```
1 using System;
2
3 namespace CS01 {
4
5     class Program {
6         public static void swap(ref int x, ref int y) {
7             int temp = x;
8             x = y;
9             y = temp;
10        }
11
12        public static void Main (string[] args) {
13            int a = 5, b = 10;
```

```
14 |         swap (ref a, ref b);
    |         15 |         // a = 10, b = 5;
16 |         Console.WriteLine ("a = {0}, b = {1}", a, b);
17 |     }
18 | }
19 | }
```

说明：Java中没有传引用实在是非常的不方便，这一点在Java 8中仍然没有得到改进，正是如此在Java编写的代码中才会出现大量的Wrapper类（将需要通过方法调用修改的引用置于一个Wrapper类中，再将Wrapper对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从C和C++转型为Java程序员的开发者无法容忍。

其实还是不够明白，我们来看看下面的例子吧：

强烈推荐鄙人的例子，参考链接：<https://www.cnblogs.com/chenmingjun/p/8698719.html>

## 19、String和StringBuilder、StringBuffer的区别？

答：Java平台提供了两种类型的字符串：String和StringBuffer/StringBuilder，它们可以储存和操作字符串。其中String是只读字符串，也就意味着String引用的字符串内容是不能被改变的。而StringBuffer/StringBuilder类表示的字符串对象可以直接进行修改。StringBuilder是Java 5中引入的，它和StringBuffer的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被synchronized修饰（非同步），因此它的效率也比StringBuffer要高。

面试题1：什么情况下用+运算符进行字符串连接比调用StringBuffer/StringBuilder对象的append方法连接字符串性能更好？

面试题2：请说出下面程序的输出。

```
1 | class StringEqualTest {
2 |
3 |     public static void main(String[] args) {
4 |         String s1 = "Programming";
5 |         String s2 = new String("Programming");
6 |         String s3 = "Program";
7 |         String s4 = "ming";
8 |         String s5 = "Program" + "ming";
9 |         String s6 = s3 + s4;
10 |         System.out.println(s1 == s2); // false
11 |         System.out.println(s1 == s5); // true
12 |         System.out.println(s1 == s6); // false
13 |         System.out.println(s1 == s6.intern()); // true
14 |         System.out.println(s2 == s2.intern()); // false
15 |     }
16 | }
```

补充：解答上面的面试题需要清除两点：

1. String对象的intern方法会得到字符串对象在常量池中对应的版本的引用（如果常量池中有一个字符串与String对象的equals结果是true），如果常量池中并没有对应的字符串，则该字符串将被添加到常量池中，然后返回常量池中字符串的引用。
2. 字符串的+操作其本质是创建了StringBuilder对象进行append操作，然后将拼接后的StringBuilder对象用toString方法处理成String对象，这一点可以用javap -c StringEqualTest.class命令获得class文件对应的JVM字节码指令就可以看出来。
3. 要想获取对象的内存地址应使用System.identityHashCode()方法。

## 20、重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？

答：方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载发生在一个



类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求。

**面试题：**华为的面试题中曾经问过这样一个问题：“为什么不能根据返回类型来区分重载”，快说出你的答案吧！

## 21、描述一下JVM加载class文件的原理机制？

答：JVM中类的装载是由类加载器（ClassLoader）和它的子类来实现的，Java中的类加载器是一个重要的Java运行时系统组件，它负责在运行时查找和装入类文件中的类。

由于Java的跨平台性，经过编译的Java源程序并不是一个可执行程序，而是一个或多个类文件。当Java程序需要使用某个类时，JVM会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class文件中的数据读入到内存中，通常是创建一个字节数组读入.class文件，然后产生与所加载类对应的Class对象。加载完成后，Class对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后JVM对类进行初始化，包括：1) 如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2) 如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：根加载器（Bootstrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader的子类）。从Java 2（JDK 1.2）开始，类加载过程采取了父亲委托机制（PDM）。PDM更好的保证了Java平台的安全性，在该机制中，JVM自带的Bootstrap是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM不会向Java程序提供对Bootstrap的引用。下面是关于几个类加载器的说明：

**Bootstrap：**一般用本地代码实现，负责加载JVM基础核心类库（rt.jar）；

**Extension：**从java.ext.dirs系统属性所指定的目录中加载类库，它的父加载器是Bootstrap；

**System：**又叫应用类加载器，其父类是Extension。它是应用最广泛的类加载器。它从环境变量classpath或者系统属性java.class.path所指定的目录中记载类，是用户自定义加载器的默认父加载器。

## 22、char 型变量中能不能存贮一个中文汉字，为什么？

答：char类型可以存储一个中文汉字，因为Java中使用的编码是Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法），一个char类型占2个字节（16比特），所以放一个中文是没问题的。

**补充：**使用Unicode意味着字符在JVM内部和外部有不同的表现形式，在JVM内部都是Unicode，当这个字符被从JVM内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以Java中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如InputStreamReader和OutputStreamReader，这两个类是字节流和字符流之间的适配器类，承担了编码转换的任务；对于C程序员来说，要完成这样的编码转换恐怕要依赖于union（联合体/共用体）共享内存的特征来实现了。

## 23、抽象类（abstract class）和接口（interface）有什么异同？

答：抽象类和接口都不能够实例化，但可以定义抽象类和接口类型的引用。一个类如果继承了某个抽象类或者实现了某个接口都需要对其中的抽象方法全部进行实现，否则该类仍然需要被声明为抽象类。接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法，而接口中不能定义构造器而且其中的方法全部都是抽象方法。抽象类中的成员可以是private、默认、protected、public的，而接口中的成员全都是public的。抽象类中可以定义成员变量，而接口中定义的成员变量实际上都是常量。有抽象方法的类必须被声明为抽象类，而抽象类未必要有抽象方法。

## 24、静态嵌套类（Static Nested Class）和内部类（Inner Class）的不同？

答：Static Nested Class是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化，其语法看起来挺诡异的。

**面试题：**下面的代码哪些地方会产生编译错误？

```
1 class Outer {
2
3     class Inner {}
4
5     public static void foo() {
6         new Inner();
7     }
8
9     public void bar() {
10        new Inner();
11    }
12
13    public static void main(String[] args) {
14        new Inner();
15    }
16 }
```

