1. 九种数据类型大小，以及他们的封装类

char 2Bite Character /u0000(null)
byte 1B Byte 0(bity)
short 2B Short 0(short)
int 4B Integer 0
long 8B Long 0L
boolean 1B Boolean false
float 4B Float 0F
double 8B Double 0D
void Void
注：基本数据类型占用的空间是不变的，这种不变性也是 java 具有可移植性的原因之一。它是放在栈中直接存储值，声明时系统会自动给它分配存储空间，所有的数值类型都有正负号。而引用类型声明时只是分配了引用空间（栈内存）

2.Switch 能否用 String 做参数

在 JDK1.7 版本以前不能用。低版本仅支持 byte、char、short、int 和 Enum 类型。
在 JDK1.7 以后增加对 Sring 的支持。

3.equal 与 == 的区别

“==”： 基本类型，比较变量对应的值是否相等
对象/引用，比较引用变量的内存地址是否相等（栈内存上的引用，堆内存上的地址。

“equals”: 基本类型，没有该方法。
对象/引用，对于 String 类型（或其他的包装类 Long、Integer..）的引用对象，是比较内容是否相等（String 内部重写了 equals()），其他对象则是比较内存地址是否相等。（需要重写 equals()方法）

4.object 有什么共有方法

equals()
hashcode()
toString()
finalize() 垃圾回收
wait()
notify()
getClass()

5.Java 四大引用

强引用：我们正常 New 出一个对象，这个对象就是强引用对象。强引用对象是我们开发中最常见最普遍的一个引用。拥有强引用的对象垃圾回收器都不会去回收。

软引用：用 SoftReference 来创建一个对象。这个对象就是软引用对象。软引用对象一般不会被垃圾回收器回收。直到内存空间不足的时候才会被回收。软引用一般可以用来实现对象缓存机制。

注：软引用可以和一个引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收器回收，Java 虚拟机就会把这个软引用加入到与之关联的引用队列中。

弱引用：用 WeakReference 来创建一个对象。这个对象就是弱引用对象。弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。

虚引用：与其他几种引用都不同，虚引用并不会决定对象的生命周期。如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。虚引用来跟踪对象的垃圾回收过程。

6.Hashcode 的作用

Hashcode()用来返回对象的哈希码值。Hashcode 用来提高数据结构中数据的查询速度。（时间复杂度从 On –> O1

如果两对象的 equals() 方法相等，则在两个对象中的每个对象上调用 hashCode()都必须生成相同的整数结果。

如果 equals 方法被重写，则要求 hashcode()方法也要重写。

7. ArrayList、LinkedList、Vector 的区别

（1）ArrayList：底层基于数组的数据结构存储数据，当数据超出数组范围时，其大小将会动态的进行增长（50%增长）。没有使用线程安全，性能会比较高。

（2）Vector：底层基于数组的数据结构存储数据，当数据超出数组范围时，其大小将会动态的进行增长（100%增长）。Vestor 底层进行了同步保证线性安全，性能相对 ArrayList 会低些。

（3）LinkedList：底层基于双向链表的数据结构存储数据，因此没有太多的数据超出的顾虑。

注：一般来说：

1. 对于数据的查询和更新，使用 ArrayList 或 Vector 优势表较大，对于给定索引下标，ArrayList 和 Vector 查找数据的时间复杂度为 O1，而 Linked 需要移动链表指针，时间复杂度为 On。
2. 同理，对于数据的增删操作，使用 LinkedList 的优势更大，因为对给定位置的数据增删 LinkedList 只需要前后对象指针，时间复杂度为 O1，而 ArrayList 和 Vector 在增删指定位置数据后，还要对后面数据进行移动，时间复杂度为 On。
3.

8.String、StringBuffer、StringBuilder 的区别

String：字符串常量，一旦初始化就不能再修改他的值。以后对 String 类型的数据修改都是重新创建一个新对象。

StringBuffer/StringBuilder：字符串缓冲区，是一个可变的对象，与 String 区别在于对字符串修改是不用重新创建一个新对象。对于需要经常修改的字符串推荐使用 StringBuffer 或 StringBuilder 来存储数据，提高系统性能。

StringBuffer 和 StringBuilder 的区别在于，StringBuffer 底层实现了同步，是线程安全的。而 StringBuilder 底层没有实现同步，因此性能会相对高些。

