转载请注明链接:https://blog.csdn.net/feather_wch/article/details/50402042

int和Integer

版本：2018/9/7-1(14:14)

问题汇总

1. int和Integer有什么区别?
2. Java的原始数据类型是什么?
3. 为什么需要原始数据类型？Java的对象不就可以了吗?
4. 包装类和原始数据类型如何实现线程安全计数器?
5. 原始类型线程安全吗?
6. 【☆】需要线程安全时，该怎么办?
   > 建议使用 `AtomicLong、AtomicInteger`
7. float、double有什么线程安全上的隐患?
8. 原始数据类型的弊端
9. Java中的泛型作用于运行时还是编译阶段？java泛型属于伪泛型吗?
10. 原始数据类型数组和对象数组的区别
11. 谨慎使用基本数据类型来处理货币存储。
12. BigDecimal有什么用?
13. 基本数据类型要注意隐式转换的问题
14. Integer是什么?
15. Integer相比于int的好处?
16. Integer数组的缺点

17. Integer的值缓存是什么?
18. Integer的缓存机制是什么?
19. Integer的值缓存范围
20. Boolean、Short、Byte、Character等包装类的缓存机制的范围?
21. 如何增加Integer的缓存范围?
22. Integer的构造方法
23. String是不可变的, 但是Integer是可变的?
24. 为什么Integer等包装类采用immutable的设计方法?
25. IntegerCache是什么?
26. 虚拟机的缓存上限的参数是如何生效的?
27. String的缓存机制和Integer的缓存机制的相同和不同?
28. 什么是自动装箱和自动拆箱?
29. Java具有编译阶段、运行时，自动装箱/拆箱发生在什么阶段?
30. valueOf的缓存机制，对于boxing有效吗?
31. 需要去避免自动装箱/拆箱吗?
32. 如何计算对象的大小?

int

1、int和Integer有什么区别?

> 1. int是原始数据类型
> 2. Integer是引用类型
> 3. int不是对象
> 4. Integer是对象，内部有一个int字段来存储数据

2、Java的原始数据类型

> 1. 原始数据类型(Primitive Types)一共有8种。
> 2. Java语言中一切都是对象，但是 原始数据类型 除外
> 3. boolean、byte、char、short、int、long、float、double

3、为什么需要原始数据类型？Java的对象不就可以了吗?

> 1. 原始数据类型、 数组 、 本地代码实现 在性能方面都有巨大的优势
> 2. 包装类、集合(ArrayList)等，性能就会较低。
> 3. 但是开发效率作为产品开发的重要因素之一，并不会过多的去追求性能上的极致。

4、包装类和原始数据类型实现线程安全计数器

> 1-原始数据类型

```
public class Counter {
    private volatile long counter;
    private static final AtomicLongFieldUpdater<Counter> updater
            = AtomicLongFieldUpdater.newUpdater(Counter.class, "counter");
    public void increase(){
        updater.incrementAndGet(this);
    }
}
```

> 2-包装类

```
public class Counter {
    private final AtomicLong counter = new AtomicLong();
    public void increase(){
        counter.incrementAndGet();
    }
}
```

5、原始类型线程安全吗?

> 1. 不安全，建议使用 `AtomicLong、AtomicInteger`
> 2. float、double因为较宽，可能会出现程序读取到只更新了一半数据位的float和double

6、原始数据类型的弊端

> 1. 无法和泛型进行配合使用，必须要使用包装类。
> 2. 无法高效的表达书，也无法表达复杂的数据结构，如Vector、tuple

7、Java中的泛型作用于运行时还是编译阶段？java泛型属于伪泛型吗?

> 1. java泛型属于伪泛型。
> 2. 处于编译阶段进行转换，而不是运行时，因此必须要是Object。

8、原始数据类型数组和对象数组的区别

> 1. 原始数据类型数组，数据在内存上是连续的。
> 2. 对象数组，对象在堆上存储，数组中的对象分散在堆中，无法利用CPU的缓存机制。

9、谨慎使用基本数据类型来处理货币存储。

> 1. 在有“毛、分”等小数据时，double会具有误差
> 2. 常采用 `BigDecimal` 、整型解决该问题。

10、BigDecimal有什么用?

> 1. 在需要超大数的计算器等场景，double的16位有效数字就太小了，而且有误差。
> 2. `BigDecimal` 支持超过20位的精度，并提供了加减乘除等API。

11、基本数据类型要注意隐式转换的问题

> 1-需要显式指出数据的类型，防止越界。

```
// 1. 下面的数值错误！！！
long result = 1234567890 * 24 * 365;

// 2. 指明为long，才会避免越界。如下：
long result = 1234567890L * 24 * 365;
```

Integer

1、Integer是什么?

> 1. 是int对应的包装类
> 2. 内部有一个int字段来存储数据
> 3. 提供了基本操作：数学运算、int和String间的转换等。

2、Integer相比于int的好处?