**注意：**Java中非静态内部类对象的创建要依赖其外部类对象，上面的面试题中foo和main方法都是静态方法，静态方法中没有this，也就是说没有所谓的外部类对象，因此无法创建内部类对象，如果要在静态方法中创建内部类对象，可以这样做：new Outer().new Inner();

## 25、Java中会存在内存泄漏吗？请简单描述。

答：理论上Java因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是Java被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被GC回收，因此也会导致内存泄露的发生。例如Hibernate的Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象，如果不及时关闭（close）或清空（flush）一级缓存就可能导致内存泄露。下面例子中的代码也会导致内存泄露。

```
1 import java.util.Arrays;
2 import java.util.EmptyStackException;
3
4 public class MyStack<T> {
5     private T[] elements;
6     private int size = 0;
7
8     private static final int INIT_CAPACITY = 16;
9
10    public MyStack() {
11        elements = (T[]) new Object[INIT_CAPACITY];
12    }
13
14    public void push(T elem) {
15        ensureCapacity();
16        elements[size++] = elem;
17    }
18
19    public T pop() {
20        if(size == 0)
21            throw new EmptyStackException();
22        return elements[--size];
23    }
24
25    private void ensureCapacity() {
26        if(elements.length == size) {
27            elements = Arrays.copyOf(elements, 2 * size + 1);
28        }
29    }
30 }
```

```
29 | }  
    | 30 | }
```

上面的代码实现了一个栈（先进后出（FILO））结构，乍看之下似乎没有什么明显的问题，它甚至可以通过你编写的各种单元测试。然而其中的pop方法却存在内存泄露的问题，当我们用pop方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部维护着对这些对象的过期引用（obsolete reference）。在支持垃圾回收的语言中，内存泄露是很隐蔽的，这种内存泄露其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理该对象引用的其他对象，即使这样的对象只有少数几个，也可能导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下会引发Disk Paging（物理内存与硬盘的虚拟内存交换数据），甚至造成OutOfMemoryError。

## 26、抽象的（abstract）方法是否可同时是静态的（static），是否可同时是本地方法（native），是否可同时被synchronized修饰？

答：都不能。抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。本地方法是由本地代码（如C代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。synchronized和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

## 27、阐述静态变量和实例变量的区别。

答：静态变量是被static修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它。静态变量可以实现让多个对象共享内存。

补充：在Java开发中，上下文类和工具类中通常会有大量的静态成员。

## 28、是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

答：不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，在调用静态方法时可能对象并没有被初始化。

## 29、如何实现对象克隆？

答：有两种方式：

- 1) 实现Cloneable接口并重写Object类中的clone()方法；
- 2) 实现Serializable接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆。

注意：基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化，这项检查是编译器完成的，不是在运行时抛出异常，这种方案明显优于使用Object类的clone方法克隆对象。让问题在编译的时候暴露出来总是好过把问题留到运行时。

## 30、GC是什么？为什么要有GC？

答：GC是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。Java程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或 Runtime.getRuntime().gc()，但JVM可以屏蔽掉显示的垃圾回收调用。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在Java诞生初期，垃圾回收是Java最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今Java的垃圾回收机制已经成为被诟病的東西。移动智能终端用户通常觉得iOS的系统比Android系统有更好的用户体验，其中一个深层次的原因就在于Android系统中垃圾回收的不可预知性。

补充：垃圾回收机制有很多种，包括：分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。标准的Java进程既有栈又有堆。栈保存了原始型局部变量，堆保存了要创建的对象。Java平台对堆内存回收和再利用的基本算法被称为标记和清除，但是Java对其

进行了改进，采用“分代式垃圾收集”。这种方法会跟据Java对象的生命周期将堆内存划分为不同的区域，在垃圾收集过程中，可能会将对象移动到不同区域：

- 伊甸园（Eden）：这是对象最初诞生的区域，并且对大多数对象来说，这里是它们唯一存在过的区域。
- 幸存者乐园（Survivor）：从伊甸园幸存下来的对象会被挪到这里。
- 终身颐养园（Tenured）：这是足够老的幸存对象的归宿。年轻代收集（Minor-GC）过程是不会触及这个地方的。当年轻代收集不能把对象放进终身颐养园时，就会触发一次完全收集（Major-GC），这里可能还会牵扯到压缩，以便为大对象腾出足够的空间。

与垃圾回收相关的JVM参数：

- -Xms / -Xmx --> 堆的初始大小 / 堆的最大大小
- -Xmn --> 堆中年轻代的大小
- -XX:-DisableExplicitGC --> 让System.gc()不产生任何作用
- -XX:+PrintGCDetails --> 打印GC的细节
- -XX:+PrintGCDateStamps --> 打印GC操作的时间戳
- -XX:NewSize --> 设置新生代大小
- -XX:MaxNewSize --> 设置新生代最大大小
- -XX:NewRatio --> 可以设置老生代和新生代的比例
- -XX:PrintTenuringDistribution --> 设置每次新生代GC后输出幸存者乐园中对象年龄的分布
- -XX:InitialTenuringThreshold --> 设置老年代阈值的初始值
- -XX:MaxTenuringThreshold --> 设置老年代阈值的最大值
- -XX:TargetSurvivorRatio --> 设置幸存者区的目标使用率

### 31、String s = new String("xyz");创建了几个字符串对象？

答：两个对象，一个是静态区的"xyz"，一个是用new创建在堆上的对象。

### 32、接口是否可继承（extends）接口？抽象类是否可实现（implements）接口？抽象类是否可继承具体类（concrete class）？

答：接口可以继承接口，而且支持多重继承。抽象类可以实现(implements)接口，抽象类可继承具体类也可以继承抽象类。

### 33、一个".java"源文件中是否可以包含多个类（不是内部类）？有什么限制？

答：可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

### 34、Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？

答：可以继承其他类或实现其他接口，在Swing编程和Android开发中常用此方式来实现事件监听和回调。

### 35、内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？

答：一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员。

### 36、Java 中的final关键字有哪些用法？

答：

- (1)修饰类：表示该类不能被继承；
- (2)修饰方法：表示方法不能被重写；
- (3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

### 37、指出下面程序的运行结果。

```
1 class A {
2
3     static {
4         System.out.print("1");
5     }
6
7     public A() {
8         System.out.print("2");
9     }
10 }
11
12 class B extends A {
13
14     static {
15         System.out.print("a");
16     }
17
18     public B() {
19         System.out.print("b");
20     }
21 }
22
23 public class Hello {
24
25     public static void main(String[] args) {
26         A ab = new B();
27         ab = new B();
28     }
29 }
```

