ArrayBuffer

2020年1月10日

13:58

* 1. **什么是二进制数组?**

ArrayBuffer对象,TypedArray视图,DataView视图是JS 操作二进制数据的一个接口,都是以数组的语法处理二进制数据,所以统称为二进制数组

**ArrayBuffer对象 :** 代表内存中的一段二进制数组,可以通过视图进行操作.视图部署了数组接口,即可以用数组的方法操作内存

**TypedArray视图 :** 共包括9种不同的视图

**DataView视图 :** 可以定义符合格式的视图,比如第一个字节是 Uint8、第二个字节是 Int16等等

**总结 :** ArrayBuffer对象代表原始的二进制数据，TypedArray视图用来读写简单类型的二进制数据，DataView视图用来读写复杂类型的二进制数据。

**TypedArray的九种数据类型(DataView支持Uint8C以外的8种)**



* 1. **使用例子 :**

**2.1 DataView**

const buf = new ArrayBuffer(32);

const dataView = new DataView(buf);

dataView.getUint8(0) // 0

**2. 2 TypedArray**

const buffer = new ArrayBuffer(12);

const x1 = new Int32Array(buffer);

x1[0] = 1;

const x2 = new Uint8Array(buffer);

x2[0] = 2;

x1[0] // 2 //两个视图对应同一段内存,一个视图修改底层内存,另一个视图也会发生改变

**TypedArray接受普通数组作为参数**

const typedArray = new Uint8Array([0,1,2]);

typedArray.length // 3

typedArray[0] = 5;

typedArray // [5, 1, 2]

* 1. **ArrayBuffer**

**3.1 ArrayBuffer.prototype.byteLength**

const buffer = new ArrayBuffer(32);

buffer.byteLength

内存分配很可能失败,所以有必要检查是否分配成功

if (buffer.byteLength === n) {

// 成功

} else {

// 失败

}

**3.1ArrayBuffer.prototype.slice()**

**拷贝指定片段的内存区域,生成一个新的ArrayBuffer**

const buffer = new ArrayBuffer(8);

const newBuffer = buffer.slice(0, 3);

**3.2ArrayBuffer.isView()**

const buffer = new ArrayBuffer(8);

ArrayBuffer.isView(buffer) // false

const v = new Int32Array(buffer);

ArrayBuffer.isView(v) // true

* 1. **TypedArray视图**

TypedArray和数组一样,都能使用length属性,都能使用[]获取单个元素,所有数组的方法,在它们上面都能使用.TypedArray和数组的主要差异在于 :

---TypedArray数组的所有成员,数据类型都是一样的

---TypedArray数组的成员是连续的,不会有空位

---TypedArray数组的成员默认值是0

---TypedArray本身不存储数组,它的数组都存储在ArrayBuffer对象中

**4.1 TypedArray(ArrayBuffer,byteOffset = 0,length ?)**

// 创建一个8字节的ArrayBuffer

const b = new ArrayBuffer(8);

// 创建一个指向b的Int32视图，开始于字节0，直到缓冲区的末尾

const v1 = new Int32Array(b);

// 创建一个指向b的Uint8视图，开始于字节2，直到缓冲区的末尾

const v2 = new Uint8Array(b, 2);

// 创建一个指向b的Int16视图，开始于字节2，长度为2

const v3 = new Int16Array(b, 2, 2);

**4.2 TypedArray(length)**

通过直接分配内存,生成TypedArray对象

const f64a = new Float64Array(8);

f64a[0] = 10;

f64a[1] = 20;

f64a[2] = f64a[0] + f64a[1];

**4.3 TypedArray(TypedArray)**

接受另一个实例作为参数,生成该实例的拷贝,新实例与旧实例的内存不一样

const x = new Int8Array([1, 1]);

const y = new Int8Array(x);

x[0] // 1

y[0] // 1

x[0] = 2;

y[0] // 1

如果希望生成新的视图并且两个视图实例对应的内存是一样的,那么可以

const x = new Int8Array([1, 1]);

const y = new Int8Array(x.buffer);

x[0] // 1

y[0] // 1

x[0] = 2;

y[0] // 2

注意 : TypedArray没有.concat和.push方法,因此,要合并二进制数组可以使用 let newArr = TypedArray([…arr1,arr2])

或者 let newArr =new TypedArray(arr1.length + arr2.length)

newArr.set(arr1,arr1.length)

newArr.set(arr2,arr3.length)

**对于多个数组合并,应使用后者避免多次生成新的视图**

**4.4 迭代器**

let ui8 = Uint8Array.of(0, 1, 2);

for (let byte of ui8) {

console.log(byte);

}

// 0

// 1

// 2

**4.5 字节序**