基础篇

1、 Java语言有哪些特点

1、简单易学、有丰富的类库

2、面向对象（Java最重要的特性，让程序耦合度更低，内聚性更高）

3、与平台无关性（JVM是Java跨平台使用的根本）

4、可靠安全

5、支持多线程

2、面向对象和面向过程的区别

面向过程：是分析解决问题的步骤，然后用函数把这些步骤一步一步地实现，然后在使用的时候一一调用则可。性能较高，所以单片机、嵌入式开发等一般采用面向过程开发

面向对象：是把构成问题的事务分解成各个对象，而建立对象的目的也不是为了完成一个个步骤，而是为了描述某个事物在解决整个问题的过程中所发生的行为。面向对象有**封装、继承、多态**的特性，所以易维护、易复用、易扩展。可以设计出低耦合的系统。 但是性能上来说，比面向过程要低。

3 、八种基本数据类型的大小，以及他们的封装类

基本类型	大小（字节）	默认值	封装类
byte	1	(byte)0	Byte
short	2	(short)0	Short
int	4	0	Integer
long	8	0L	Long
float	4	0.0f	Float
double	8	0.0d	Double
boolean	-	false	Boolean
char	2	\u0000(null)	Character

注:

1.int是基本数据类型，Integer是int的封装类，是引用类型。int默认值是0，而Integer默认值是null，所以Integer能区分出0和null的情况。一旦java看到null，就知道这个引用还没有指向某个对象，再任何引用使用前，必须为其指定一个对象，否则会报错。

2.基本数据类型在声明时系统会自动给它分配空间，而引用类型声明时只是分配了引用空间，必须通过实例化开辟数据空间之后才可以赋值。数组对象也是一个引用对象，将一个数组赋值给另一个数组时只是复制了一个引用，所以通过某一个数组所做的修改在另一个数组中也看的见。

虽然定义了boolean这种数据类型，但是只对它提供了非常有限的支持。在Java虚拟机中没有任何供boolean值专用的字节码指令，Java语言表达式所操作的boolean值，在编译之后都使用Java虚拟机中的int数据类型来代替，而boolean数组将会被编码成Java虚拟机的byte数组，每个元素boolean元素占8位。这样我们可以得出boolean类型占了单独使用是4个字节，在数组中又是1个字节。使用int的原因是，对于当下32位的处理器（CPU）来说，一次处理数据是32位（这里不是指的是32/64位系统，而是指CPU硬件层面），具有高效存取的特点。

4、标识符的命名规则。

标识符的含义：
是指在程序中，我们自己定义的内容，譬如，类的名字，方法名称以及变量名称等等，都是标识符。

命名规则：（硬性要求）
标识符可以包含英文字母，0-9的数字，$以及_
标识符不能以数字开头
标识符不是关键字

命名规范：（非硬性要求）
类名规范：首字符大写，后面每个单词首字母大写（大驼峰式）。
变量名规范：首字母小写，后面每个单词首字母大写（小驼峰式）。
方法名规范：同变量名。

5、instanceof 关键字的作用

instanceof 严格来说是Java中的一个双目运算符，用来测试一个对象是否为一个类的实例，用法为：

```
boolean result = obj instanceof Class
```

其中 obj 为一个对象，Class 表示一个类或者一个接口，当 obj 为 Class 的对象，或者是其直接或间接子类，或者是其接口的实现类，结果result 都返回 true，否则返回false。

注意：编译器会检查 obj 是否能转换成右边的class类型，如果不能转换则直接报错，如果不能确定类型，则通过编译，具体看运行时定。

```
int i = 0;
System.out.println(i instanceof Integer);//编译不通过  i必须是引用类型，不能是基本类型
System.out.println(i instanceof Object);//编译不通过
```

```
Integer integer = new Integer(1);
System.out.println(integer instanceof  Integer);//true
```

```
//false    ,在 JavaSE规范 中对 instanceof 运算符的规定就是：如果 obj 为 null，那么将返回false。
System.out.println(null instanceof Object);
```

6、Java自动装箱与拆箱

装箱就是自动将基本数据类型转换为包装器类型（int-->Integer）；调用方法：Integer的valueOf(int) 方法

拆箱就是自动将包装器类型转换为基本数据类型（Integer-->int）。调用方法：Integer的intValue方法

在Java SE5之前，如果要生成一个数值为10的Integer对象，必须这样进行：

```
Integer i = new Integer(10);
```

而在从Java SE5开始就提供了自动装箱的特性，如果要生成一个数值为10的Integer对象，只需要这样就可以了：

```
Integer i = 10;
```

