

The diagram illustrates the IEEE 754 single-precision floating-point representation for the decimal values 0.2 and 0.3. Each representation is shown in a yellow box with a 'toggle details' link below it.

0.2: The sign bit is 0. The 8-bit exponent is 11111111 (biased). The 23-bit significand is 00100110011001100110011.

0.3: The sign bit is 0. The 8-bit exponent is 11111110 (biased). The 23-bit significand is 00111111001100110011001.

如前文中所述，很多十进制有限小数转换到二进制后变为无限小数，这与有限的尾数所

许多数值编程语言如原生支持或开源社区提供了十进制小数类。以Java为例，其

```
public class BigDecimal extends Number implements
```

● <code>intval</code>	去除小数点后的所有数字	123456
-----------------------	-------------	--------

- 假如 `BigInteger` 用数组存储时，我们存储的精读将会变得近乎无限大。

左边的日 一个用收便用田左理数左左 八数可以精确地表示我们所用到的数 同样以

```
public class Rational implements Comparable<Rational> {  
    private int num;    // the numerator  
    private int den;    // the denominator  
    public double toDouble() {  
        return (double) num / den;  
    }  
    ...  
}
```

其num, den分别为有理数的分子与分母，较为简单，也不再赘述。

以上两个类虽然可以很好的消除精度的问题，但带来了性能的问题，运算速率大大降低。因此，在实际应用中我们需要做好精度和性能两方面的trade-off来选择合适的实现方式。

References:

[1] 为什么 $0.1 + 0.2 = 0.300000004$ - 面向信仰编程 (draveness.me)

[2] IEEE-754 Floating Point representation explained (bartaz.github.io)