8.Map、Set、List、Queue、Stack 的特点和用法

Collection 三个子接口：List、Set、Queue

List 三个主要实现类：ArrayList、LinkedList、Vector

Set 两个主要实现类：HashSet、TreeSet

Map 又有四个实现类：HashMap、Hashtable、TreeMap、LinkedMap

Stack 是基于 Vector 的实现。

以上都是存储数据的容器，

对于 Map：基于键值映射，Key、Value 一一映射，容器内不能包含重复的 Key。

TreeMap，数据有序，键不可以为 null 值，内部实现使用红黑树实现的；

HashMap，数据无序，线程不安全，键、值 都可以为 null。

HashTable，是线程安全的，不能存储 null 值

对于 List：有序的可以索引到元素的容器，并且里面的元素可以重复。

ArrayList 是线程不安全的， Vector 是线程安全的，这两个类底层都是由数组实现的。

LinkedList 是非线程安全的，底层是由双向链表实现的。

查询、更新 使用 ArrayList / Vector，插入、删除 使用 LinkedList

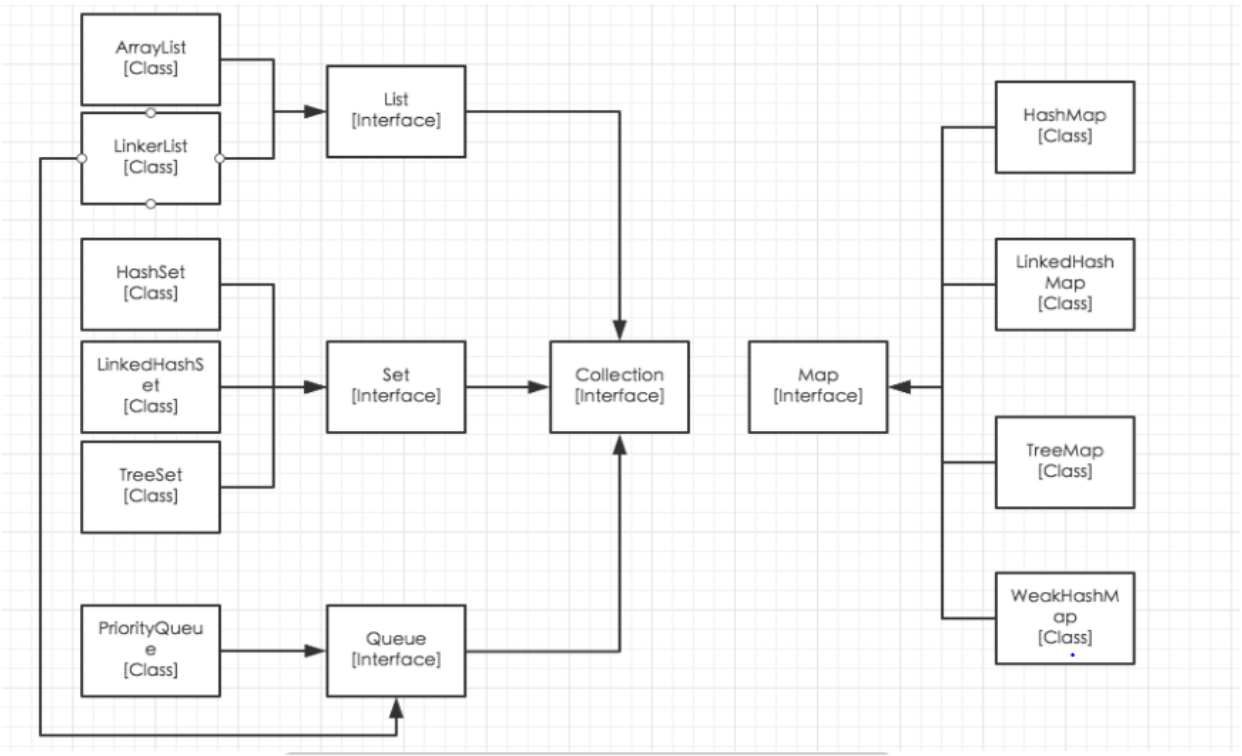
对于 Set：不包含重复元素的集合，Set 中最多包含一个 null 元素，支持 Iterator/for-earch 遍历。

HashSet，无序的、无重复的数据集合。基于 HashMap。（允许有一个 Null 值）

TreeSet，有序的，无重复的数据集合。元素需要实现 Comparable 接口。基于 TreeMap（不允许有 Null 值）

对于 Queue：遵从先进先出原则。

对于 Stack：遵从后进先出原则。Stack 是 Vector 的实现，常用方法 push 和 pop 操作，以及取堆栈顶点的 peek()