面试题1： 以下代码会输出什么？

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

        Integer i1 = 100;
        Integer i2 = 100;
        Integer i3 = 200;
        Integer i4 = 200;

        System.out.println(i1==i2);
        System.out.println(i3==i4);
    }
}
```


运行结果：

```
true
false
```

为什么会出现这样的结果？输出结果表明i1和i2指向的是同一个对象，而i3和i4指向的是不同的对象。此时只需一看源码便知究竟，下面这段代码是Integer的valueOf方法的具体实现：

```
public static Integer valueOf(int i) {
        if(i >= -128 && i <= IntegerCache.high)
            return IntegerCache.cache[i + 128];
        else
            return new Integer(i);
    }
```

其中IntegerCache类的实现为：

```
private static class IntegerCache {
        static final int high;
        static final Integer cache[];

        static {
            final int low = -128;

            // high value may be configured by property
            int h = 127;
            if (integerCacheHighPropValue != null) {
                // Use Long.decode here to avoid invoking methods that
                // require Integer's autoboxing cache to be initialized
                int i = Long.decode(integerCacheHighPropValue).intValue();
                i = Math.max(i, 127);
                // Maximum array size is Integer.MAX_VALUE
                h = Math.min(i, Integer.MAX_VALUE - -low);
            }
            high = h;

            cache = new Integer[(high - low) + 1];
            int j = low;
            for(int k = 0; k < cache.length; k++)
                cache[k] = new Integer(j++);
        }

        private IntegerCache() {}
    }
```

从这2段代码可以看出，在通过valueOf方法创建Integer对象的时候，如果数值在[-128,127]之间，便返回指向IntegerCache.cache中已经存在的对象的引用；否则创建一个新的Integer对象。

上面的代码中i1和i2的数值为100，因此会直接从cache中取已经存在的对象，所以i1和i2指向的是同一个对象，而i3和i4则是分别指向不同的对象。

面试题2：以下代码输出什么？

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

        Double i1 = 100.0;
        Double i2 = 100.0;
        Double i3 = 200.0;
        Double i4 = 200.0;

        System.out.println(i1==i2);
        System.out.println(i3==i4);
    }
}
```

运行结果：

```
false
false
```

原因： 在某个范围内的整型数值的个数是有限的，而浮点数却不是。

7、 重载和重写的区别

重写(Override)

从字面上看，重写就是 重新写一遍的意思。其实就是在子类中把父类本身有的方法重新写一遍。子类继承了父类原有的方法，但有时子类并不想原封不动的继承父类中的某个方法，所以在方法名，参数列表，返回类型(除过子类中方法的返回值是父类中方法返回值的子类时)都相同的情况下， 对方法体进行修改或重写，这就是重写。但要注意子类函数的访问修饰权限不能少于父类的。

```
public class Father {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Son s = new Son();
        s.sayHello();
    }

    public void sayHello() {
        System.out.println("Hello");
    }
}

class Son extends Father{

    @Override
    public void sayHello() {
        // TODO Auto-generated method stub
        System.out.println("hello by ");
    }

}
```

重写 总结:
1.发生在父类与子类之间
2.方法名，参数列表，返回类型（除过子类中方法的返回类型是父类中返回类型的子类）必须相同
3.访问修饰符的限制一定要大于被重写方法的访问修饰符（public>protected>default>private)
4.重写方法一定不能抛出新的检查异常或者比被重写方法申明更加宽泛的检查型异常

重载（Overload）

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、参数个数不同甚至是参数顺序不同**）则视为重载。同时，重载对返回类型没有要求，可以相同也可以不同，但**不能通过返回类型是否相同来判断重载。**

```
public class Father {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Father s = new Father();
        s.sayHello();
        s.sayHello("wintershii");

    }

    public void sayHello() {
        System.out.println("Hello");
    }

    public void sayHello(String name) {
        System.out.println("Hello" + " " + name);
    }
}
```

重载 总结:
1.重载Overload是一个类中多态性的一种表现
2.重载要求同名方法的参数列表不同(参数类型，参数个数甚至是参数顺序)
3.重载的时候，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准

8、 equals与==的区别

==：

== 比较的是变量(栈)内存中存放的对象的(堆)内存地址，用来判断两个对象的地址是否相同，即是否是指相同一个对象。比较的是真正意义上的指针操作。

1、比较的是操作符两端的操作数是否是同一个对象。
2、两边的操作数必须是同一类型的（可以是父子类之间）才能编译通过。
3、比较的是地址，如果是具体的阿拉伯数字的比较，值相等则为true，如：
int a=10 与 long b=10L 与 double c=10.0都是相同的（为true），因为他们都指向地址为10的堆。

equals:

equals用来比较的是两个对象的内容是否相等，由于所有的类都是继承自java.lang.Object类的，所以适用于所有对象，如果没有对该方法进行覆盖的话，调用的仍然是Object类中的方法，而Object中的equals方法返回的却是==的判断。

