基础

1.java基本数据类型有哪些，int， long占几个字节

- byte boolean 1位
- char short 2位
- int float 4位
- long double 8位

2.== 和 equals有什么区别

初步了解在JVM中的内存分配知识

在JVM中，内存分为堆内存跟栈内存。他们二者的区别是： 当我们创建一个对象（new Object）时，就会调用对象的构造函数来开辟空间，将对象数据存储到堆内存中，与此同时在栈内存中生成对应的引用，当我们在后续代码中调用的时候用的都是栈内存中的引用。还需注意的一点，基本数据类型是存储在栈内存中。

初步认识equals与==的区别：

- ==是判断两个变量或实例是不是指向同一个内存空间，equals是判断两个变量或实例所指向的内存空间的值是不是相同
- ==是指对内存地址进行比较， equals()是对字符串的内容进行比较
- ==指引用是否相同， equals()指的是值是否相同

3.hashcode 和 equals作用

hashCode()的作用是获取哈希码，返回一个int整数，作用是查找hashMap的索引位置。

至于为什么要有hashCode()

用HashMap来举例hashCode()的作用，当往HashMap里插入一个元素之后，通过hashCode()确定插入的位置，如果该位置如果为空直接插入，如果有则equals()方法比较与该位置的下所用键值是否相同,如果相同就将value进行替换，否则插入在链表末尾

equals的作用

因为两个对象相等所以hashCode()必须相同，两个对象相等，两个对象调用的equals()方法返回为true。两个相等的hashCode(),并不一定是相同的对象。
hashCode() 的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode() ，则该class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）

4.new String创建了几个对象

- 一个对象是：**new关键字**在堆空间创建的
- 另一个对象是：字符串常量池中的**对象"ab"**。 字节码指令：ldc

5.位运算符的一些计算

1.**对于有符号的数而言，最高位为符号位，0表示正数，1是表示负数**
例如：1的有符号二进制值为00000001，-1的有符号二进制值为10000001
2.**正数的原码、反码、补码都一样**
例如：1的原码为00000001，反码也为00000001，补码也为00000001
3.**负数的反码为符号位不变，其他位取反，补码为反码+1**
例如：-1的原码为10000001，反码为11111110，补码为11111111

4.**0的反码、补码都为0**
在计算机运行时，都是以补码的方式来运算的，但是在我们看的时候看到的是原码

6.java的拆装箱

基本数据类型	包装类
byte	Byte
boolean	Boolean
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

7.compareable 和 compartor的区别

https://blog.csdn.net/belongtocode/article/details/102930203

数据结构

1.ArrayList和LinkedList的区别，优缺点

```
1.ArrayList是实现了基于动态数组的数据结构，LinkedList是基于链表结构。
2.对于随机访问的get和set方法，ArrayList要优于LinkedList，因为LinkedList要移动指针。
3.对于新增和删除操作add和remove，LinkedList比较占优势，因为ArrayList要移动数据。
```

2.hashmap实现，扩容是怎么做的，怎么处理hash冲突，hashcode算法等

3.链表需要知道。LinkedHashMap一般再问LRU的时候会问到

4.二分搜索树的特性和原理。前中后序遍历写出其中一种，当问到二分搜索树的缺点的时候，你需要提出基于二分搜索树的红黑树，说出他的特性。

5.堆的实现，最大堆，最小堆，优先队列原理。

简单算法

1.手写快速排序，插入排序，冒泡排序

- [x] 插入

```
// 插入排序
void InsertSort(int arr[], int len){
```

```
    // 检查数据合法性
    if(arr == NULL || len <= 0){
        return;
    }
    for(int i = 1; i < len; i++){
        int tmp = arr[i];
        int j;
        for(j = i-1; j >= 0; j--){
            //如果比tmp大把值往后移动一位
            if(arr[j] > tmp){
               arr[j+1] = arr[j];
            }
            else{
               break;
            }
        }
        arr[j+1] = tmp;
    }
}
```

☑ 快速

```
package algorithm.dynamic_programming.TreeTraverse;

import java.util.Arrays;

public class QuickSort {

    public static void quickSort(int[] arr, int left, int right){
        if (left < right){
            // 把数组分块
            int pivot = partition(arr, left, right);
            System.out.println(Arrays.toString(arr));
            // 基准元素左边递归
            quickSort(arr, left, pivot-1);
            // 基准元素右边递归
            quickSort(arr, pivot+1, right);
        }
    }

    public static int partition(int[] arr,int left,int right){
        int pivot = arr[left];       //默认第一个元素是基准元素
        while (left<right){
            //右边往左移动 直到遇到小于基准元素的
            while(left < right && arr[right] >= pivot){
                right--;
            }
            arr[left] = arr[right]; //记录那个较小的值

            //左往右移动 直到遇到大于基准元素的
            while (left < right && arr[left] <= pivot){
                left++;
            }
            arr[right] = arr[left]; //记录那个较大的值
        }
        arr[left] = pivot;           //把基准元素放到当前指向的地方
        return left;                 //返回基准元素的索引
    }
}
```

```
38
```

