* 1.多线程，并发及线程基础
* 2.数据类型转换的基本原则
* 3.垃圾回收（GC）
* 4.Java 集合框架
* 5.数组
* 6.字符串
* 7.GOF 设计模式
* 8.SOLID （单一功能、开闭原则、里氏替换、接口隔离以及依赖反转）设计原则
* 9.抽象类与接口
* 10.Java 基础，如 equals 和 hashcode
* 11.泛型与枚举
* 12.Java IO 与 NIO
* 13常用网络协议
* 14.Java 中的数据结构和算法
* 15.正则表达式
* 16.JVM 底层
* 17.Java 最佳实践
* 18.JDBC
* 19.Date, Time 与 Calendar
* 20.Java 处理 XML
* 21.JUnit
* 22.编程
* 23.Java常见的集合
* 24.数据库相关
* 25.MVC框架
* 26.Linux命令

**多线程，并发及线程基础：**

在java5.0中提供了java.util.concurrent（JUC）包，增加了在并发编程中常见的工具类。

1. 如何区别多线程和单线程？

在单核CPU中，每次只执行一个线程，将CPU分为很小的时间片，是一种轮流占用CPU的机制。

多线程存在线程中上下文切换，导致程序的执行速度变慢。

采用多线程不会提供程序的执行速度，但是可以减少用户的响应速度。

1. 如何指定多个线程的执行顺序？

设定flag和id，判断flag是否和ID相等，如果不相等则调用thread.sleep()，方法，如果相等则调用synchronize方法，使得flag++，然后创建数组，遍历数组，把数组中的每个位置创建thread对象，然后调用start()方法执行。

1. 线程和进程的区别?

进程是“执行中的程序”，进行资源分配和调度的独立单位；线程是进程的实体，一个进程可以拥有多个线程，线程的上下文切换比进程要快，线程之间共享地址空间和其他资源。

1. 多线程产生死锁的必要条件？
2. 互斥条件：一个资源每次只能被一个线程使用；
3. 请求与保持条件：一个线程因请求资源而阻塞时，对已获得的资源保持不放；
4. 不剥夺条件：进程已经获得的资源，在未使用完之前，不能强行剥夺；
5. 循环等待条件：若干线程之间形成一种头尾相接的循环等待资源关系。

5.如何避免死锁？

按顺序获取锁就可以避免，避免循环等待条件。

6.sleep( ) 和 wait( n)、wait( ) 的区别：

答：

sleep 方法：是 Thread 类的静态方法，当前线程将睡眠 n 毫秒，线程进入阻塞状态。当睡眠时间到了，会解除阻塞，进行可运行状态，等待 CPU 的到来。睡眠不释放锁（如果有的话）；

wait 方法：是 Object 的方法，必须与 synchronized 关键字一起使用，线程进入阻塞状态，当 notify 或者 notifyall 被调用后，会解除阻塞。但是，只有重新占用互斥锁之后才会进入可运行状态。睡眠时，释放互斥锁。

1. synchronize关键字？

底层实现：

1. 进入时，执行monitorenter，将计数器+1，释放锁monitorexit时，计数器-1；
2. 当一个线程判断计数器为0的时候，则当前锁空闲，可以占用，反之，当前线程进入等待状态。
3. volatile关键字：

可以保证可见性和顺序性，不保证原子性，该关键字修饰的变量为共享变量，不会进行重新排序，是比synchronize更轻量级的同步。

1. ThreadLocal（线程局部变量）关键字？

ThreadLocal为每个使用改变量的线程提供变量副本。

内部源码：

里面会维护一个ThreadLocalMap对象，里面会有若干个Entry，其中key指的ThreadLocal实例，value指的是线程特有的对象。Entry 对 Key 的引用是弱引用；Entry 对 Value 的引用是强引用。

1. 线程池？

java.util.concurrent.ThreadPoolExecutor 类就是一个线程池。客户端调用 ThreadPoolExecutor.submit(Runnable task) 提交任务，线程池内部维护的工作者线程的数量就是该线程池的线程池大小，有 3 种形态：

* 当前线程池大小 ：表示线程池中实际工作者线程的数量；
* 最大线程池大小 （maxinumPoolSize）：表示线程池中允许存在的工作者线程的数量上限；
* 核心线程大小 （corePoolSize ）：表示一个不大于最大线程池大小的工作者线程数量上限。