总结：

所有比较是否相等时，都是用equals 并且在对常量相比较时，把常量写在前面，因为使用object的equals object可能为null 则空指针

在阿里的代码规范中只使用equals ，阿里插件默认会识别，并可以快速修改，推荐安装阿里插件来排查老代码使用“==”，替换成equals

9、 Hashcode的作用

java的集合有两类，一类是List，还有一类是Set。前者有序可重复，后者无序不重复。当我们在set中插入的时候怎么判断是否已经存在该元素呢，可以通过equals方法。但是如果元素太多，用这样的方法就会比较满。

于是有人发明了哈希算法来提高集合中查找元素的效率。 这种方式将集合分成若干个存储区域，每个对象可以计算出一个哈希码，可以将哈希码分组，每组分别对应某个存储区域，根据一个对象的哈希码就可以确定该对象应该存储的那个区域。

hashCode方法可以这样理解：它返回的就是根据对象的内存地址换算出的一个值。这样一来，当集合要添加新的元素时，先调用这个元素的hashCode方法，就一下子能定位到它应该放置的物理位置上。如果这个位置上没有元素，它就可以直接存储在这个位置上，不用再进行任何比较了；如果这个位置上已经有元素了，就调用它的equals方法与新元素进行比较，相同的话就不存了，不相同就散列其它的地址。这样一来实际调用equals方法的次数就大大降低了，几乎只需要一两次。

10、String、String StringBuffer 和 StringBuilder 的区别是什么?

String是只读字符串，它并不是基本数据类型，而是一个对象。从底层源码来看是一个final类型的字符数组，所引用的字符串不能被改变，一经定义，无法再增删改。每次对String的操作都会生成新的String对象。

```
private final char value[];
```

每次+操作： 隐式在堆上new了一个跟原字符串相同的StringBuilder对象，再调用append方法 拼接+后面的字符。

StringBuffer和StringBuilder他们两都继承了AbstractStringBuilder抽象类，从AbstractStringBuilder抽象类中我们可以看到

```
/**
     * The value is used for character storage.
     */
    char[] value;
```

他们的底层都是可变的字符数组，所以在进行频繁的字符串操作时，建议使用StringBuffer和StringBuilder来进行操作。 另外StringBuffer 对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。StringBuilder 并没有对方法进行加同步锁，所以是非线程安全的。

11、ArrayList和linkedList的区别

Array（数组）是基于索引(index)的数据结构，它使用索引在数组中搜索和读取数据是很快的。

Array获取数据的时间复杂度是O(1),但是要删除数据却是开销很大，因为这需要重排数组中的所有数据,(因为删除数据以后, 需要把后面所有的数据前移)

缺点: 数组初始化必须指定初始化的长度, 否则报错

例如:

```
int[] a = new int[4];//推介使用int[] 这种方式初始化

int c[] = {23,43,56,78};//长度：4，索引范围：[0,3]
```

List—是一个有序的集合，可以包含重复的元素，提供了按索引访问的方式，它继承Collection。

List有两个重要的实现类：ArrayList和LinkedList

ArrayList: 可以看作是能够自动增长容量的数组

ArrayList的toArray方法返回一个数组

ArrayList的asList方法返回一个列表

ArrayList底层的实现是Array, 数组扩容实现

LinkList是一个双链表,在添加和删除元素时具有比ArrayList更好的性能.但在get与set方面弱于ArrayList.当然,这些对比都是指数据量很大或者操作很频繁。

12、 HashMap和HashTable的区别

1、两者父类不同

HashMap是继承自AbstractMap类，而Hashtable是继承自Dictionary类。不过它们都实现了同时实现了map、Cloneable（可复制）、Serializable（可序列化）这三个接口。

2、对外提供的接口不同

Hashtable比HashMap多提供了elments() 和contains() 两个方法。
elments() 方法继承自Hashtable的父类Dictionnary。elements() 方法用于返回此Hashtable中的value的枚举。

contains()方法判断该Hashtable是否包含传入的value。它的作用与containsValue()一致。事实上，contansValue() 就只是调用了一下contains() 方法。