答：执行结果：1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

提示：如果不能给出此题的正确答案，说明之前第21题Java类加载机制还没有完全理解，赶紧再看看吧。

## 38、数据类型之间的转换

如何将字符串转换为基本数据类型？

答：调用基本数据类型对应的包装类中的方法`parseXXX(String)`或`valueOf(String)`即可返回相应基本数据类型。

如何将基本数据类型转换为字符串？

答：一种方法是将基本数据类型与空字符串（""）连接（+）即可获得其所对应的字符串；另一种方法是调用`String`类中的`valueOf()`方法返回相应字符串。

## 39、如何实现字符串的反转及替换？

答：方法很多，可以自己写实现也可以使用`String`或`StringBuffer`/`StringBuilder`中的方法。有一道很常见的面试题是用递归实现字符串反转，代码如下所示：

```
1     public static String reverse(String originStr) {
2         if (originStr == null || originStr.length() <= 1)
3             return originStr;
4         return reverse(originStr.substring(1)) + originStr.charAt(0);
5     }
```

## 40、怎样将GB2312编码的字符串转换为ISO-8859-1编码的字符串？

答：代码如下所示：

```
1 | String s1 = "你好";
2 | String s2 = new String(s1.getBytes("GB2312"), "ISO-8859-1");
```

## 41、日期和时间

- 如何取得年月日、小时分钟秒？
- 如何取得从1970年1月1日0时0分0秒到现在的毫秒数？
- 如何取得某月的最后一天？
- 如何格式化日期？

答：

问题1：创建java.util.Calendar实例，调用其get()方法传入不同的参数即可获得参数所对应的值。

Java 8中可以使用java.time.LocalDateTime来获取，代码如下所示。

```
1 | public class DateTimeTest {
2 |     public static void main(String[] args) {
3 |         Calendar cal = Calendar.getInstance();
4 |         System.out.println(cal.get(Calendar.YEAR));
5 |         System.out.println(cal.get(Calendar.MONTH)); // 0 - 11
6 |         System.out.println(cal.get(Calendar.DATE));
7 |         System.out.println(cal.get(Calendar.HOUR_OF_DAY));
8 |         System.out.println(cal.get(Calendar.MINUTE));
9 |         System.out.println(cal.get(Calendar.SECOND));
10 |
11 |         // Java 8
12 |         LocalDateTime dt = LocalDateTime.now();
13 |         System.out.println(dt.getYear());
14 |         System.out.println(dt.getMonthValue()); // 1 - 12
15 |         System.out.println(dt.getDayOfMonth());
16 |         System.out.println(dt.getHour());
17 |         System.out.println(dt.getMinute());
18 |         System.out.println(dt.getSecond());
19 |     }
20 | }
```

问题2：以下方法均可获得该毫秒数。

```
1 | Calendar.getInstance().getTimeInMillis();
2 | System.currentTimeMillis();
3 | Clock.systemDefaultZone().millis(); // Java 8
```

问题3：代码如下所示。

```
1 | Calendar time = Calendar.getInstance();
2 | time.getActualMaximum(Calendar.DAY_OF_MONTH);
```

问题4：利用java.text.DateFormat 的子类（如SimpleDateFormat类）中的format(Date)方法可将日期格式化。Java 8中可以用java.time.format.DateTimeFormatter来格式化时间日期，代码如下所示。

```
1 | import java.text.SimpleDateFormat;
2 | import java.time.LocalDate;
3 | import java.time.format.DateTimeFormatter;
```



```
4 | import java.util.Date; 5 |
6 | class DateFormatTest {
7 |
8 |     public static void main(String[] args) {
9 |         SimpleDateFormat oldFormatter = new SimpleDateFormat("yyyy/MM/dd");
10 |         Date date1 = new Date();
11 |         System.out.println(oldFormatter.format(date1));
12 |
13 |         // Java 8
14 |         DateTimeFormatter newFormatter = DateTimeFormatter.ofPattern("yyyy/MM/dd");
15 |         LocalDate date2 = LocalDate.now();
16 |         System.out.println(date2.format(newFormatter));
17 |     }
18 | }
```

补充：Java的时间日期API一直以来都是被诟病的東西，为了解决这一问题，Java 8中引入了新的时间日期API，其中包括LocalDate、LocalTime、LocalDateTime、Clock、Instant等类，这些的类的设计都使用了不变模式，因此是线程安全的设计。如果不理解这些内容，可以参考我的另一篇文章《关于Java并发编程的总结和思考》。

## 42、打印昨天的当前时刻。

```
1 | import java.util.Calendar;
2 |
3 | class YesterdayCurrent {
4 |     public static void main(String[] args){
5 |         Calendar cal = Calendar.getInstance();
6 |         cal.add(Calendar.DATE, -1);
7 |         System.out.println(cal.getTime());
8 |     }
9 | }
```

在Java 8中，可以用下面的代码实现相同的功能。

```
1 | import java.time.LocalDateTime;
2 |
3 | class YesterdayCurrent {
4 |
5 |     public static void main(String[] args) {
6 |         LocalDateTime today = LocalDateTime.now();
7 |         LocalDateTime yesterday = today.minusDays(1);
8 |
9 |         System.out.println(yesterday);
10 |     }
11 | }
```

## 43、比较一下Java和JavaScript。

答：JavaScript 与Java是两个公司开发的不同的两个产品。Java 是原Sun Microsystems公司推出的面向对象的程序设计语言，特别适合于互联网应用程序开发；而JavaScript是Netscape公司的产品，为了扩展Netscape浏览器的功能而开发的一种可以嵌入Web页面中运行的基于对象和事件驱动的解释性语言。JavaScript的前身是LiveScript；而Java的前身是Oak语言。

下面对两种语言间的异同作如下比较：

- 基于对象和面向对象：Java是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象；JavaScript是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object-Based）和事件驱动（Event-Driven）的编程语

言，因而它本身提供了非常丰富的内部对象供设计人员使用。

- 解释和编译：Java的源代码在执行之前，必须经过编译。JavaScript是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行。（目前的浏览器几乎都使用了JIT（即时编译）技术来提升JavaScript的运行效率）
- 强类型变量和类型弱变量：Java采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript中变量是弱类型的，甚至在使用变量前可以不作声明，JavaScript的解释器在运行时检查推断其数据类型。
- 代码格式不一样。