> 1. 常用于Bean
> 2. int的初值是0
> 3. Integer的初值是null
> 4. 采用int会导致字段本来没有数据，但是却变成了0，存在二义性。

3、Integer数组的缺点

> 1. Integer在内存中分散
> 2. 无法利用CPU的缓存机制。
> 3. 每次计算需要先找到目标内存，才能从内存地址中取出数据。性能要低。

缓存机制

4、Integer的值缓存是什么?

> 1. Java 5中进行改进
> 2. 传统构建Integer的方式是直接new一个对象。
> 3. Java 5开始，新增了静态工厂方法 `valueOf`
> 4. Integer大部分数据操作都集中在较小的范围内，因此Java将 `-128 ~ 127` 的数值进行了缓存。

5、Integer的缓存机制是什么?

> 1. `Integer.valueOf()` 能进行缓存, 获取到Integer。
> 2. `Integer.intValue()` ，取出缓存，获取到Int。

6、Integer的值缓存范围

> `-128 ~ 127`

7、包装类的缓存机制表

包装类	缓存内容	备注
Integer	-128~127	
Boolean	Boolean.TRUE/Boolean.FALSE	
Short	-128~127	
Byte	全部缓存	Byte的数值范围比较少，因此全部缓存，效率也较高。
Character	'\u0000'~'\u007F'	

8、如何增加Integer的缓存范围?

> 通过虚拟机参数 `-XX:AutoBoxCacheMax=N`

Integer的源码

9、Integer的构造方法

> 1. 内部存储的是int字段

```
private final int value;
public Integer(int value) {
    this.value = value;
}
```

10、String是不可变的, 但是Integer是可变的?

> 错误!
>
> 1. Integer内部的value是 `final int` ， 因此也是 `immutable` 类
> 2. Boolean底层也是final的

11、为什么Integer等包装类采用immutable的设计方法?

> 1. 比如在获取系统设置时，使用到了Ineteger.getInteger()方法，获取端口。如果能去修改Ineteger会导致，会让程序具有不稳定性。

IntegerCache

12、IntegerCache是什么?

> 1. Inetegr内部的缓存
> 2. static静态代码块中，缓存了从low(-128)~high(127)的Integer

```
private static class IntegerCache {
    // 缓存的下限
    static final int low = -128;
    // 缓存的上限
    static final int high;
    // 缓存Integer的数组
    static final Integer cache[];
    static {
        int h = 127;
        /**============================================
         * 1、从JVM中参数中获取缓存的上限
         *============================================*/
        String integerCacheHighPropValue =
            sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
        // 2、缓存最大不超过Integer的上限。
        if (integerCacheHighPropValue != null) {
            int i = parseInt(integerCacheHighPropValue);
            i = Math.max(i, 127);
            h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
        }
        high = h;
        /**============================================
         * 2、进行缓存
         *============================================*/
        cache = new Integer[(high - low) + 1];
        int j = low;
        for(int k = 0; k < cache.length; k++)
            cache[k] = new Integer(j++);
        assert IntegerCache.high >= 127;
    }
    private IntegerCache() {}
}
```

13、虚拟机的缓存上限的参数是如何生效的?

> 1. Integer的静态内部类IntegerCache，负责处理Integer的缓存。
> 2. 在IntegerCache的静态代码块中读取了JVM参数

```
String integerCacheHighPropValue =
    sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
```

String缓存机制对比

14、String的缓存机制和Integer的缓存机制的相同和不同?

> 1. 相同处都是进行缓存，避免创建重复的对象。
> 2. 不同在于，String的缓存机制是使用到String时，才将其缓存到常量池。

> 3. Integer的缓存，是在初始化时就缓存了范围 `-128~127` 的Integer。

自动拆箱、装箱

1、什么是自动装箱和自动拆箱?

> 1. boxing和unboxing是在Java 5中提出
> 2. Java可以根据上下文，自动进行转换
> 3. 实际上是一种语法糖。（Java平台）会自动做一些转换。

2、Java具有编译阶段、运行时，自动装箱/拆箱发生在什么阶段?

> 1. boxing是Java自动做一些转换，用于保证不同的写法在 运行时是等价的
> 2. boxing发生在 编译阶段
> 3. 从而保证生成的 字节码 是一致的。

3、valueOf的缓存机制，对于boxing有效吗?

> 1. boxing就是 `javac` 自动转换为了 `Integer.valueOf()` 和 `Integer.intValue()`
> 2. 因此缓存机制是生效的。

4、自动装箱/拆箱的注意点?

> 1. 要避免自动装箱和拆箱。
> 2. 大量的Java对象会产生内存和处理速度上的开销。

知识扩展

1、如何计算对象的大小?

> 1. dump内存后，用memory analyze分析
> 2. 也可以通过jol，jmap，Instrument API进行分析。

参考资料

1. BigDecimal的一些用法
2. java如何获取一个对象的大小