3、对null的支持不同

Hashtable：key和value都不能为null。

HashMap：key可以为null，但是这样的key只能有一个，因为必须保证key的唯一性；可以有多个key值对应的value为null。

4、安全性不同

HashMap是线程不安全的，在多线程并发的环境下，可能会产生死锁等问题，因此需要开发人员自己处理多线程的安全问题。

Hashtable是线程安全的，它的每个方法上都有synchronized 关键字，因此可直接用于多线程中。

虽然HashMap是线程不安全的，但是它的效率远远高于Hashtable，这样设计是合理的，因为大部分的使用场景都是单线程。当需要多线程操作的时候可以使用线程安全的ConcurrentHashMap。

ConcurrentHashMap虽然也是线程安全的，但是它的效率比Hashtable要高好多倍。因为ConcurrentHashMap使用了分段锁，并不对整个数据进行锁定。

5、初始容量大小和每次扩充容量大小不同

6、计算hash值的方法不同

13、 Collection包结构，与Collections的区别

Collection是集合类的上级接口，子接口有 Set、List、LinkedList、ArrayList、Vector、Stack、Set；

Collections是集合类的一个帮助类， 它包含有各种有关集合操作的静态多态方法，用于实现对各种集合的搜索、排序、线程安全化等操作。此类不能实例化，就像一个工具类，服务于Java的Collection框架。

14、 Java的四种引用，强弱软虚

- 强引用

  强引用是平常中使用最多的引用，强引用在程序内存不足（OOM）的时候也不会被回收，使用方式：

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

- 软引用

  软引用在程序内存不足时，会被回收，使用方式：

  ```
  // 注意：wrf这个引用也是强引用，它是指向SoftReference这个对象的，
  // 这里的软引用指的是指向new String("str")的引用，也就是SoftReference类中T
  SoftReference<String> wrf = new SoftReference<String>(new String("str"));
  ```

  可用场景： 创建缓存的时候，创建的对象放进缓存中，当内存不足时，JVM就会回收早先创建的对象。

- 弱引用

  弱引用就是只要JVM垃圾回收器发现了它，就会将之回收，使用方式：

  ```
  WeakReference<String> wrf = new WeakReference<String>(str);
  ```

  可用场景： Java源码中的`java.util.WeakHashMap`中的`key`就是使用弱引用，我的理解就是，一旦我不需要某个引用，JVM会自动帮我处理它，这样我就不需要做其它操作。

- 虚引用

  虚引用的回收机制跟弱引用差不多，但是它被回收之前，会被放入`ReferenceQueue`中。注意哦，其它引用是被JVM回收后才被传入`ReferenceQueue`中的。由于这个机制，所以虚引用大多被用于引用销毁前的处理工作。还有就是，虚引用创建的时候，必须带有`ReferenceQueue`，使用例子：

  ```
  PhantomReference<String> prf = new PhantomReference<String>(new
  String("str"), new ReferenceQueue<>());
  ```

  可用场景： 对象销毁前的一些操作，比如说资源释放等。**`Object.finalize()`虽然也可以做这类动作，但是这个方式即不安全又低效

上诉所说的几类引用，都是指对象本身的引用，而不是指`Reference`的四个子类的引用(`SoftReference`等)。

15、 泛型常用特点

泛型是Java SE 1.5之后的特性， 《Java 核心技术》中对泛型的定义是：

> “泛型”意味着编写的代码可以被不同类型的对象所重用。

“泛型”，顾名思义，“泛指的类型”。我们提供了泛指的概念，但具体执行的时候却可以有具体的规则来约束，比如我们用的非常多的ArrayList就是个泛型类，ArrayList作为集合可以存放各种元素，如Integer, String，自定义的各种类型等，但在我们使用的时候通过具体的规则来约束，如我们可以约束集合中只存放Integer类型的元素，如

```
List<Integer> iniData = new ArrayList<>()
```

使用泛型的好处?

以集合来举例，使用泛型的好处是我们不必因为添加元素类型的不同而定义不同类型的集合，如整型集合类，浮点型集合类，字符串集合类，我们可以定义一个集合来存放整型、浮点型，字符串型数据，而这并不是最重要的，因为我们只要把底层存储设置了Object即可，添加的数据全部都可向上转型为Object。 更重要的是我们可以通过规则按照自己的想法控制存储的数据类型。

16、Java创建对象有几种方式?

java中提供了以下四种创建对象的方式:

- new创建新对象
- 通过反射机制
- 采用clone机制
- 通过序列化机制

17、有没有可能两个不相等的对象有相同的hashcode

有可能.在产生hash冲突时,两个不相等的对象就会有相同的 hashcode 值.当hash冲突产生时,一般有以下几种方式来处理: