Object类中的方法，按照用途可以分为以下几种：

构造函数

1.hashCode() 和 equals() 函数用来判断对象是否相同,

2.wait(), wait(long), wait(long,int), notify(), notifyAll() 线程等待和唤醒。

3.toString() 和 getClass()

4.clone()

5.finalize() 用于在垃圾回收

clone：保护方法，实现对象的浅复制。需实现Cloneable接口才可以调用，否则报CloneNotSupportedException。

getClass方法：final方法，获得运行时类的类型。

toString方法：返回一个String对象，用来标识自己。用的多，一般子类会重写。

finalize方法：用于释放资源。因为无法确定该方法什么时候被调用，很少使用。

equals方法：一般 equals 和 == 是不一样的，但是在 Object 中两者是一样的。子类一般都要重写这个方法。

hashCode方法：重写了equals方法一般都要重写hashCode方法。一般必须满足 obj1.equals(obj2) == true。 可以推出 obj1.hashCode() == obj2.hashCode()， 但是 hashCode 相等不一定就满足 equals。 不过为了提高效率，应该尽量使上面两个条件接近等价。

**wait 方法：**就是使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。wait() 方法一直等待，直到获得锁或者被中断。wait(long timeout) 设定一个超时间隔，如果在规定时间内没有获得锁就返回。

调用该方法后当前线程进入睡眠状态，直到以下事件发生。

（1）其他线程调用了该对象的 notify 方法。

（2）其他线程调用了该对象的 notifyAll 方法。

（3）其他线程调用了 interrupt 中断该线程。

（4）时间间隔到了。

此时该线程就可以被调度了，如果是被中断的话就抛出一个InterruptedException 异常。

**2.8 notify 方法**

该方法唤醒在该对象上等待的某个线程。

**2.9 notifyAll 方法**

该方法唤醒在该对象上等待的所有线程。

**接口和抽象类的区别和联系：**

[**https://blog.csdn.net/weixin\_36759405/article/details/82774558**](https://blog.csdn.net/weixin_36759405/article/details/82774558)

|  |  |  |
| --- | --- | --- |
| 参数 | 抽象类 | 接口 |
| 默认的方法实现 | 可以有默认的方法实现 | 完全抽象的，不存在方法实现 |
| 实现 | extends继承，若子类不是抽象类，则需实现所有抽象类中的方法实现 | implements实现接口所有声明方法 |
| 构造器 | 可以有构造器 | 不能有构造器 |
| 与正常Java类的区别 | 除不能实例化之外，与普通类无异 | 完全不同的类型 |
| 访问修饰符 | 抽象方法可以有public和protected修饰符 | 默认public，不可使用其他修饰符。只能有static，final变量 |
| main方法 | 可以有main方法，可以运行 | 没有main方法，不能运行 |
| 多继承 | 继承一个类实现多个接口 | 接口只可以继承一个或多个其他接口，如**public** **interface** InterfaceA **extends** InterfaceB，InterfaceC |
| 速度 | 比接口速度快 | 接口稍微慢，因为需要时间去寻找在类中的实现方法 |
| 添加新方法 | 添加新方法可以提供默认实现，这样不需要改变其他代码。 | 添加新方法需要改变所有该接口的实现类 |

**JDK8里面接口的新特性：**

接口不在只能使用抽象的方法了，可以添加具体方法，可以是静态的也可以是非静态的，非静态的方法前面要使用关键字defualt。

静态方法只能被抽口引用，非静态的方法只能被implement类实例化的对象引用。所以当2个接口的静态方法名字相同时，不会存在问题，只能被接口的类名. 引用。而非静态方法就会出现问题，因为对象不知道使用引用哪个接口的方法，就会造成编译失败。

**public** **interface** InterfaceA {

void testExcel(File file);//最常见的默认的public abstract方法

**default** **void** test() {

System.***out***.println("ss");

}

**static** **void** test2() {

System.***out***.println("ss");

}

……

}default关键字只能在接口中使用（以及用在switch语句的default分支），不能用在抽象类中；接口默认方法不能覆写Object类的equals、hashCode和toString方法；接口中的静态方法必须是public的，public修饰符可以省略，static修饰符不能省略。

**为什么要有默认方法?**

在Java 8之前，接口与其实现类之间的耦合度过高（tightly coupled），当需要为一个接口添加方法时，所有的实现类都必须随之修改。默认方法可以为接口添加新的方法，而不会破坏已有的接口的实现。这在lambda表达式作为Java 8语言的重要特性出现之际，为升级旧接口且保持向后兼容（backward compatibility）提供了途径。

接口默认方法的继承分三种情况（分别对应下面的InterfaceB接口、InterfaceC接口和InterfaceD接口）：

不覆写默认方法，直接从父接口中获取方法的默认实现。

覆写默认方法，这跟类与类之间的覆写规则相类似。

覆写默认方法并将它重新声明为抽象方法，这样新接口的子类必须再次覆写并实现这个抽象方法。i

nterface InterfaceD extends InterfaceA { @Override void foo(); }

接口中的变量总是需要定义为“public static final 接口名称”，但可以不包含这些修饰符，编译器默认就是这样，显示的包含修饰符主要是为了程序清晰。

类的构造器方法不能够有修饰符、返回类型和throws子句。

Java 8的一个大亮点是引入Lambda表达式，使用它设计的代码会更加简洁。当开发者在编写Lambda表达式时，也会随之被编译成一个函数式接口。下面这个例子就是使用Lambda语法来代替匿名的内部类，代码不仅简洁，而且还可读

lambda表达式：

https://www.cnblogs.com/knowledgesea/p/3163725.html