1. 如果运行的线程少于 corePoolSize，则 Executor 始终首选添加新的线程，而不进行排队；
2. 如果运行的线程等于或者多于 corePoolSize，则 Executor 始终首选将请求加入队列，而不是添加新线程；
3. 如果无法将请求加入队列，即队列已经满了，则创建新的线程，除非创建此线程超出 maxinumPoolSize， 在这种情况下，任务将被拒绝。

内存：

1. “a==b”和”a.equals(b)”区别？
2. ==在比较基本类型时，比较的是值，在比较引用类型时，比较两个对象的地址值，equals方法存于Object类中，如果不重写方法，那么与==原理相同，绝大多数类都重写了equals方法，重写后的equals方法比较的是两个对象的值。

如何编写equals方法？

重写equals方法的要求：1.自反性2.对称性3.传递性4.一致性5.非空性

Public boolean equals(Object obj){

If(obj instanceof Person){

Person person=(Person)obj;

Return name.equalsIgnoreCase(person.getName().trim());

}

Return false;

}

1. Final、finalize、finally的不同之处？

Final是一个修饰符可以修饰变量方法和类，被final修饰后意味着值在初始化后不能不改变，不能改变对象的引用，但可以使用set方法。

GC垃圾回收机制，将调用finalize（）方法将对象从内存中清除之前进行的清理工作。

Finally是一个关键字，与try catch一起进行异常处理，finally代码块一定会被执行到。

1. java中编译器常量是什么？有什么风险？

公共静态变量（public static final ）就是编译期常量。

1. StringBuffer和StringBuilder区别？StringBuffer是如何扩容的？

StringBuffer是线程安全的，支持并发操作，支持多线程；

StringBuilder是线程不安全的，不支持并发操作，支持单线程。

StringBuffer继承了抽象类AbstractStringBuilder,StringBuffer可以给指定对象的实体初始化容量字符串长度再另外加16个字符，super(str.length+16),这是自动扩容，还有就是扩容算法：使用append（）方法在字符串后面追加东西的时候，如果超过了字符串存储空间大小就需要进行扩容，传入一个方法：

ensureCapacityInternal进行是否扩容判断，需要扩容就调用expandCapacity方法进行扩容。

集合：

**1.poll() 方法和 remove() 方法的区别？**

poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

**2.J**ava 中 LinkedHashMap 和 PriorityQueue 的区别是什么？(答案)

PriorityQueue 保证最高或者最低优先级的的元素总是在队列头部，但是 LinkedHashMap 维持的顺序是元素插入的顺序。当遍历一个 PriorityQueue 时，没有任何顺序保证，但是 LinkedHashMap 课保证遍历顺序是元素插入的顺序。

3.**）ArrayList 与 LinkedList 的不区别？(答案)**

最明显的区别是 ArrrayList 底层的数据结构是数组，支持随机访问，而 LinkedList 的底层数据结构书链表，不支持随机访问。使用下标访问一个元素，ArrayList 的时间复杂度是 O(1)，而 LinkedList 是 O(n)

4.**用哪两种方式来实现集合的排序？(答案)**

你可以使用有序集合，如 TreeSet 或 TreeMap，你也可以使用有顺序的的集合，如 list，然后通过 Collections.sort() 来排序。

5.**Java 中怎么打印数组？(answer答案)**

你可以使用 Arrays.toString() 和 Arrays.deepToString() 方法来打印数组。由于数组没有实现 toString() 方法，所以如果将数组传递给 System.out.println() 方法，将无法打印出数组的内容，但是 Arrays.toString() 可以打印每个元素。

6.**Java 中的 LinkedList 是单向链表还是双向链表？(答案)**

是双向链表

7.**Java 中的 TreeMap 是采用什么树实现的？(答案)**

Java 中的 TreeMap 是使用红黑树实现的。

**8.Hashtable 与 HashMap 有什么不同之处？(答案)**

这两个类有许多不同的地方，下面列出了一部分： a) Hashtable 是 JDK 1 遗留下来的类，而 HashMap 是后来增加的。 b）Hashtable 是同步的，比较慢，但 HashMap 没有同步策略，所以会更快。 c）Hashtable 不允许有个空的 key，但是 HashMap 允许出现一个 null key。 更多的不同之处参见答案。

9.**Java 中的 HashSet，内部是如何工作的？(answer答案)**

HashSet 的内部采用 HashMap来实现。由于 Map 需要 key 和 value，所以所有 key 的都有一个默认 value。类似于 HashMap，HashSet 不允许重复的 key，只允许有一个null key，意思就是 HashSet 中只允许存储一个 null 对象。