**补充：**上面列出的四点是网上流传的所谓的标准答案。其实Java和JavaScript最重要的区别是一个是静态语言，一个是动态语言。目前的编程语言的发展趋势是函数式语言和动态语言。在Java中类（class）是一等公民，而JavaScript中函数（function）是一等公民，因此JavaScript支持函数式编程，可以使用Lambda函数和闭包（closure），当然Java 8也开始支持函数式编程，提供了对Lambda表达式以及函数式接口的支持。对于这类问题，在面试的时候最好还是用自己的语言回答会更加靠谱，不要背网上所谓的标准答案。

## 44、什么时候用断言（assert）？

答：断言在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，断言用于保证程序最基本、键的正确性。断言检查通常在开发和测试时开启。为了保证程序的执行效率，在软件发布后断言检查通常是关闭的。断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为true；如果表达式的值为false，那么系统会报告一个AssertionError。断言的使用如下面的代码所示：

```
assert(a > 0); // throws an AssertionError if a <= 0
```

断言可以有两种形式：

```
assert Expression1;  
assert Expression1 : Expression2 ;
```

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的字符串消息。

要在运行时启用断言，可以在启动JVM时使用-enableassertions或者-ea标记。要在运行时选择禁用断言，可以在启动JVM时使用-da或者-disableassertions标记。要在系统类中启用或禁用断言，可使用-esa或-dsa标记。还可以在包的基础上启用或者禁用断言。

**注意：**断言不应该以任何方式改变程序的状态。简单的说，如果希望在不满足某些条件时阻止代码的执行，就可以考虑用断言来阻止它。

## 45、Error和Exception有什么区别？

答：Error表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；Exception表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

**面试题：**2005年摩托罗拉的面试中曾经问过这么一个问题“If a process reports a stack overflow run-time error, what's the most possible cause?”，给了四个选项a. lack of memory; b. write on an invalid memory space; c. recursive function calling; d. array index out of boundary. Java程序在运行时也可能会遭遇StackOverflowError，这是一个无法恢复的错误，只能重新修改代码了，这个面试题的答案是c。如果写了不能迅速收敛的递归，则很有可能引发栈溢出的错误，如下所示：

```
1 class StackOverflowErrorTest {  
2  
3     public static void main(String[] args) {  
4         main(null);  
5     }  
6 }
```

```
5 | } 6 | }
```

提示：用递归编写程序时一定要牢记两点：1. 递归公式；2. 收敛条件（什么时候就不再继续递归）。

#### 46、try{}里有一个return语句，那么紧跟在这个try后的finally{}里的代码会不会被执行，什么时候被执行，在return前还是后？

答：会执行，在方法返回调用者前执行。

注意：在finally中改变返回值的做法是不好的，因为如果存在finally代码块，try中的return语句不会立马返回调用者，而是记录下返回值待finally代码块执行完毕之后再向调用者返回其值，然后如果在finally中修改了返回值，就会返回修改后的值。显然，在finally中返回或者修改返回值会对程序造成很大的困扰，C#中直接用编译错误的方式来阻止程序员干这种龌龊的事情，Java中也可以通过提升编译器的语法检查级别来产生警告或错误，Eclipse中可以在如图所示的地方进行设置，强烈建议将此项设置为编译错误。

#### 47、Java语言如何进行异常处理，关键字：throws、throw、try、catch、finally分别如何使用？

答：Java通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在Java中，每个异常都是一个对象，它是Throwable类或其子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个方法可以捕获到这个异常并可以对其进行处理。Java的异常处理是通过5个关键词来实现的：try、catch、throw、throws和finally。一般情况下是用try来执行一段程序，如果系统会抛出（throw）一个异常对象，可以通过它的类型来捕获（catch）它，或通过总是执行代码块（finally）来处理；try用来指定一块预防所有异常的程序；catch子句紧跟在try块后面，用来指定你想要捕获的异常的类型；throw语句用来明确地抛出一个异常；throws用来声明一个方法可能抛出的各种异常（当然声明异常时允许无病呻吟）；finally为确保一段代码不管发生什么异常状况都要被执行；try语句可以嵌套，每当遇到一个try语句，异常的结构就会被放入异常栈中，直到所有的try语句都完成。如果下一级的try语句没有对某种异常进行处理，异常栈就会执行出栈操作，直到遇到有处理这种异常的try语句或者最终将异常抛给JVM。

#### 48、运行时异常与受检异常有何异同？

答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误，只要程序设计得没有问题通常就不会发生。受检异常跟程序运行的上下文环境有关，即使程序设计无误，仍然可能因使用的问题而引发。Java编译器要求方法必须声明抛出可能发生的受检异常，但是并不要求必须声明抛出未被捕获的运行时异常。异常和继承一样，是面向对象程序设计中经常被滥用的东西，在《Effective Java》中对异常的使用给出了以下指导原则：

- 不要将异常处理用于正常的控制流（设计良好的API不应该强迫它的调用者为了正常的控制流而使用异常）
- 对可以恢复的情况使用受检异常，对编程错误使用运行时异常
- 避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常的发生）
- 优先使用标准的异常
- 每个方法抛出的异常都要有文档
- 保持异常的原子性
- 不要在catch中忽略掉捕获到的异常

#### 49、列出一些你常见的运行时异常？

答：

- ArithmeticException（算术异常）
- ClassCastException（类转换异常）
- IllegalArgumentException（非法参数异常）
- IndexOutOfBoundsException（下标越界异常）
- NullPointerException（空指针异常）

- SecurityException （安全异常）

## 50、阐述final、finally、finalize的区别。

答：

- final：修饰符（关键字）有三种用法：如果一个类被声明为final，意味着它不能再派生出新的子类，即不能被继承，因此它和abstract是反义词。将变量声明为final，可以保证它们在使用中不被改变，被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取不可修改。被声明为final的方法也同样只能使用，不能在子类中被重写。
- finally：通常放在try...catch...的后面构造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要JVM不关闭都能执行，可以将释放外部资源的代码写在finally块中。
- finalize：Object类中定义的方法，Java中允许使用finalize()方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写finalize()方法可以整理系统资源或者执行其他清理工作。

我有一个微信公众号，经常会分享一些Java技术相关的干货；如果你喜欢我的分享，可以用微信搜索“Java团长”或者“javatuanzhang”关注。

## 下一篇：2019年最新Java面试题及答案整理（下）

👍 点赞 100    ☆ 收藏    📄 分享    ...



