

# 为什么JavaScript中0.1 + 0.2 ≠ 0.3

## 1. IEEE754标准

JavaScript中对于浮点数的表示采用IEEE二进制浮点数算术标准。这个标准定义了表示浮点数的常规值与非规格化值(denormal number)，一些特殊值(infinity)和非数值(NaN), 以及这些数值的浮点运算。另外它还规定了运算结果的近似原则和例外状况。

IEEE754中64位浮点数用如下格式表示



其中S代表符号位，E代表阶码，M代表尾数。对于规格化的表示来说，表示数 =  $(-1)^S \cdot 1.M \cdot 2^{E-1023}$ 。当阶码为0，且尾数不为0时，采用非规格化的表示，表示数 =  $(-1)^S \cdot 0.M \cdot 2^{-1022}$ 。当阶码全为1，尾数全为0时，代表无穷大。当阶码全为1，尾数不为0时，代表NaN。

IEEE754表示的浮点数进行加法时遵循以下步骤：1. 对阶 2. 尾数运算 3. 规格化

## 2. 问题原因

根据IEEE754标准表示，0.1、0.2、0.3 的二进制表示如下表所示。

0.1	0 01111111011 1001100110011001100110011001100110011001100110011010
0.2	0 01111111100 1001100110011001100110011001100110011001100110011010
0.3	0 01111111101 0011001100110011001100110011001100110011001100110011

根据浮点数的加法法则，0.1 + 0.2 的二进制表示如下所示

0.1 + 0.2	0 01111111101 0011001100110011001100110011001100110011001100110100
-----------	--

所以 0.1 + 0.2 的二进制与 0.3 的二进制表示并不相同，因此JavaScript中 0.1 + 0.2 ≠ 0.3。

## 3. 解决方案

为了解决精度的问题，我们可以设置一个误差范围值，当两个数值之差小于这个范围值时，就可以认为这两个数是相等的，就如ES6中增加的Number.EPSILON，这个值等于 $2^{-52}$ ，可以作为判断的误差范围值，因此判断时可以用  $0.1 + 0.2 - 0.3 < \text{Number.EPSILON}$ ，就能粗略地解决浮点数判等的问题。