10.**ArrayList 和 HashMap 的默认大小是多数？(答案)**

在 Java 7 中，ArrayList 的默认大小是 10 个元素，HashMap 的默认大小是16个元素（必须是2的幂）。

11.**有没有可能两个不相等的对象有有相同的 hashcode？**

有可能，两个不相等的对象可能会有相同的 hashcode 值，这就是为什么在 hashmap 中会有冲突。相等 hashcode 值的规定只是说如果两个对象相等，必须有相同的hashcode 值，但是没有关于不相等对象的任何规定。

12.**）Java 中，Comparator 与 Comparable 有什么不同？(答案)**

Comparable 接口用于定义对象的自然顺序，而 comparator 通常用于定义用户定制的顺序。Comparable 总是只有一个，但是可以有多个 comparator 来定义对象的顺序。

13.**为什么在重写 equals 方法的时候需要重写 hashCode 方法？(答案)**

因为有强制的规范指定需要同时重写 hashcode 与 equal 是方法，许多容器类，如 HashMap、HashSet 都依赖于 hashcode 与 equals 的规定。

### **Java IO 和 NIO 的面试题**

1. **Java 中怎么创建 ByteBuffer？**

**1.1**

byte[] bytes = new byte[10];

ByteBuffer buf = ByteBuffer.wrap(bytes);

1.2

使用allocate()静态方法  
    ByteBuffer buffer=ByteBuffer.allocate(256);  
    以上方法将创建一个容量为256字节的ByteBuffer,如果发现创建的缓冲区容量太小,唯一的选择就是重新创建一个大小合适的缓冲区.

1. 回绕缓冲区  
     buffer.flip();  
     这个方法用来将缓冲区准备为数据传出状态,执行以上方法后,输出通道会从数据的开头而不是末尾开始.回绕保持缓冲区中的数据不变,只是准备写入而不是读取.  
     
   3.清除缓冲区  
     buffer.clear();  
     这个方法实际上也不会改变缓冲区的数据,而只是简单的重置了缓冲区的主要索引值.不必为了每次读写都创建新的缓冲区,那样做会降低性能.相反,要重用现在的缓冲区,在再次读取之前要清除缓冲区.  
     
   4.从套接字通道(信道)读取数据  
     int bytesReaded=socketChannel.read(buffer);  
     执行以上方法后,通道会从socket读取的数据填充此缓冲区,它返回成功读取并存储在缓冲区的字节数.在默认情况下,这至少会读取一个字节,或者返回-1指示数据结束.  
     
   5.向套接字通道(信道)写入数据  
     socketChannel.write(buffer);  
     此方法以一个ByteBuffer为参数,试图将该缓冲区中剩余的字节写入信道.
2. java中的大端和小端
3. Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。  
    2) Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

### **4.ByteBuffer 中的字节序是什么？**

### **java中转换字节序**

ByteBuffer类中的order(ByteOrder bo) 方法可以设置 ByteBuffer 的字节序。

**5.Java 中，直接缓冲区与非直接缓冲器有什么区别？(答案)？**

非直接缓冲区：通过allocate方法分配缓冲区，将缓冲区建立在JVM的内存中。

直接缓冲区：通过allocateDirect方法分配直接缓冲区，将缓冲区建立在物理内存中，可以提供效率。

6.**Java 中的内存映射缓存区是什么？(answer答案)**

MappedByteBuffer

1. **socket 选项 TCP NO DELAY 是指什么？**

Socket编程中，TCP\_NODELAY选项是用来控制是否开启Nagle算法，该算法是为了提高较慢的广域网传输效率，减小小分组的报文个数，完整描述：

该算法要求一个TCP连接上最多只能有一个未被确认的小分组，在该小分组的确认到来之前，不能发送其他小分组。

1. **TCP 协议与 UDP 协议有什么区别？**
2. 基于连接与无连接；  
   2.对系统资源的要求（TCP较多，UDP少）；  
   3.UDP程序结构较简单；  
   4.流模式与数据报模式 ；  
   5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

