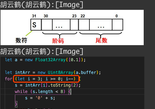
大端、小端 群聊记录

胡云鹤(胡云鹤):IEEE754标准定义的32位浮点数格式是以小端模式表示。
js中的TypedArroy则是大端模式，所以需要将Uint8Array反转一下，最后的结果就是正确的了
胡云鹤(胡云鹤):[Image]



胡云鹤(胡云鹤):[Image]

```
let a = new Float32Array([0.1]);

let intArr = new Uint8Array(a.buffer);
for(let i = 0; i < 4; i++){
    s = intArr[i].toString(2);
    while (s.length < 8) {
        s = '0' + s;
    }
    console.log(s);
    document.write(s);
}

document.write('<br>');
```

彭富晓(彭富晓):IEEE754 本身并没有定义数据是以大端还是小端存储，我们自己将 10 进制的数转换为 2 进制的数后，其实符合人的直觉，像这种存储方式就是大端模式存储，浏览器的大小端模式依赖于 cpu 的，intel 系列芯片都是小端模式是示的。
胡云鹤(胡云鹤):Intel芯片决定了浮点数以小端模式存储，但是浏览器又是以大端模式显示造成了这样的结果。
彭富晓(彭富晓):可能是我没表达清楚，【捂脸哭】
cpu 的类型决定了操作系统如何存储多字节数，浏览器依赖操作系统，我们常用的 intel 系列芯片都是 小端模式，我们自己手动将 10 进制小数转换为的 2 进制小数，这样符合人的直觉的，高字节在低地址位，低字节在高字节位，是大端模式，
吴林胜(吴林胜):@彭富晓(彭富晓) 你说的意思是，为了更适应人的视觉，将它逆向调换了位置？
彭富晓(彭富晓):@吴林胜(吴林胜) 是的

判断符号，需要考虑正负 0，num/Math.abs(num)，以及 INFINITY 的情况

```
function check(zero) {
    if (1/zero === Infinity) return 1;
    if (1/zero === -Infinity) return -1;
}
```

19.9*100 !== 1990
浮点数 **可接受精度丢失值**
减少对浮点数的算术运算，需要考虑精度丢失问题

- 小数转二进制精度会丢失
- 对浮点数进行运算也会造成精度丢失

十进制转二进制
整数部分：除二取余
小数部分：乘二取整
0.1 * 2 = 0.2 --> 0 0.2 * 2 = 0.4 --> 0 0.4 * 2 = 0.8 --> 0 0.8 * 2 = 1.6 --> 1 --> 0.6 0.6 * 2 = 1.2 --> 1 0.2 * 2 = 0.4 --> 0 0.4 * 2 = 0.8 --> 0 0.8 * 2 = 1.6 --> 1 --> 0.6 _ 循环

https://jsfiddle.net/plh8qeor/19/
http://www.softelectro.ru/ieee754_en.html

INFINITY、NaN、这些数值如何表示？
FLOAT 在内存中的排布方法。

Expression

优先级：Tree vs Priority，表达式生成树

> 12.3 Left-Hand-Side Expressions

Member 运算符

- a.b 访问对象成员
- a[b] 动态运行时，支持变量访问属性
- foo`string`
- super.b 只能在 constructor 中使用，调父类
- super['b']
- new.target 原封不动，只能在构造函数中使用，访问 new 生成的对象，**如何判断 foo 是否被 new 调用 => 防御性 feature，这个可以用于安全隔离吗**
- new Foo() 比不带括号优先级更高

```
// 如何判断 foo 被 new 调用 => 防御性代码，这个可以用于安全隔离吗
function foo () {
    console.log(this);
    console.log(new.target);
}

new foo;

var fakeObject = {};
Object.setPrototypeOf(fakeObject, foo.prototype);
foo.apply(fakeObject);

fakeObject instanceof foo; // true
```

```
var name = 'winter';

function foo() {
    console.log(arguments);
}

foo`${name}`;
```

Member 运算符 返回 Reference 类型

- Object
- Key

下面两种运算才会体现出赋值，可写的特性

- delete
- assign

New 运算符

- new Foo

Call

- foo()
- super()
- foo()['b']
- foo().b
- foo()`abc`

Left HandSide & Right HandSide

- a.b = c
- a + b = c

Update，No LineTerminator，中间是注释也不行，影响自动分号插入

- a++
- a--
- --a
- ++a

```
let a = 1, b = 1, c = 1;

a
++
b
++
c;
```

Unary

- delete a.b
- void foo(); // void 0 生成 undefined，改变语法结构的作用，IIFE，用括号时多个 IIFE 合并在一起且没加分号时会导致异常
- typeof a   // js 中没有一种完美的方案，toString 无法区分包装类型原始类型，typeof 与类型表现不一致，无法区分对象中不同 class
    - null
    - function
- +a
- -a
- ~a
- !a，类型转换
- await a

```
void function(i) {
    button.onclick = function() {
        console.log(i);
    }
}(i);

(function(i) {
    button.onclick = function() {
        console.log(i);
    }
})(i){function(i) {
    button.onclick = function() {
        console.log(i);
    }
})(i)
```

Exponental

- **

Multiplicative

- * / %

Additive

- + -

Shift

- << >> >>>

RelationShip

- < > <= >= instanceof in

Equality

- ==
- !=
- ===
- !==

Bitwise

- & ^ |

Logical，短路逻辑

- &&
- ||

Conditional，也是短路逻辑

- ? . :
- 三目运算 + 箭头函数

Comma

- ,

	Null	Undefined	Boolean(true)	Boolean(false)	Number	String	Symbol	Object
Boolean	FALSE	FALSE	-	-	0/NaN-false	""-false	TRUE	TRUE
Number	0	NaN	1	0	-	#StringToNumber	TypeError	#拆箱转换
String	"null"	"undefined"	TRUE	FALSE	#NumberToString	-	TypeError	#拆箱转换
Object	TypeError	TypeError	#装箱转换	#装箱转换	#装箱转换	#装箱转换	#装箱转换	-

---

boxing && unboxing

- toPrimitive // 7.1.1
- valueOf vs toString

StringToNumber

- parseInt, parseFloar
- Number
- Literal

NumberToString

```
function convertStringToNumber(str, x) {
    let chars = str.split('');
    let number = 0;
    for (let i = 0; i < chars.length; i++) {
        number = number * x;
        number += chars[i].codePointAt(0) - '0'.codePointAt(0);
    }
    return number;
}


function convertNumberToString(number, x) {

}
```

https://github.com/tc39/test262