1 == 符号的功能

- 对于基本类型(byte,short,char,int,long,float,double,boolean)：比较的是值
- 对于引用类型(String)：比较的是地址

2 == 符号的优点

- 不会抛出 NullPointerException（null != 任何对象）
- 在编译期检测类型兼容性（== 符号两边的**对象类型必须一样**，不然会编译出错）
- 速度很快（因为是比较地址）

3 == 符号的使用场景

- 比较基本类型时
- 可以直接比较地址的情况
    - 枚举类就可以直接使用 == 符号，因为都是单例(只有地址一样，才是同一实例)
    - 其它涉及到单例的情况（如静态工厂）

4 equals() 方法介绍

- equals() 方法是 Object 类中的方法，默认实现就是 == ：即比较地址
- 可以覆盖Object 类 的equals() 方法，来实现自己的相等逻辑

4.1 equals() 方法使用场景

- 大量用于散列表中（HashMap 源码中）
- 需要进行对象比较的时候（不想仅仅比较地址）

4.2 谨慎的重写 equals() 方法

- 在不需要重写 equals() 方法的情况下，尽量不要重写 equals() 方法，已避免出现问题
- 不需要重写 equals() 方法的情况
    - 类的每个实例本质上都是唯一的（单例、枚举）

    - 类是私有的或是包级私有的，可以确定它的 equals() 方法永远不会被调用（这句没懂）
    - 不需要进行类之间比较的

4.3 重写 equals() 方法请遵守通用约定

- **自反性：** 对于任何非 null 的引用值 x，x.equals(x) 必须为 true
- **对称性：** 对于任何非 null 的引用值 x,y，当且仅当 y.equals(x) 为 true 时，x.equals(y) 必须为 true
- **一致性：** 对于任何非 null 的引用值 x,y，只要 equals 的比较操作在对象中所用的信息没有被修改，多次调用 x.equals(y) 的返回值必须是一致的
- **传递性：** 对于任何非 null 的引用值 x,y,z，如果 x.equals(y) 为 true，并且 y.equals(z) 也为 true，那么 x.equals(z) 必须也为 true
- **非空性：** 对于任何非 null 的引用值 x，x.equals(null) 必须返回 false

4.4 什么时候要重写 equals() 方法

- 在一个对象要用于散列表中时(即作为 map 的key时)，需要重写 equals() 方法，不然此 map 毫无意义（同样，当用作散列表时，也必须重写 hashcode() 方法）
- 想自定义对象比较的逻辑时（而不是比较地址）

4.5 覆盖 equals() 方法时，必须覆盖 hashCode() 方法

- 这个设计到了 hashCode() 方法的规定

4.6 重写 equals()、hashCode() 方法时，请使用集成环境自动生成

- 自动生成后，可以进行微调，这样可以避免完全自己手打可能出现的各种各样的问题

4.7 equals() 方法的优缺点

- 优点：可以自定义逻辑，比较灵活
- 缺点：当调用对象为 null 时，会抛 NullPointerException 异常

参考

比较java枚举成员使用equal还是== 浅谈java中的"=="和eqals区别 Java hashCode() 和 equals()的若干问题解答
Effective Java 中文第二版