☑ 归并

```java
package algorithm.dynamic_programming;
import java.util.Arrays;
import java.util.Scanner;

public class MergeSort {

    public static void main(String[] args) {
        int arr[]={8,4,5,7,1,3,6,2};
        int temp[]=new int[arr.length];
        mergeSort(arr,0, arr.length-1,temp);
        System.out.println("并归排序后"+ Arrays.toString(arr));
    }
//分解方法
    public static void mergeSort(int[] arr, int left, int right, int[] temp) {
        if (left < right) {
            int mid = (left + right) / 2; //中间索引
            //向左递归进行分解
            mergeSort(arr, left, mid, temp);
            //向右递归进行分解
            mergeSort(arr, mid + 1, right, temp);
            //进行合并
            merge(arr,left,mid,right,temp);
        }
    }
    //合并的方法
    /**
     *
     *  @param  arr 排序的原始数组
     *  @param left 左边有序序列的初始索引
     *  @param mid 中间索引
     *  @param right 右边索引
     *  @param temp 做中转的数组
     **/
    public static void merge(int[] arr, int left, int mid, int right, int[] temp) {
        //初始化i,左边有序序列的初始索引
        int i = left;
        //初始化j,右边有序序列的初始索引
        int j = mid+1;
        int t = 0;//指向temp数组的当前索引

        while (i <= mid & j <= right) {//继续

            //如果左边的有序序列的当前元素，小于等于右边有序序列的当前元素
            //即将左边的当前元素，拷贝到temp数组
            //假设数组arr{1,3,5,6,2,4}
            //左边 0 - mid 即是{1,3,5}
            //右边 mid+1 -right 即是{6,2,4}
            //若arr[i]<= arr[j] 即是1 <= 6
            if (arr[i] <= arr[j]) {
                temp[t] = arr[i];//temp[0]=arr[i];
                t += 1;//指向temp数组下一位
                i += 1;//指向左边下一位arr[i+1]...
            }else{
                //反之arr[i] >= arr[j] 左边大于右边
                //则进行右边赋值给temp数组
```

```
                temp[t] = arr[j];//temp[0]=arr[i];
                t += 1;//指向temp数组下一位
                j += 1;//指向右边边下一位arr[j+1]...
            }
        }
        //把有剩余数据的一边的数据依次全部填充到temp
        //左边的有序序列还有剩余的元素，就全部填充到temp
        while( i <= mid){
            temp[t] = arr[i];
            t += 1;
            i += 1;
        }
        //右边的有序序列还有剩余的元素，就全部填充到temp
        while( j <= right){
            temp[t] = arr[j];
            t += 1;
            j += 1;
        }

        //将temp数组的元素拷贝到arr
        //为什么 t=0 ?
        //因为合并的时候按图所示数组：{8,4,5,7,1,3,6,2}
        //最先进入的是84 left=0 right = 1
        //经过上面的左边与右边比较，得出temp数组：4,8
        // 此时清空指向temp数组的下标指针t 重新回到0
        //tempLeft = 0 进行将temp数组里的4，8 赋值给arr数组
        t = 0;
        int tempLeft= left;
        while( tempLeft <= right){
            arr[tempLeft]=temp[t];
            t += 1;//赋值成功后指向temp数组的下标指针t往后移
            tempLeft +=1;//84 完成后到57 此时left=2 right = 3 ...
        }
    }
}
```

☑ 冒泡

```
public class BubbleSort {
    public static void main(String[] args) {
        int [] arr = {6,2,4,7,5,1,9,8,3};
        BubbleSort(arr);
        System.out.println(Arrays.toString(arr));
    }
    public static void BubbleSort(int[] arr){
        int swap=0;
        for(int i = 0;i< arr.length-1;i++){
            for (int j = 0;j<arr.length-1-i;j++){
                if(arr[j]>arr[j+1]){
                   swap = arr[j];
                   arr[j]=arr[j+1];
                   arr[j+1]=swap;
                }
            }
```

```
        }
    }
}
```

