链滴

transient 关键字

作者：mihone

原文链接：https://ld246.com/article/1584208841314

来源网站：链滴

一、什么是 transient

简而言之，被 transient标记的变量在序列化时会被忽略，当然没有序列化就没有反序列化

看看下面的例子来加深理解：

```
@Data
class Employee implements Serializable
{
   private String          firstName;
   private String          lastName;
   private transient String password;
}

//序列化对象
try
{
    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("test.txt"));
    Employee emp = new Employee();
    emp.setFirstName("john");
    emp.setLastName("jack");
    emp.setPassword("password");
    oos.writeObject(emp);
    oos.close();
} catch (Exception e)
{
    System.out.println(e);
}

// 反序列化
try
{
    ObjectInputStream ooi = new ObjectInputStream(new FileInputStream("test.txt"));
    Employee readEmpInfo = (Employee) ooi.readObject();
    System.out.println(readEmpInfo.getFirstName());
    System.out.println(readEmpInfo.getLastName());
    System.out.println(readEmpInfo.getPassword());
    // john
    // jack
    // null
    ooi.close();
} catch (Exception e)
{
    System.out.println(e);
}
```

可以看到被 transient修饰的属性反序列化为null。

二、transient应用场景

1. 对象的某个属性的值，需要根据其他属性来计算的时候，应该把这个属性加上 transient。

这种属性应该是每次都用程序计算，而不是用序列化来持久化。比如说基于时间戳的值，像年龄，或

需要保存一段时间，这段时间是某个时间戳和当前时间戳的时间段。这种情况就不应该将这种属性值列化。

2. 涉及隐私以及安全问题的属性不应该序列化。

3. 本身就没有实现序列化接口的属性，如果被序列化会抛出 NotSerializableException异常。

4. 逻辑道理上不需要序列化。比如说类里面有个 Logger对象，但是 Logger只是用来写日志，不要保留这个对象的状态。再比如线程对象，线程对象保留是某个时间点的线程状态，没有持久化的意。

三、transient和 final组合

一个属性同时被 transient和 final修饰，如果这个属性属于基本数据类型和String，那么序列化的时候ransient会被会忽略掉。因为会被 JVM认为是常量。

其他引用类型(包括基本数据类型的包装类)，即使被final修饰，transient也会生效

四、实际应用举例--HashMap

```
// HashMap中使用 transient标记的字段
transient Entry          table[];
transient int            size;
transient int            modCount;
transient int            hashSeed;
private transient Set    entrySet;
```

可以看出 HashMap的key-value相关的存储都是用transient修饰的，也就是说序列化的时候不会被列化。

```
private void readObject(java.io.ObjectInputStream s)
    throws IOException, ClassNotFoundException {
    // Read in the threshold (ignored), loadfactor, and any hidden stuff
    s.defaultReadObject();
    reinitialize();
    if (loadFactor <= 0 || Float.isNaN(loadFactor))
        throw new InvalidObjectException("Illegal load factor: " +
                                         loadFactor);
    s.readInt();                // Read and ignore number of buckets
    int mappings = s.readInt(); // Read number of mappings (size)
    if (mappings < 0)
        throw new InvalidObjectException("Illegal mappings count: " +
                                         mappings);
    else if (mappings > 0) { // (if zero, use defaults)
        // Size the table using given load factor only if within
        // range of 0.25...4.0
        float lf = Math.min(Math.max(0.25f, loadFactor), 4.0f);
        float fc = (float)mappings / lf + 1.0f;
        int cap = ((fc < DEFAULT_INITIAL_CAPACITY) ?
                   DEFAULT_INITIAL_CAPACITY :
                   (fc >= MAXIMUM_CAPACITY) ?
                   MAXIMUM_CAPACITY :
                   tableSizeFor((int)fc));
        float ft = (float)cap * lf;
```

```
        threshold = ((cap < MAXIMUM_CAPACITY && ft < MAXIMUM_CAPACITY) ?
                     (int)ft : Integer.MAX_VALUE);

        // Check Map.Entry[].class since it's the nearest public type to
        // what we're actually creating.
        SharedSecrets.getJavaOISAccess().checkArray(s, Map.Entry[].class, cap);
        @SuppressWarnings({"rawtypes","unchecked"})
        Node<K,V>[] tab = (Node<K,V>[])new Node[cap];
        table = tab;

        // Read the keys and values, and put the mappings in the HashMap
        for (int i = 0; i < mappings; i++) {
            @SuppressWarnings("unchecked")
            K key = (K) s.readObject();
            @SuppressWarnings("unchecked")
            V value = (V) s.readObject();
            putVal(hash(key), key, value, false, false);
        }
    }
}
```

HashMap反序列化后，会重新hash计算把key-value存进去。