New集合继承关系图

10. HashMap 和 HashTable 的区别

HashMap 和 Hashtable 都是 Map 接口的实现。

区别在于 HashMap 是非线程安全的，涉及线程同步问题需要自行增加同步处理。而 Hashtable 底层实现了线程同步，性能相对 HashMap 会低一些。

HashMap 的 Key 允许为 Null（1 个），而 Hashtable 的 Key 不允许为 Null。

11.HashMap 和 ConcurrentHashMap 的区别，HashMap 的底层源码

---HashMap 和 ConcurrentHashMap(分段数组+链表)底层都是由数组+链表的离散数据结构构成。

HashMap 是非线程安全的，底层没有进行线程同步处理。

ConcurrentHashMap 是线程安全的，底层实现线程同步。

ConcurrentHashMap(高版本 JDK)的同步不像 HashTable 那样（全表锁）的方式实现，实现同步的细节更复杂（分段锁），性能更高。

在没有线程并发问题情况下使用 HashMap 会好些，否则在多线程并发环境下使用 ConcurrentHashMap 会更好。

扩容：当 Map 中元素总数超过 Entry 数组的 75%，触发扩容操作 newSize = oldSize * 2

12.TreeMap、HashMap、LinkedHashMap 的区别

都是 Map 接口的实现类。

TreeMap 底层是通过红黑树（二叉树）实现，数据是有序的。

HashMap 是最常见也是最常用的 Map 容器，底层是通过 hashcode 来存储数据。只能有一个键为 Null。本身是非线程安全的，需要同步可使用 Collections.synchronizedMap(Map m);

LinkedHashMap 底层通过链表和哈希表的数据结构实现。因此对数据的增删操作性能效率更高。而查询或更新数据使用 HashMap 优势更大。

13.Clooection 包结构，Collections 的区别

Collection 是集合的上级接口，也就是 List、Set 和 Queue 的父接口。

Collections 是针对集合的辅助类，能够加强基本集合的功能，比如对集合实现同步（ synchronsizedList(List l) ），或者对集合进行排序（ sort() ）；

14.try、catch、finally 的区别，try 里面 return finally 还会执行吗？

Try、Catch 块：一般代码有可能出现 Expection 异常情况是需要执行 trycatch 操作。

而 finally 中代码能够在代码执行后期或出现异常之后做一些收尾工作，比如关闭流资源，关闭数据库连接等等。。

Finally 中的代码一般都会执行到，无论 Try 块中是否出现 Expction 异常，除非手动强制关闭 Jvm 虚拟机才不会被执行。

15.Excption 与 Error 包结构，其中 OOM 有遇到过哪些情况。SOF 有遇到过没？

一般 Java 的错误异常结构分为 3 种：

1 错误 Error

代码出现重大错误，直接导致程序终止。比如内存溢出、栈溢出等。

2 运行时异常 RuntimeExcption

这类异常发生原因一般是程序的 BUG，无法通过 trycatch 处理，典型例子：NullPointerException 、ClassCastException 、ArrayIndexOutOfBoundsException 。。

3 受检异常/编译时异常 Checked Excption

受检异常能够通过 trycatch 处理，或者 throw 抛给上级。典型例子：各种 IOExcption、

EOFException(end of file)、FileNotFoundExcption。。

其中 Error 错误和 RuntimeExcption 运行时异常都不能被编译器检测。遇到就需要通过检查代码或 Debug 程序。

OOM: 内存溢出，当内存占有量超过虚拟机分配的最大值时就会产生 OOM。

一般产生原因：加载对象多大、资源过多，来不及加载、各种内存泄露也有可能引发 OOM。

一般解决办法 :对象用软引用来封装、对图片进行边界压缩、缓存处理。修改虚拟机的堆内存分配大小。

SOF: 栈溢出，当应用程序递归太深而发生堆栈溢出时，抛出该 SOF 异常。

一般产生原因：程序内递归调用、大量使用循环或循环嵌套。。

一般解决办法：避免程序的递归调用，或大量循环。

16.Java 面向对象的三特征和含义

继承、封装、多态

继承：继承的概念是相对与子类和父类而言的，子类拥有父类的属性和方法，同时也有自己的独立行为。

封装：对类内部实现细节进行抽取、封装，隐藏内部实现、对外暴露最简单的接口。

多态：接口的多种不同的实现方式即为多态。相同的行为，不同的实现。代码中就是同一个接口使用不同的实例而执行不同操作。

17.Override 和 OverLoad 的区别

Override：重写方法，一般对于子类和父类而言的概念。子类方法用 Override 声明就代表该方法是重写父类的方法。

OverLoad：重载方法，一般对于同一个类而言，表示同一个类中能够有多个名字相同的方法，但参数类型和参数个数不同。

Override 是子父类间多态的表现，OverLoad 是一个类上的多态表现。

18.Interface 和 Abstract 的区别

Interface 声明的是接口，abstract 声明的是抽象类或抽象方法。

Interface 接口上的方法和变量都是共有的，用于让实现类实现。

abstract 抽象类中则是可以声明私有方法或私有变量。子类实现只需实现 abstract 声明的抽象方法。和选择性的重写方法。

一般来说开发中最顶级的是接口，然后抽象类实现接口，最后写具体的实现类。

19.Static Class 和 Not Static Class 的区别

一般来说 Static 指的是静态内部类，而 Not Static Class 则是指非静态内部类。

对于静态内部类：

不需要有指向外部类的引用。

只能访问外部类的静态成员，不能够访问外部类非静态成员。

对于非静态内部类：

会持有外部类的引用（导致内存泄露的常见原因）

能够访问外部类的静态和非静态成员。
非静态内部类的创建不能脱离外部类实例。

20.Java 中多态的实现

多态的含义简单来说是相同的方法拥有不同的实现。
一般来说多态的实现分为两种：
子类父类而言：子类重写父类方法，常见的是用父类引用指向子类对象。在运行时动态确认子类具体实现。
对单独的一个类而言：通过方法重载来实现类内多态。

21.实现多线程的两种方式：Thread、Runable
（1）继承 Thread 然后去重写他的 run()方法。创建该 Thread 类调用 Start 方法；
缺点：Java 只能单继承，一旦集成了 Thread 类就无法基层其他类。
（2）实现 Runnable 接口，实现 run()方法。然后通过 new Thread(Runnable run)把 runnable 对象传入进去，调用 start()方法即可实现多线程。（推荐）

```
public class ThreadTest extends Thread{
继承 Thread，重写 run（）方法
    public static class NewThread extends Thread {
        @Override
        public void run() {
            System.out.println("This is Thread Running~");
        }
    }

    //实现 runnable 接口
public static class MyRunable implements Runnable{
        @Override
        public void run() {
            System.out.println("This is My Runnable!~");
        }
    }


    public static void main(String[] args) throws InterruptedException {
        new Thread(new MyRunable()).start();

        new NewThread().start();
    }
}
```

22.线程同步的方法：synchronized、lock、reentrantLock

Synchronized 是 Java 的一个关键字，是一种同步锁。他能够修饰代码块、静态或非静态方法

修饰方法时，一般以当前对象作为锁。

修饰静态方法时，一般以类 Class 对象作为锁。

修饰代码块时，需要自行传入一个对象作为同步锁。

Lock 是一个接口，而 ReentrantLock 则是 Lock 接口的实现。ReentrantLock 拥有 Synchronized 的功能，都是用于解决多线程环境下并发同步的问题。

23.锁的等级：方法锁、对象锁、类锁

方法锁：一般通过 synchronized 关键字在方法上声明。

```
public synchronized void method() {
        System.out.println("我是对象锁也是方法锁");
    }
```

对象锁：通过 synchronized 修饰的方法或 synchronized（this）同步的代码块。进入同步区域时，需要获取对象锁。

```
public synchronized void method1() {
    // 同步方法
    System.out.println("对象锁：方式一");
}

public void method2() {
    // 同步代码块
    synchronized (this) {
        System.out.println("对象锁：方式二");
    }
}
```

类锁：通过 synchronized 关键字修饰的静态代码或 synchronized（xx.class）同步的代码块。（静态方法或静态变量在内存中都只存在一份，因此同步是共用一把锁）进入同步区域时，需要先获取类锁。

```
public static synchronized void method1() {
    System.out.println("类锁:方式一");
}

public void method2() {
    synchronized (Trhead_.class) {
        System.out.println("类锁：方式二");
```

```
        }
    }
```

24.手撕生产者、消费者模式

25.ThreadLocal 的作用和设计理念

ThreadLocal 是一个创建线程私有变量的类，一般情况我们创建变量都能够被所有线程所修改，而使用 ThreadLocal 创建的变量只能够被当前线程所访问和修改，其他线程无法被访问。