2.翻转一个数字

```
public int reverse(int x){
        long y = 0;
        while(x!=0){
            y = y*10+x%10;
            x = x/10;
        }
        return (int)y==y? (int) y :0;
    }
```

3.翻转一个链表

```
public class Solution {
    public ListNode ReverseList(ListNode head) {
//      如何调整链表指针，达到链表反转的目的。
        ListNode prev = null; // prev : 指向反转好节点的最后一个节点
        ListNode curr = head;    //指向反转链表的第一个节点
        while(curr != null){
            ListNode next = curr.next;
            curr.next = prev;

            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

4.O(n)复杂度找出数组中和是9的两个数的索引

5.写出二分搜索树前中后序遍历中的其中一个

```
//二叉树遍历-前序遍历
    public void preTraverse(TreeNode root){
        if (root!=null){
            return;
        }
        System.out.println(root.data);
        preTraverse(root.leftChild);
        preTraverse(root.rightChild);
    }
    //二叉树遍历-中序遍历
    public void inTraverse(TreeNode root){
        if (root!=null){
            return;
        }
        inTraverse(root.leftChild);
        System.out.println(root.data);
```

```
        inTraverse(root.rightChild);
    }
    //二叉树遍历-后序遍历
    public void afterTraverse(TreeNode root){
        if (root!=null){
            return;
        }
        afterTraverse(root.leftChild);
        afterTraverse(root.rightChild);
        System.out.println(root.data);
    }
```

6.实现一个队列，并能记录队列中最大的数。

JVM虚拟机我们需要知道他们内部组成：堆，虚拟机栈，本地方法栈，方法区，计数器。每一块都存放什么东西，以及垃圾回收的时候主要回收哪些块的东西。

面向对象

封装，继承，多态，抽象，反射，注解，设计模式，设计模式的原则。

面试中一般会问下：

1.抽象和接口有什么不一样

https://blog.csdn.net/m0_51358164/article/details/125153230

2.工作中常用的设计模式，一些源码中的设计模式

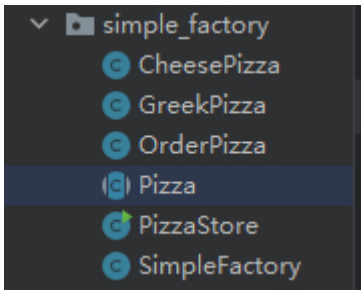
https://blog.csdn.net/qq_36386908/article/details/123416250

单例模式

```
public class SingletonTestFinal {
    public static void main(String[] args) {
        Singleton instance1 = Singleton.INSTANCE;
        Singleton instance2 = Singleton.INSTANCE;
        System.out.println(instance1 == instance2);
    }
}

enum Singleton {
    INSTANCE;
}
```

工厂模式




```java
// 1.披萨抽象类
public abstract class Pizza {

    protected String name;

    public void setName(String name) {
        this.name = name;
    }

    public abstract void prepare();

    public void bake() {
        System.out.println(name + "baking;");
    }

    public void cut() {
        System.out.println(name + "cut;");
    }

    public void box() {
        System.out.println(name + "box;");
    }
}
// 2.希腊披萨
public class GreekPizza extends Pizza {
    @Override
    public void prepare() {
        System.out.println(name + "披萨准备原材料");
    }
}
// 3.奶酪披萨
public class CheesePizza extends Pizza {
    @Override
    public void prepare() {
        System.out.println(name + " 准备原材料");
    }
}
//4.披萨工厂根据用户的而不同需求，生产不同类别的披萨
public class SimpleFactory {

    public Pizza createPizza(String pizzaType) {
        Pizza pizza = null;
        if("cheese".equals(pizzaType)) {
            pizza = new CheesePizza();
            pizza.setName("cheese");
        }else if("greek".equals(pizzaType)) {
            pizza = new GreekPizza();
            pizza.setName("greek");
        }
        return pizza;
    }
}
```

```java
// 5.订购披萨
public class OrderPizza {

    private static Scanner scanner = new Scanner(System.in);
    private SimpleFactory simpleFactory;

