Java基础总结

*面向对象的特征

- 封装：把描述的一个对象的属性和行为封装在一个模块中，属性用字段定义，行为用方法定义。
- 继承：子类可以继承父类的特征和行为，子类也可以对父类中的行为重新定义(重写)。缺点就是提高了代码之间的耦合性。
- 多态：是指程序中定义引用变量所指向的具体类型和通过该引用发出的方法调用在编程的时候并不确定，而是在程序运行期间才能确定。比如说：向上转型。方法的覆盖和重载体现了多态性。
- 抽象：现实生活中的对象抽象成类。数据抽象(类的属性)和过程抽象(类的方法)

*public protected private

```
public：可以被所有其他类所访问

private：只能被自己访问和修改

protected：自身、子类及同一个包中类可以访问

default：同一包中的类可以访问，声明时没有加修饰符，认为是friendly。
```

*包装类型

- 目的是让基本类型也具有对象的特征。
- 例如int->Integer，long->Long
- 自动装箱和自动拆箱
  - 自动装箱：Integer x = 10；实际上执行了Integer x = Integer.valueOf(10)

    实际上就是将基本类型转换成对象的包装类型
  - 自动拆箱：int a = x； 实际上执行了int a = x.intValue();

    实际上就是将包装类型转化成基本类型

==和eauals的区别

- ==是判断两个引用是否指向的同一个实例对象即同一块内存空间。
- eauals也是用来比较两个引用是否指向同一块内存空间，但是String,Date,Integer重写了equals方法，这几个类在使用eauqls时，比较的时内存空间中的内容是否一样。

*String和StringBuffer和StringBuilder

- 底层实现上：String采用char数组，用final修饰，不可变，但是StringBuffer和StringBuilder都是采用可变char数组来实现的。
- 线程安全上:由于String是不可变的，因此可以认为string是线程安全的，StringBuffer是线程安全的，加了同步锁，StringBuilder是线程不安全的。

- 读取字符串的速度：StringBuilder>StringBuffer>String(String在修改对象的时候，会产生一个新的对象，所以会比较慢)
- 使用范围：
    - 操作较少的数据：String
    - 单线程操作大量数据：StringBuilder
    - 多线程操作大量数据：StringBuffer

*Java中的集合

- HashMap，HashTable的区别
    - 就线程安全而言，HashMap不是线程安全的，HashTable是线程安全的。
    - HashMap可以把null当作key或者value，但是HashTable是不行的。
    - 初始容量大小：HashMap初始值为16，每次扩容都是原来的2倍，HsahTable初始值是11，每次扩容都是原来的2*n+1.
    - 底层结构：HashMap是散列表+链表+红黑树，HashTable是散列表+链表。
    - 解决冲突的方法：拉链法。
- HashMap
    - 初始容量为16，以后每次扩容都是原来容量的2倍，只是因为当key的hashcode的值大于数组的长度的时候，需要对hashcode操作进行取余操作，即hashcode()%len，但是该操作可以使用位运算来实现即hashcode()&(len-1)，但是如果采用该运算式的话，必须保证len是2的次幂。
    - 当链表的长度大于8的时候，会转换成红黑树，进而提高数据的查找速度。
    - 在并发的情况下，HashMap可能会发生死循环问题，原因就是在对HashMap进行扩容的时候，链表中节点的顺序会发生变化。即原来的顺序被另一个线程a颠倒了，而被挂起的线程b唤醒之后拿着扩容前的节点和顺序继续完成第一次循环后，有遵循了a线程修改之后的i按发表结点的顺序，最终形成了环。
    - 如何让HashMap同步： Map xMap=Collections.synchronizedMap(m)//m是HashMap对象实例。
- CurrentHashMap
    - 采用分段的思想，加锁的范围并不是整个table数组整体，而是指对其中某一段进行加锁，那么在某段加锁的时候，其他线程可以访问没有加锁的段数据，
- 线程安全的集合：Vector,HashTable,CurrentHashMap,

*拷贝文件的工具类使用字节流还是字符流：字节流

- 字节流：传递的是字节(二进制)
- 字符流：传递的是字符

*永久代和元空间

- 在java 1.8中，已经取消了永久代，取而代之的是元空间.元空间不在虚拟机中，而是使用的是本地内存。
- 原因
    - 字符串在永久代中，容易出现性能问题和内存溢出
    - 类及方法的信息等比较难以确定其大小，因此对于永久代的大小这顶很困难。
    - 永久代会为GC带来不必要的都咋读，并且回收效率偏低

*String s=“hello”和 String s=new String("hello")的区别

- 对于String s = “hello”来说，如果字符串常量池中存在"hello"的引用，那么就直接返回该引用，如果没有，那就创建一个“hello”的字符串对象，并将该字符串的引用放到字符串常量中

- 对于String s = new String("hello")来说，可能创建一个对象，也可能创建两个对象，当字符串常量池中没有"hello"时，会创建一个字符串对象，然后将对象的引用放入到字符串常量中，但是对于new来说，无论字符串常量池中是否含有指向"hello"的引用，它都会在堆中创建一个"hello"的对象，并将引用传给s。

*switch可以使用的数据类型

- char(Character)，byte(Byte)，short(Short)，int(Integer)，String，enum

*Integer的一些想法

-
```
Integer x1 = 100;
Integer x2 = 100;
System.out.println(x1==x2);

Integer a1 = 128;
Integer a2 = 128;
System.out.println(a1==a2);

int i=100;
System.out.println(i==x1);

/*
true
false
true
*/
//当用Integer来去定义一个引用的时候，当值小于127且大于-128的时候，Integer类型实在内存栈中创建值得，所有得引用会指向同一个栈中得元素。当值大于127得时候，那么每个Integer对象就会用new去实例化一个对象，并将对象的引用返回给变量。
//i==x1为true是因为经过反编译可以看到i==x1.intValue()所以为true
```

*ceil,floor,round

- ceil：向上取整
- floor：向下取整
- round：Math.round(x)==Math.floor(x+0.5)

```
System.out.println(Math.ceil(1.2));
System.out.println(Math.ceil(-1.2));

System.out.println(Math.floor(11.4));
System.out.println(Math.floor(-11.4));

System.out.println(Math.round(11.4));
System.out.println(Math.round(-11.4));

/*
2.0
-1.0
```

```
11.0
-12.0
11
-11   //-11.4+0.5=-10.9向上取整为-11

*/
```

*抽象类和接口的区别

- 抽象类中可以含有普通成员变量，接口不行
- 抽象类可以含有构造方法，接口不行
- 抽象类可以包含静态方法，但是接口不行
- 一个类继承抽象类，如果没有实现抽象类中的抽象方法，必须用abstruct修饰，也就是说抽象类，一个类继承接口的时候，必须重写接口中的方法。
- 一个类可以实现多个接口，但只能继承一个抽象类

*List，Set，Map是否继承自Collection接口

- List，Set是继承了Collection接口的，Map是没有继承Collection接口。
- 三者存取元素时各有什么特点?
    - List和Set具有相似性，他们都是单列元素的集合。Set中不允许有重复的元素，List有先后顺序，主要是List是由链表构成，Map时双列的集合，key-value形式，key值唯一，但是可以由多个key值对应一个value值。