Java程序员-张凯

发布了242 篇原创文章 · 获赞 1492 · 访问量 60万+

他的留言板

关注

### 开发个小程序大概要多少钱



👤 张凯



想对作者说点什么



柒\_咚 9个月前 这都是笔试的，面试基本不会问这些东西了

查看回复(1)    🗨 3



危险游戏I 10个月前

🗨 1

JAVA8以前接口中不能有具体的方法，但是JAVA8开始，接口中可以有默认方法和静态方法。



开始做实施的小李同学 5个月前

👍

JDK7以后，常量池从方法区移出到堆里了

登录 查看 11 条热评

### java面试题大全（整理版）

阅读数 8万+

这几天在网上搜集各种java面试题：一是为了自己能复习方便，二是为了分享给大家~~题目都是来自网上大佬的分享，感谢...

博文 来自： [习惯很重要](#)

### 我遇到的Java面试题汇总（2019秋招篇）

阅读数 1万+

来自一名2019届应届毕业生总结的Java研发面试题汇总（2019秋招篇）2018年Java研发工程师面试题Java研发工程师面试...

博文 来自： [Mr\\_Van](#)

2019年最新Java面试题及答案整理（下）

阅读数 3万+

上一篇：2019年最新Java面试题及答案整理（上）51、类ExampleA继承Exception，类ExampleB继承ExampleA。有如下代... 博文 来自： Java笔记

2019年最新java面试题及答案，顺带面试技巧

阅读数 1064

2019年最新java面试题及答案java基础1.什么是Java虚拟机？为什么Java被称作是“平台无关的编程语言”？Java虚拟机是一... 博文



几款项目管理工具对比

项目管理的软件

广告

2019史上最全java面试题题库大全800题含答案

阅读数 4915

2019史上最全java面试题题库大全\_辟邪剑谱葵花宝典800题含答案1、 meta标签的作用是什么2、 ReentrantLock可重入锁... 博文 来自： weixin\_4...

从入门到精通，Java学习路线导航（附学习资源）

阅读数 7万+

引言最近也有很多人来向我“请教”，他们大都是一些刚入门的新手，还不了解这个行业，也不知道从何学起，开始的时候非... 博文 来自： java\_sha...

程序员必须掌握的核心算法有哪些？

阅读数 23万+

由于我之前一直强调数据结构以及算法学习的重要性，所以就有一些读者经常问我，数据结构与算法应该要学习到哪个程度... 博文 来自： 帅地

208道最常见的Java面试题整理（面试必备）

阅读数 1908

适宜阅读人群需要面试的初/中/高级 java 程序员想要查漏补缺的人想要不断完善和扩充自己 java 技术栈的人java 面试官具体... 博文 来自： weixin\_3...

c++制作的植物大战僵尸，开源，一代二代结合游戏

阅读数 1万+

此游戏全部由本人自己制作完成。游戏大部分的素材来源于原版游戏素材，少部分搜集于网络，以及自己制作。此游戏为同... 博文 来自： 尔灵尔亿...

现在这个年代，面试官竟然还要求我会C++？！

广告 关闭

今天去面试，面试官突然问我：会C++吗？掌握的怎么样？我心里：这都什么年代了？

字节跳动视频编解码面经

阅读数 7万+

三四月份投了字节跳动的实习（图形图像岗位），然后hr打电话过来问了一下会不会opengl，c++，shadon，当时只会一点c... 博文 来自： lj\_h\_shuai...

2018年最新Java面试题及答案整理

阅读数 6万+


下列面试题都是在网上收集的，本人抱着学习的态度找了下参考答案，有不足的地方还请指正，更多精彩内容可以关注我的... 博文 来自： Java笔记



习惯~

14篇文章

关注 排名:千里之外



private\_static

39篇文章

关注 排名:千里之外



优雅de程序员

16篇文章

关注 排名:千里之外

130 个相见恨晚的超实用网站，一次性分享出来

阅读数 9万+

相见恨晚的超实用网站持续更新中。。。 博文 来自： 藏冰的博客

python 简易微信实现（注册登录+数据库存储+聊天+GUI+文件传输）

阅读数 5697

socket+tkinter详解+简易微信实现历经多天的努力，查阅了许多大佬的博客后终于实现了一个简易的微信O(∩\_∩)O~~简易数... 博文 来自： weixin\_4...

2017JAVA面试题附答案

阅读数 17万+

声明，本人能力有限，只是列出来参考，不对之处欢迎指正。JAVA基础JAVA中的几种基本类型，各占用多少字节？下图单... 博文 来自： 前往JAVA...

热议：程序员要不要学数学？我被结果整懵了

广告 学院 关闭

我经常在后台收到各种各样的问题，有一个问题一直被问起：程序员要不要学数学？

<div><div>Java学习的正确打开方式</div><div>在博主认为，对于入门级学习java的最佳学习方法莫过于视频+博客+书籍+总结，前三者博主将淋漓尽致地挥毫于这篇博客文...</div></div>	阅读数 14万+	博文	来自： <a href="#">程序员宜...</a>
<div><div>终于明白阿里百度这样的大公司，为什么面试经常拿ThreadLocal考验求职者了</div><div>点击上方↑「爱开发」关注我们每晚10点，捕获技术思考和创业资源洞察什么是ThreadLocalThreadLocal是一个本地线程副...</div></div>	阅读数 10万+	博文	来自： <a href="#">爱开发</a>
<div><div>爬虫小程序 - 爬取王者荣耀全皮肤</div><div>王者荣耀全皮肤图片爬取</div></div>	阅读数 12万+	博文	来自： <a href="#">君莫笑</a>
<div><div>2018、2019年java技术面试题整理</div><div>1、servlet执行流程客户端发出http请求，web服务器将请求转发到servlet容器，servlet容器解析url并根据web.xml找到相对...</div></div>	阅读数 5万+	博文	来自： <a href="#">猿立方</a>
<div><div>【吐血整理】年度盘点   2019年Java面试题汇总篇——附答案</div><div>在这岁月更替辞旧迎新的时刻，老王盘点了一下自己 2019 年发布的所有文章，意外的发现关于「Java面试」的主题文章，...</div></div>	阅读数 970	博文	来自： <a href="#">王磊的博客</a>
<div><div>反转！“只问了1个框架，就给了35K的Python岗”</div><div>学Python的程序员建议收藏！</div></div>	<div>学院广告</div> <div>关闭</div>		
<div><div>第七届蓝桥杯A组C</div><div>1.网友年龄某君新认识一网友。当问及年龄时，他的网友说：我的年龄是个2位数，我比儿子大27岁,如果把我的年龄的两位...</div></div>	阅读数 1万+	博文	来自： <a href="#">duchenlo...</a>
<div><div>网页实现一个简单的音乐播放器（大佬别看。(๑___๑)）</div><div>今天闲着无事，就想写点东西。然后听了下歌，就打算写个播放器。于是乎用h5 audio的加上js简单的播放器完工了。演示地...</div></div>	阅读数 5万+	博文	来自： <a href="#">qq_4421...</a>
<div><div>2019JAVA面试题附答案(长期更新)</div><div>https://blog.csdn.net/weixin_38399962/article/details/80358168</div></div>	阅读数 1427	博文	来自： <a href="#">后知后觉...</a>
<div><div>2019年10月中国编程语言排行榜</div><div>2019年10月2日，我统计了某招聘网站，获得有效程序员招聘数据9万条。针对招聘信息，提取编程语言关键字，并统计如下...</div></div>	阅读数 1万+	博文	来自： <a href="#">毛毛虫</a>
<div><div>比特币原理详解</div><div>一、什么是比特币比特币是一种电子货币，是一种基于密码学的货币，在2008年11月1日由中本聪发表比特币白皮书，文中...</div></div>	阅读数 15万+	博文	来自： <a href="#">zcg_7414...</a>
<div><div>数据库优化 - SQL优化</div><div>以实际SQL入手，带你一步一步走上SQL优化之路！</div></div>	阅读数 14万+	博文	来自： <a href="#">飘渺Jam...</a>
<div><div>通俗易懂地给女朋友讲：线程池的内部原理</div><div>餐盘在灯光的照耀下格外晶莹剔透，女朋友拿起红酒杯轻轻地抿了一小口，对我说：“经常听你说线程池，到底线程池到底是...</div></div>	阅读数 8万+	博文	来自： <a href="#">万猫学社</a>
<div><div>经典算法（5）杨辉三角</div><div>杨辉三角 是经典算法，这篇博客对它的算法思想进行了讲解，并有完整的代码实现。...</div></div>	阅读数 7万+	博文	来自： <a href="#">扬帆向海...</a>
<div><div>2019最新整理JAVA面试题附答案</div><div>点击上方“全球Java架构师集中营”，选择“设为星标”技术文章第一时间送达！包含的模块：本文分为十九个模块，分别是：Ja...</div></div>	阅读数 111	博文	来自： <a href="#">Java架构...</a>
<div><div>有哪些让程序员受益终生的建议</div><div>从业五年多，辗转两个大厂，出过书，创过业，从技术小白成长为基层管理，联合几个业内大牛回答下这个问题，希望能帮...</div></div>	阅读数 13万+	博文	来自： <a href="#">启舰</a>



<div><div>这30个CSS选择器，你必须熟记（上）</div><div>关注前端达人，与你共同进步CSS的魅力就是让我们前端工程师像设计师一样进行网页的设计，我们能轻而易举的改变颜色...</div></div>	<div>博文</div> <div>来自： <a href="#">前端达人</a></div>	<div>阅读数 2万+</div>
<div><div>英特尔不为人知的 B 面</div><div>从 PC 时代至今，众人只知在 CPU、GPU、XPU、制程、工艺等战场中，英特尔在与同行硬件芯片制造商们的竞争中杀出重...</div></div>	<div>博文</div> <div>来自： <a href="#">CSDN资讯</a></div>	<div>阅读数 3万+</div>
<div><div>redis分布式锁，面试官请随便问，我都会</div><div>文章有点长并且绕，先来个图片缓冲下！前言现在的业务场景越来越复杂，使用的架构也就越来越复杂，分布式、高并发已...</div></div>	<div>博文</div> <div>来自： <a href="#">公众号-[...]</a></div>	<div>阅读数 1万+</div>
<div><div>14位享誉全球的程序员</div><div>本文转载至： <a href="http://www.cricode.com/2922.html">http://www.cricode.com/2922.html</a></div></div>	<div>博文</div> <div>来自： <a href="#">闲云孤鹤</a></div>	<div>阅读数 3万+</div>
<div><div>Spring Boot 2.0(四)：使用 Docker 部署 Spring Boot</div><div>Docker 技术发展为微服务落地提供了更加便利的环境，使用 Docker 部署 Spring Boot 其实非常简单，这篇文章我们就来简...</div></div>	<div>博文</div> <div>来自： <a href="#">weixin_3...</a></div>	<div>阅读数 988</div>
<div><div>源码阅读(19)：Java中主要的Map结构——HashMap容器（下1）</div><div>HashMap容器从字面的理解就是，基于Hash算法构造的Map容器。从数据结构的知识体系来说，HashMap容器是散列表在J...</div></div>	<div>博文</div> <div>来自： <a href="#">JAVA入门中</a></div>	<div>阅读数 5335</div>
<div><div>《吊打面试官》系列-秒杀系统设计</div><div>你知道的越多，你不知道的越多 点赞再看，养成习惯 GitHub上已经开源 <a href="https://github.com/JavaFamily">https://github.com/JavaFamily</a> 有一线大厂面试点脑...</div></div>	<div>博文</div> <div>来自： <a href="#">敖丙</a></div>	<div>阅读数 1万+</div>
<div><div>面试专题</div><div>Java 最常见 200+ 面试题全解析：面试必备序言在本篇文章开始之前，我想先来回答一个问题：我为什么要写这样一篇关于...</div></div>	<div>博文</div> <div>来自： <a href="#">庸人自扰...</a></div>	<div>阅读数 287</div>
<div><div>dp</div><div>2.1斐波那契系列问题2.2矩阵系列问题2.3跳跃系列问题3.1 01背包3.2 完全背包3.3多重背包3.4 一些变形选讲2.1斐波那契系...</div></div>	<div>博文</div>	<div>阅读数 1万+</div>
<div><div>我花了一夜用数据结构给女朋友写个H5走迷宫游戏</div><div>起因 又到深夜了，我按照以往在csdn和公众号写着数据结构！这占用了我大量的时间！我的超越妹妹严重缺乏陪伴而 怨气...</div></div>	<div>博文</div>	<div>阅读数 33万+</div>
<div><div>酸奶什么时候喝好?一次该</div><div>一天什么时候喝酸奶好</div></div>	<div></div>	
<div><div>Python——画一棵漂亮的樱花树（不同种樱花+玫瑰+圣诞树喔）</div><div>最近翻到一篇知乎，上面有不少用Python（大多是turtle库）绘制的树图，感觉很漂亮，我整理了一下，挑了一些我觉得不错...</div></div>	<div>博文</div>	<div>阅读数 16万+</div>
<div><div>大学四年自学走来，这些私藏的实用工具/学习网站我贡献出来了</div><div>大学四年，看课本是不可能一直看课本的了，对于学习，特别是自学，善于搜索网上的一些资源来辅助，还是非常有必要的...</div></div>	<div>博文</div>	<div>阅读数 27万+</div>
<div><div>Python十大装B语法</div><div>Python 是一种代表简单思想的语言，其语法相对简单，很容易上手。不过，如果就此小视 Python 语法的精妙和深邃，那就...</div></div>	<div>博文</div>	<div>阅读数 25万+</div>
<div><div>《奇巧淫技》系列-python！！每天早上八点自动发送天气预报邮件到QQ邮箱</div><div>将代码部署服务器，每日早上定时获取到天气数据，并发送到邮箱。也可以说是一个小型人工智障。知识可以运用在不同地...</div></div>	<div>博文</div>	<div>阅读数 3万+</div>
<div><div>Python实例大全（基于Python3.7.4）</div><div>博客说明： 这是自己写的有关python语言的一篇综合博客。 只作为知识广度和编程技巧学习，不过于追究学习深度，点到即...</div></div>	<div>博文</div>	<div>阅读数 1万+</div>
<div><div>软件测试面试笔试题汇总</div><div>软件测试</div></div>	<div></div>	