    // 传入工厂
    public OrderPizza(SimpleFactory simpleFactory) {
        this.simpleFactory = simpleFactory;
    }

    public void getPizza() {
        while (true) {
            System.out.println("请输入披萨类型：");
            // 从工厂获取披萨
            Pizza pizza = simpleFactory.createPizza(scanner.next());
            if (pizza != null) {
                pizza.prepare();
                pizza.bake();
                pizza.cut();
                pizza.box();
            } else {
                System.out.println("暂无该类型披萨！");
                break;
            }
        }
    }
}
// 6.购买披萨
public class PizzaStore {
    public static void main(String[] args) {
        // 创建披萨订购类，并传入披萨工厂
        OrderPizza orderPizza = new OrderPizza(new SimpleFactory());
        orderPizza.getPizza();
    }
}
```

3.具体给你一个设计模式让你说说你对他的了解，比如观察者，工厂。

以上这些东西主要考察你的代码设计能力。

简单工厂

> 优点
>
> 工厂类包含必要的逻辑判断，可以决定在什么时候创建哪一个产品的实例。客户端可以免除直接创建产品对象的职责，很方便的创建出相应的产品。工厂和产品的职责区分明确。
> 客户端无需知道所创建具体产品的类名，只需知道参数即可。
> 也可以引入配置文件，在不修改客户端代码的情况下更换和添加新的具体产品类
>
> 缺点
>
> 简单工厂模式的工厂类单一，负责所有产品的创建，职责过重，一旦异常，整个系统将受影响。且工厂类代码会非常臃肿，违背高聚合原则。
> 使用简单工厂模式会增加系统中类的个数（引入新的工厂类），增加系统的复杂度和理解难度
> 系统扩展困难，一旦增加新产品不得不修改工厂逻辑，在产品类型较多时，可能造成逻辑过于复杂
> 简单工厂模式使用了 static 工厂方法，造成工厂角色无法形成基于继承的等级结构

网络编程

1.互联网的实现主要分为几层，http、ftp、tcp、ip分别位于哪一层。

```
应用层----HTTP协议
传输层----TCP协议
网络层----IP协议
链接层
物理层
```

2.http和https的区别

```
HTTP协议以明文方式发送内容，不提供任何方式的数据加密。HTTP协议不适合传输一些敏感信息，比如：信用卡号、密码等支付信息。https则是具有安全性的ssl加密传输协议。http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。并且https协议需要到ca申请证书。HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。
```

3.为什么tcp要经过三次握手，四次挥手

```
三次握手
用于保证可靠性和流量控制维护的某些状态信息，这些信息的组合，包括Socket、序列号和窗口大小称为连接。
三次握手才可以阻止重复历史连接的初始化（主要原因）
三次握手才可以同步双方的初始序列号
三次握手才可以避免资源浪费

四次挥手
第一次挥手客户端发起关闭连接的请求给服务端；
第二次挥手：服务端收到关闭请求的时候可能这个时候数据还没发送完，所以服务端会先回复一个确认报文，表示自己知道客户端想要关闭连接了，但是因为数据还没传输完，所以还需要等待；
第三次挥手：当数据传输完了，服务端会主动发送一个 FIN 报文，告诉客户端，表示数据已经发送完了，服务端这边准备关闭连接了。
第四次挥手：当客户端收到服务端的 FIN 报文过后，会回复一个 ACK 报文，告诉服务端自己知道了，再等待一会就关闭连接。
```

4.socket了解过吗

Android部分

1.四大组件有哪些，说出你对他们在Android系统中的作用和理解。

```
安卓四大组件：Activity、Service、BroadcastReceiver和ContentProvider，作用：

1、Activity组件的主要作用是展示一个界面并和用户交互，它扮演的是一种前台界面的角色

Activity是一种展示型组件，主要是向用户展示一个界面，并且可以接收用户的输入信息从而和用户进行交互。对用户来说，Activity就是Android应用的全部，因为其他三大组件对用户来说是不可感知的。Activity的启动由Intent触发，其中Intent分为显式启动和隐式启动。

2、Service组件的主要作用是在后台执行计算任务，执行任务的结果可以和外界进行通信
```

```
Service是一种计算型组件，用于在后台执行一系列计算任务。由于Service组件工作在后台，因此用户无法直接感知到它的存在。Service组件和Activity组件不同，Activity组件只有一种运行模式，即Activity处于启动状态，但是Service组件却有两种状态：启动状态和绑定状态。Service组件处于启动状态时，它的内部可以执行一些后台计算，并且不需要和外界有直接的交互。Service处于绑定状态，Service内部同样也可以执行后台计算，但是处于这种状态的Service可以很方便地和外界进行通信。

3、BroadcastReceiver组件的主要作用是消息的传递，该消息的传递可以在应用内，也可以在应用之间，它的角色是一个消息的传递者

BroadcastReceiver是一种消息型组件，用于在不同组件乃至不同应用之间传递消息。BroadcastReceiver同样无法被用户所感知，因为它工作在系统内部。BroadcastReceiver也叫做广播，广播的注册方式有两种：静态注册和动态注册。静态注册指在AndroidManifest中注册广播，这种广播在应用安装时被系统解析，此种形式的广播不需要应用启动就可以接收到相应的广播。动态广播需要通过Context.registerReceiver()来实现，并且在不需要的时候通过Context.unRegisterReceiver()解除广播，此种形态的广播必须要应用启动才能注册并接收广播。

4、ContentProvider组件的主要作用是作为一个平台，提供数据的共享，并且提供数据的增删改查功能。主要应用于应用之间的数据共享场景

ContentProvider是一种数据共享型组件，用于向其他组件乃至其他应用共享数据。同样的，它也无法被用户所感知。对于ContentProvider组件来说，它的内部需要实现增删改查这四种操作。需要注意的是，ContentProvider内部的delete、update和query方法需要处理好线程同步，因为这几个方法都是在Binder线程池中被调用的。ContentProvider组件不需要手动停止。
```

2.Activity生命周期，A启动B两个页面生命周期怎么运行的，为什么会 这样，生命周期为什么这么设计，你有了解过吗。

https://blog.csdn.net/lentolove/article/details/124170429

3.四种启动模式，内部堆栈是怎么回事

同上

4.Activity的启动过程，这个我强烈建议每个Android开发人员都要清楚的知道，并且跟一下源码，几个核心类的作用。你会对Android有一个更好的认识。

```
（1）点击桌面APP图标，Launcher进程采用Binder IPC的方式向system_server进程的AMS(ActivityManagerService)发起startActivity的请求。
（2）system_server进程接收到请求后，采用Socket IPC向Zygote进程发出创建APP进程的请求；
Zygote进程fork出新的进程，即APP进程；
（3）APP进程通过Binder IPC向system_server进程发起attachApplication请求；
system_server进程在接收到请求后，进行一系列的准备工作后，再通过Binder IPC向APP进程发送scheduleLaunchActivity的请求
（4）APP进程接收到请求后，通过Handler向主线程发送LAUNCH_ACTIVITY消息，创建目标Activity，进入Activity的生命周期
```

5.事件分发流程，怎么处理滑动冲突。举例：长按ListView的一个Item它变灰了。这个时候在滑动。item恢复原来的样子，这个时候他们内部的事件传递是什么样子。有很多种问法，所以你一定要搞清楚。

http://t.zoukankan.com/huolongluo-p-6431688.html

6.自定义View,View的绘制流程。onMeasure,onLayout,onDraw都是什么作用。ViewGroup是怎么分发绘制的。onDraw里面怎么去做绘制，Canvas,Path,Paint你都需要了解。并且配合ValueAnimtor或者Scroller去实现动画。有时候面试的会突发奇想问你ViewGroup是树形结构，我想知道树的深度，你怎么计算，突然就变成了一个数据结构和算法的题。

7.Bitmap和Drawable

```
Bitmap：称作位图，一般的位图的文件格式扩展名为.bmp，当然编码器也有很多，RGB565，RGB8888，作为一种追个像素的显示对象，执行效率高，但是存储效率低，可以理解成一种存储对象
Drawable：Android下的通用的图片形象，它可以装载常用格式的图像，比如GIF,PNG,JPG，BMP，提供一些高级的可视化方法。
```

8.Animation和Animator
9.LinearLayout、RelativeLayout、FrameLayout三种常用布局的特性，他在布局的时候是怎么计算的。效率如何。CoordinatorLayout配合AppbarLayout的使用，以及自定义Behavior。ConstraintLayout的使用。用来减少层级。

https://blog.csdn.net/ShineKaizi/article/details/95164497

10.Handler消息机制，推荐看一下Looper的源码

https://blog.csdn.net/xingyu19911016/article/details/117326695

11.进程间通信，Binder机制
12.AsyncTask源码看一下。
13.图片的压缩处理，三级缓存，Lru算法
14.分辨率和屏幕密度，以及计算一个图片大小。mdpi,hdpi的关系和比例。

https://blog.csdn.net/wujian543/article/details/79929983

15.优化，内存优化，布局优化，启动优化，性能优化。内存泄露，内存溢出。怎么优化，用了什么工具，具体怎么做的。

16.listView和RecycleView对比，以及缓存策略。

https://blog.csdn.net/yuan1244487110/article/details/90401874

17.MVC,MVP,MVVM

https://blog.csdn.net/zg0601/article/details/123587933

18.RecyclerView四大块，能实现什么效果，大致怎么实现的，心里要有数