# cloneable接口的作用

cloneable其实就是一个标记接口，只有实现这个接口后，然后在类中重写Object中的clone方法，然后通过类调用clone方法才能克隆成功，如果不实现这个接口，则会抛出CloneNotSupportedException(克隆不被支持)异常。Object中clone方法：

Protected native Object clone() throws CloneNotSupportedException;

这里有一个疑问，Object中的clone方法是一个空的方法，那么他是如何判断类是否实现了cloneable接口呢？

原因在于这个方法中有一个native关键字修饰。native修饰的方法都是空的方法，但是这些方法都是有实现体的（这里也就间接说明了native关键字不能与abstract同时使用。因为abstract修饰的方法与java的接口中的方法类似，他显式的说明了修饰的方法，在当前是没有实现体的，abstract的方法的实现体都由子类重写），只不过native方法调用的实现体，都是非java代码编写的（例如：调用的是在jvm中编写的C的接口），每一个native方法在jvm中都有一个同名的实现体，native方法在逻辑上的判断都是由实现体实现的，另外这种native修饰的方法对返回类型，异常控制等都没有约束。

也就是说只是引用变量在深拷贝和浅拷贝中有不同。因为普通类型的变量是通过值传递但是引用变量的传递是内存的传递。

什么是深拷贝，什么是潜拷贝？

**浅拷贝**（Shallow Copy）：①对于数据类型是基本数据类型的成员变量，浅拷贝会直接进行值传递，也就是将该属性值复制一份给新的对象。因为是两份不同的数据，所以对其中一个对象的该成员变量值进行修改，不会影响另一个对象拷贝得到的数据。②对于数据类型是引用数据类型的成员变量，比如说成员变量是某个数组、某个类的对象等，那么浅拷贝会进行引用传递，也就是只是将该成员变量的引用值（内存地址）复制一份给新的对象。因为实际上两个对象的该成员变量都指向同一个实例。在这种情况下，在一个对象中修改该成员变量会影响到另一个对象的该成员变量值。

深拷贝：首先介绍对象图的概念。设想一下，一个类有一个对象，其成员变量中又有一个对象，该对象指向另一个对象，另一个对象又指向另一个对象，直到一个确定的实例。这就形成了对象图。那么，对于深拷贝来说，不仅要复制对象的所有基本数据类型的成员变量值，还要为所有引用数据类型的成员变量申请存储空间，并复制每个引用数据类型成员变量所引用的对象，直到该对象可达的所有对象。也就是说，对象进行深拷贝要对整个对象图进行拷贝！

简单地说，深拷贝对引用数据类型的成员变量的对象图中所有的对象都开辟了内存空间；而浅拷贝只是传递地址指向，新的对象并没有对引用数据类型创建内存空间