<div>腾讯算法面试题：64匹马8个跑道需要多少轮才能选出最快的四匹？</div> <div>昨天，有网友私信我，说去阿里面试，彻底的被打击到了。问了为什么网上大量使用ThreadLocal的源码都会加上private stat...</div>	<div>阅读数 6万+</div> <div>博文</div>
<div>面试官：你连RESTful都不知道我怎么敢要你？</div> <div>干货，2019 RESTful最贱实践</div>	<div>阅读数 10万+</div> <div>博文</div>
<div>机械转行java自学经历，零基础学java，血泪总结的干货</div> <div>机械转行java自学经历，零基础学java，血泪总结的干货 据说，再恩爱的夫妻，一生中都有100次想离婚的念头和50次想掐...</div>	<div>阅读数 1万+</div> <div>博文</div>
<div>刷了几千道算法题，这些我私藏的刷题网站都在这里了！</div> <div>遥想当年，机缘巧合入了 ACM 的坑，周边巨擘林立，从此过上了"天天被虐似死狗"的生活... 然而我是谁，我可是死狗中的...</div>	<div>阅读数 8万+</div> <div>博文</div>
<div>项目中的if else太多了，该怎么重构？</div> <div>介绍 最近跟着公司的大佬开发了一款IM系统，类似QQ和微信哈，就是聊天软件。我们有一部分业务逻辑是这样的 if (msgTy...</div>	<div>阅读数 13万+</div> <div>博文</div>
<div> 硕士毕业论文两天写完</div> <div>硕士论文</div>	<div>阅读数 1万+</div> <div>博文</div>
<div>致 Python 初学者</div> <div>欢迎来到"Python进阶"专栏！来到这里的每一位同学，应该大致上学习了很多 Python 的基础知识，正在努力成长的过程中。...</div>	<div>阅读数 18万+</div> <div>博文</div>
<div>YouTube排名第一的励志英文演讲《Dream(梦想)》</div> <div>Idon't know what that dream is that you have, I don't care how disappointing it might have been as y...</div>	<div>阅读数 4万+</div> <div>博文</div>
<div>“狗屁不通文章生成器”登顶GitHub热榜，分分钟写出万字形式主义大作</div> <div>一、垃圾文字生成器介绍 最近在浏览GitHub的时候，发现了这样一个骨骼清奇的雷人项目，而且热度还特别高。项目中文...</div>	<div>阅读数 14万+</div> <div>博文</div>
<div>程序员：我终于知道post和get的区别</div> <div>是一个老生常谈的话题，然而随着不断的学习，对于以前的认识有很多误区，所以还是需要不断地总结的，学而时习之，不...</div>	<div>阅读数 22万+</div> <div>博文</div>
<div>《程序人生》系列-这个程序员只用了20行代码就拿了冠军</div> <div>你知道的越多，你不知道的越多 点赞再看，养成习惯GitHub上已经开源https://github.com/JavaFamily，有一线大厂面试点脑...</div>	<div>阅读数 6万+</div> <div>博文</div>
<div>加快推动区块链技术和产业创新发展，2019可信区块链峰会在京召开</div> <div>11月8日，由中国信息通信研究院、中国通信标准化协会、中国互联网协会、可信区块链推进计划联合主办，科技行者协办的...</div>	<div>阅读数 6万+</div> <div>博文</div>
<div>Python3.7黑帽编程——病毒篇（基础篇）</div> <div>引子 Hacker（黑客），往往被人们理解为只会用非法手段来破坏网络安全的计算机高手。但是，黑客其实不是这样的，真正...</div>	<div>阅读数 1万+</div> <div>博文</div>
<div>程序员把地府后台管理系统做出来了，还有3.0版本！12月7号最新消息：已在开发中有github地址</div> <div>第一幕：缘起 听说阎王爷要做个生死簿后台管理系统，我们派去了一个程序员..... 996程序员做的梦： 第一场：团队招募 ...</div>	<div>阅读数 17万+</div> <div>博文</div>
<div>网易云6亿用户音乐推荐算法</div> <div>网易云音乐是音乐爱好者的集聚地，云音乐推荐系统致力于通过 AI 算法的落地，实现用户千人千面的个性化推荐，为用户带...</div>	<div>阅读数 5万+</div> <div>博文</div>
<div>【技巧总结】位运算装逼指南</div> <div>位算法的效率有多快我就不说，不信你可以去用 10 亿个数据模拟一下，今天给大家讲一讲位运算的一些经典例子。不过， ...</div>	<div>阅读数 2万+</div> <div>博文</div>
<div>大学生生活这样过，校招 offer 飞来找</div> <div>本篇我们来聊聊大学生生活如何度过，才能在校招中拿到 offer。</div>	<div>阅读数 1万+</div> <div>博文</div>

<div>为什么要学数据结构？</div> <div>一、前言 在可视化化程序设计的今天，借助于集成开发环境可以很快地生成程序，程序设计不再是计算机专业人员的专利。...</div>	<div>阅读数 2万+</div> <div>博文</div>
<div>8年经验面试官详解 Java 面试秘诀</div> <div>作者  胡书敏 责编   刘静 出品   CSDN（ID：CSDNnews） 本人目前在一家知名外企担任架构师，而且最近八年来，在多家...</div>	<div>阅读数 10万+</div> <div>博文</div>
<div>面试官如何考察你的思维方式？</div> <div>1.两种思维方式在求职面试中，经常会考察这种问题：北京有多少量特斯拉汽车？某胡同口的煎饼摊一年能卖出多少个煎饼...</div>	<div>阅读数 5万+</div> <div>博文</div>
<div>碎片化的时代，如何学习</div> <div>今天周末，和大家聊聊学习这件事情。在如今这个社会，我们的时间被各类 APP 撕的粉碎。刷知乎、刷微博、刷朋友圈； ...</div>	<div>阅读数 2万+</div> <div>博文</div>
<div>27 个提升开发幸福度的 VsCode 插件</div> <div>作者：Jsmanifest 译者：前端小智 来源：Medium Visual Studio Code（也称为VSCode）是一种轻量级但功能强大的跨平台...</div>	<div>阅读数 2万+</div> <div>博文</div>
<div>so easy！10行代码写个"狗屁不通"文章生成器</div> <div>前几天，GitHub 有个开源项目特别火，只要输入标题就可以生成一篇长长的文章。背后实现代码一定很复杂吧，里面一定...</div>	<div>阅读数 9万+</div> <div>博文</div>
<div>知乎高赞：中国有什么拿得出手的开源软件产品？(整理自本人原创回答)</div> <div>知乎高赞：中国有什么拿得出手的开源软件产品？ 在知乎上，有个问题问“中国有什么拿得出手的开源软件产品（在 GitHub ...</div>	<div>阅读数 6万+</div> <div>博文</div>
<div>MySQL数据库总结</div> <div>一、数据库简介 数据库(Database，DB)是按照数据结构来组织，存储和管理数据的仓库。典型特征：数据的结构化、数据...</div>	<div>阅读数 7万+</div> <div>博文</div>
<div>记一次腾讯面试：进程之间究竟有哪些通信方式？如何通信？ ---- 告别死记硬背</div> <div>有一次面试的时候，被问到进程之间有哪些通信方式，不过由于之前没深入思考且整理过，说的并不好。想必大家也都知道...</div>	<div>阅读数 4万+</div> <div>博文</div>
<div>20行Python代码爬取王者荣耀全英雄皮肤</div> <div>引言 王者荣耀大家都玩过吧，没玩过的也应该听说过，作为时下最火的手机MOBA游戏，咳咳，好像跑题了。我们今天的重...</div>	<div>阅读数 13万+</div> <div>博文</div>
<div>腾讯架构师，为了家庭去小厂，一个月后主动离职：不做中台就是等死</div> <div>今天咱们第一课，来讲讲大家一直很关注的数据中台。其实，数据中台也是企业数据管理的一部分，甚至可以说是很重要的...</div>	<div>阅读数 9744</div> <div>博文</div>
<div>张小龙-年薪近3亿的微信之父，他是如何做到的？</div> <div>张小龙生于湖南邵东魏家桥镇， 家庭主要特点：穷。不仅自己穷，亲戚也都很穷，可以说穷以类聚。爷爷做过铜匠，总的来...</div>	<div>阅读数 11万+</div> <div>博文</div>
<div>西游记团队中如果需要裁掉一个人，会先裁掉谁？</div> <div>2019年互联网寒冬，大批企业开始裁员，下图是网上流传的一张截图： 裁员不可避免，那如何才能做到不管大环境如何变化...</div>	<div>阅读数 6万+</div> <div>博文</div>
<div>iOS Bug 太多，苹果终于坐不住了！</div> <div>开源的 Android 和闭源的 iOS，作为用户的你，更偏向哪一个呢？ 整理   屠敏 出品   CSDN（ID：CSDNnews） 毋庸置疑，...</div>	<div>阅读数 6万+</div> <div>博文</div>
<div>聊聊C语言和指针的本质</div> <div>坐着绿皮车上海到杭州，24块钱，很宽敞，在火车上非正式地聊几句。很多编程语言都以“没有指针”作为自己的优势来宣传...</div>	<div>阅读数 1万+</div> <div>博文</div>
<div>究竟你适不适合买Mac？</div> <div>我清晰的记得，刚买的macbook pro回到家，开机后第一件事情，就是上了淘宝网，花了500元钱，找了一个上门维修电脑的...</div>	<div>阅读数 3万+</div> <div>博文</div>
<div>程序员一般通过什么途径接私活？</div> <div>二哥，你好，我想知道一般程序猿都如何接私活，我也想接，能告诉我一些方法吗？ 上面是一个读者“烦不烦”问我的一个问...</div>	<div>阅读数 10万+</div> <div>博文</div>

（经验分享）作为一名普通本科计算机专业学生，我大学四年到底走了多少弯路

阅读数 6万+

今年正式步入了大四，离毕业也只剩半年多的时间，回想一下大学四年，感觉自己走了不少弯路，今天就来分享一下自己大...

博文

python json java mysql pycharm android linux json格式 c#中dns类 c#合并的excel c# implicit c#怎么保留3个小数点 c# 串口通信、 网络调试助手c# c# 泛型比较大小 c#解压分卷问题 c#启动居中 c# 逻辑或运算符