**单元测试：**

1. **@Before 和 @BeforeClass 有什么区别？**

@before在每次执行方法之前都执行一次，需声明public

@beforeClass只在类中执行一次，需要声明public static

**字符串编程题：**

**正则表达式：**

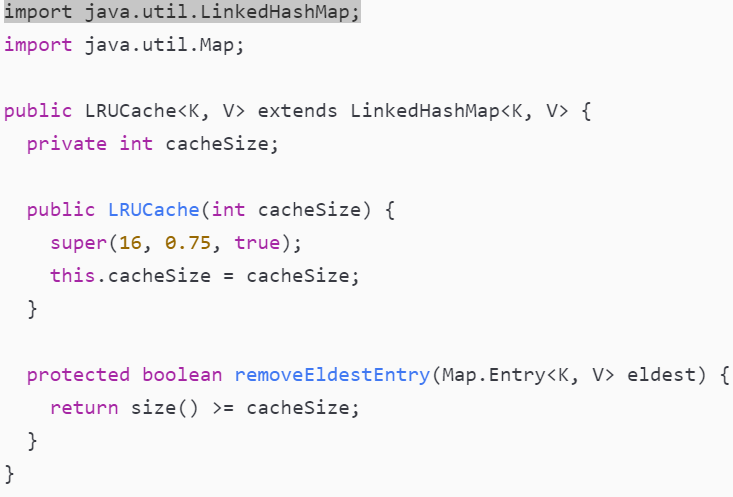
　字符串；tel:086-0666-88810009999  
　　原始正则："^tel:[0-9]{1,3}-[0][0-9]{2,3}-[0-9]{8,11}$"   
　　速记理解：开始 "tel:普通文本"[0-9数字]{1至3位}"-普通文本"[0数字][0-9数字]{2至3位}"-普通文本"[0-9数字]{8至11位} 结束"

**1.怎么检查一个字符串只包含数字？**

String regex="[0,9]+";  
String data="3457658654";  
System.*out*.println(data.matches(regex));

2.String regex = "^\\d+$";  
String data = "2233";  
System.*out*.println(data.matches(regex));

2.**Java 中如何利用泛型写一个 LRU 缓存？**



**3写一段 Java 程序将 byte 转换为 long？(答案)**

**public** **class** Test {

**private** **static** ByteBuffer buffer = ByteBuffer.allocate(8);

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

        //测试 int 转 byte

**int** int0 = 234;

**byte** byte0 = intToByte(int0);

        System.out.println("byte0=" + byte0);//byte0=-22

        //测试 byte 转 int

**int** int1 = byteToInt(byte0);

        System.out.println("int1=" + int1);//int1=234

        //测试 int 转 byte 数组

**int** int2 = 1417;

**byte**[] bytesInt = intToByteArray(int2);

        System.out.println("bytesInt=" + bytesInt);//bytesInt=[B@de6ced

        //测试 byte 数组转 int

**int** int3 = byteArrayToInt(bytesInt);

        System.out.println("int3=" + int3);//int3=1417

        //测试 long 转 byte 数组

**long** long1 = 2223;

**byte**[] bytesLong = longToBytes(long1);

        System.out.println("bytes=" + bytesLong);//bytes=[B@c17164

        //测试 byte 数组 转 long

**long** long2 = bytesToLong(bytesLong);

        System.out.println("long2=" + long2);//long2=2223

    }

    //byte 与 int 的相互转换

**public** **static** **byte** intToByte(**int** x) {

**return** (**byte**) x;

    }

**public** **static** **int** byteToInt(**byte** b) {

        //Java 总是把 byte 当做有符处理；我们可以通过将其和 0xFF 进行二进制与得到它的无符值

**return** b & 0xFF;

    }

    //byte 数组与 int 的相互转换

**public** **static** **int** byteArrayToInt(**byte**[] b) {

**return**   b[3] & 0xFF |

                (b[2] & 0xFF) << 8 |

                (b[1] & 0xFF) << 16 |

                (b[0] & 0xFF) << 24;

    }

**public** **static** **byte**[] intToByteArray(**int** a) {

**return** **new** **byte**[] {

            (**byte**) ((a >> 24) & 0xFF),

            (**byte**) ((a >> 16) & 0xFF),

            (**byte**) ((a >> 8) & 0xFF),

            (**byte**) (a & 0xFF)

        };

    }

    //byte 数组与 long 的相互转换

**public** **static** **byte**[] longToBytes(**long** x) {

        buffer.putLong(0, x);

**return** buffer.array();

    }

**public** **static** **long** bytesToLong(**byte**[] bytes) {

        buffer.put(bytes, 0, bytes.length);

        buffer.flip();//need flip

**return** buffer.getLong();

    }

}

**4.在不使用 StringBuffer 的前提下，怎么反转一个字符串？**

4.Scanner sc=new Scanner(System.*in*);  
String str1=sc.nextLine();  
String str2="";  
char[] chars = str1.toCharArray();  
int len=chars.length-1;  
for(int i=len;i>=0;i--){  
 str2=str2+chars[i];  
}  
System.*out*.println(str2);

1. 复制文本文档？

BufferedReader br= null;  
 BufferedWriter bw= null;  
 String line= null;  
 try {  
 br = new BufferedReader(new FileReader("C:\\Users\\A\\Desktop\\新建文件夹\\1.txt"));  
 bw = new BufferedWriter(new FileWriter("C:\\Users\\A\\Desktop\\新建文件夹\\2.txt"));  
 line = null;  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 while((line=br.readLine())!=null){  
 bw.write(line);  
 bw.newLine();  
 bw.flush();  
 bw.close();  
}

1. **Java 中，怎么获取一个文件中单词出现的最高频率？**

|  |  |
| --- | --- |
| \s | 匹配任何不可见字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。 |
| \w | 匹配包括下划线的任何单词字符。类似但不等价于“[A-Za-z0-9\_]”，这里的"单词"字符使用Unicode字符集。 |
| \d | 匹配一个数字字符。等价于[0-9]。grep 要加上-P，perl正则支持 |

BufferedReader br=new BufferedReader(new FileReader("C:\\Users\\A\\Desktop\\新建文件夹\\1.txt"));  
 String line;  
 TreeMap<String, Integer> tm = new TreeMap<String, Integer>();  
 while ((line=br.readLine())!=null){  
 line.toLowerCase();  
 String regex1="\\s+";  
 String regex2="\\w+";  
 String[] str = line.split(regex1);  
 for (String s : str) {  
 if(s.matches(regex2)){  
 if(!tm.containsKey(s)){  
 tm.put(s,1);  
 }else {  
 tm.put(s,tm.get(s)+1);  
 }  
 }  
 }  
  
  
 }  
  
 System.*out*.println(tm);  
  
}

**6.Java 中，怎么打印出一个字符串的所有排列？**

String str = "";  
  
 Scanner scan = new Scanner(System.*in*);  
  
 str = scan.nextLine();  
  
 *permutation*(str.toCharArray(), 0);  
}  
  
public static void permutation(char[] str, int i) {  
 if (i >= str.length)  
 return;  
 if (i == str.length - 1) {  
 System.*out*.println(String.*valueOf*(str));  
 } else {  
 for (int j = i; j < str.length; j++) {  
 char temp = str[j];  
 str[j] = str[i];  
 str[i] = temp;  
  
  
 *permutation*(str, i + 1);  
  
  
 }  
 }  
}

## **关于 OOP 和设计模式的面试题**

这部分包含 Java 面试过程中关于 SOLID 的设计原则，OOP 基础，如类，对象，接口，继承，多态，封装，抽象以及更高级的一些概念，如组合、聚合及关联。也包含了 GOF 设计模式的问题。

1.抽象类和接口的区别有哪些？

**1.抽象类中可以没有抽象方法，但接口中的方法必须是抽象方法；**

**2.抽象类中可以存在普通属性，方法，静态属性和方法；**

**3.接口中的变量必须是static final类型的，必须被初始化，只有常量没有变量；**

**4.抽象类只能单继承，接口可以继承多个父接口。**

**2.**抽象类和接口如何选择？

答：

1. 如果要创建不带任何方法定义和成员变量的基类，那么就应该选择接口而不是抽象类。
2. 如果知道某个类应该是基类，那么第一个选择的应该是让它成为一个接口，只有在必须要有方法定义和成员变量的时候，才应该选择抽象类。因为抽象类中允许存在一个或多个被具体实现的方法，只要方法没有被全部实现该类就仍是抽象类。

**3.**Java 和 C++ 的区别：

解析：虽然我们不太懂 C++，但是就是会这么问，尤其是三面（总监级别）面试中。

答：

1. 都是面向对象的语言，都支持封装、继承和多态；
2. 指针：Java 不提供指针来直接访问内存，程序更加安全；
3. 继承： Java 的类是单继承的，C++ 支持多重继承； Java 通过一个类实现多个接口来实现 C++ 中的多重继承； Java 中类不可以多继承，但是！！！接口可以多继承；
4. 内存： Java 有自动内存管理机制，不需要程序员手动释放无用内存。