显然，ThreadLocal 这个类的目的是用来解决多线程环境下的并发同步问题。(ThreadLocal 的本质不是 Thread)

26.ThreadPool 的用法和优势

ThreadPool 也就是线程池。

优势 :通过固定数量的线程来统一为大量的操作服务，减少高并发环境下大量线程的创建和销毁所消耗的时间和资源。从而执行提高效率。

用法：Executor.newFixedThreadPool(4);

executor.execute(new Runnable());

底层：

```
executor = new ThreadPoolExecutor(corePoolSize, maximumPoolSize, keepAliveTime,
        TimeUnit.SECONDS, new LinkedBlockingDeque<Runnable>(),
        Executors.defaultThreadFactory(), new ThreadPoolExecutor.AbortPolicy());
```

27.Concurrent 包下的其他：ArrayBlockingQueue、CountDownLatch..

ArrayBlockingQueue: 一个由数组支持的有界阻塞队列。FIFO 原理。

CountDownLatch: 是一个倒数计数的锁，当倒数到 0 时触发事件，也就是开锁。

28.wait() 和 sleep() 区别

wait 是 object 的方法，sleep 是 Thread 的方法。

每当执行到 wait 或 sleep 时，进程都会停止执行。

他们的区别在于：

wait：wait 睡眠时会释放对象锁，不占用 cpu 资源。

sleep：sleep 睡眠时会一直保持对象锁，会一直占用 cpu 资源。

29.forEach 和 For 循环的效率对比

ForEach 是属于高版本的语法糖，底层实现是通过 iterater 进行遍历。

对于一般的遍历，普通 For 循环效率会更高。其次是 iterater 迭代器，效率最低的是 forEach 遍历。

For > Iterater > ForEach

30.Java IO 和 NIO

Java 中的 IO 一般分为：本地 IO 和 网络 IO
本地 IO 就是应用从本地磁盘中读写数据。
网络 IO 就是通过 ip+端口进行的远程数据读写操作。

NIO 即 No-Blocking IO：非阻塞式 IO，在输入输出操作的同时可以做别的事。
而 Java IO 是阻塞式 IO，读写数据时不能做其他事。

JavaIO 默认没有缓冲区的，而 NIO 有缓冲区。

31.反射的原理和作用

原理：我们的 Java 程序要被执行，走需要将编译完成的 Java 类加载到虚拟机上运行。这样的程序执行我们在编译期就知道哪个类被加载。　　而我们反射则是在编译期并不知道加载类的信息，知道运行时才动态得去获取和加载这个类。

作用：反射一般都 3 种用途：

（1） 运行时动态获取对象所属的类。
（2） 运行时动态的构造类对象。
（3） 运行时动态获取类的属性和方法。（getFields()、getMethods()）

32.泛型的常用特点，List<String>能否转为 List<Object>

实现类型安全：泛型定义了变量类型的约束，另变量在编译阶段就能够进行类型检查。
消除强制类型转换：使用泛型的话，能够避免强制类型的转换。

List<String>理论上不该转化为 List<Object>：强转-> List<Object> objs = (List<String>)strs

33.解析 XML 的几种方法原理及特点：DOM、SAX、PULL

DOM:　将整个 xml 文档树加载进内存。能够读写 xml 内容。。
SAX:　采用事件驱动模型，不需要将整个 xml 文档加载进内存。内存占用小。不能写入 xml
PULL:　android 内置的 xml 解析器。原理跟 SAX 解析类似。Android 下开发推荐 PULL。解析效率更高。

34.Java 和 C++对比

Java 跨平台，程序移植性好。
C++程序性能、运行速度、执行效率。
Java 更多用来开发网站、手机应用、小型软件居多。C++用来开发驱动、游戏、嵌入式系统等对性能要求比较高的东西。
Java 开发效率高、C++执行效率高。

35.Java1.7 和 Java1.8 的新特性。

JDK1.7：Switch 中可以使用 String 作为参数。　　一个 catch 内可铺货多个异常，用 “ | ”隔开。
JDK1.8：接口中实现具体方法、静态方法。　Lambada 函数，简化代码书写。

36.设计模式：单例、工厂、适配器、责任链、观察者。

37.JNI 的使用

2D/3D 绘画渲染、各种驱动、媒体播放器等等高性能的需求一般都要使用 JNI 调用本地 C/C++方法.
Java Native Interface - Java 本地调用.
用来调用 Native 底层本地方法：C/C++函数。Native 方法又能反向调用 Java 代码。