原文地址：<https://blog.csdn.net/ithanmang/article/details/105162357>

**ArrayBuffer**

Array对象并不是直接操作内存的，为了更高效的操作数据，所以就有了ArrayBuffer对象

// 创建8个字节的缓冲区

let buffer = new ArrayBuffer(8);

console.log(buffer.byteLength); // 返回字节长度8

ArrayBuffer称为缓存，我们无法直接操作ArrayBuffer的数据

**DataView**

想要操作ArrayBuffer数据可以通过DataView进行操作，DataView封装了访问缓存的方法

interface DataView {

    readonly buffer: ArrayBuffer;

    readonly byteLength: number;

    readonly byteOffset: number;

    // 获取数据

    getFloat32(byteOffset: number, littleEndian?: boolean): number;

    getFloat64(byteOffset: number, littleEndian?: boolean): number;

    getInt8(byteOffset: number): number;

    getInt16(byteOffset: number, littleEndian?: boolean): number;

    getInt32(byteOffset: number, littleEndian?: boolean): number;

    getUint8(byteOffset: number): number;

    getUint16(byteOffset: number, littleEndian?: boolean): number;

    getUint32(byteOffset: number, littleEndian?: boolean): number;

    // 设置数据

    setFloat32(byteOffset: number, value: number, littleEndian?: boolean): void;

    setFloat64(byteOffset: number, value: number, littleEndian?: boolean): void;

    setInt8(byteOffset: number, value: number): void;

    setInt16(byteOffset: number, value: number, littleEndian?: boolean): void;

    setInt32(byteOffset: number, value: number, littleEndian?: boolean): void;

    setUint8(byteOffset: number, value: number): void;

    setUint16(byteOffset: number, value: number, littleEndian?: boolean): void;

    setUint32(byteOffset: number, value: number, littleEndian?: boolean): void;

}

通过get和set系列函数操作ArrayBuffer

- 示例

// 创建8个字节的缓冲区

let buffer = new ArrayBuffer(8);

// 创建一个视图，从第一个字节开始到buffer的长度为止，通过视图可以操作缓冲区

let view = new DataView(buffer);

view.setUint8(0, 31);// 在 0 偏移位置存储一个 8 位无符号整数31 占一个字节

view.setUint8(1, 32);// 在 1 偏移位置存储一个 8 位无符号整数31 占一个字节

view.setInt16(2, 67);// 在 2 偏移位置也就是第3个字节开始 存储一个 16位有符号整数 占两个字节

view.setFloat32(4, 12.5)// 在 4 偏移位置存储一个有符号32位浮点数占 4 个字节

console.log(view.getUint16(2)); // 67

console.log(view.getUint8(0));  // 31

**类型化数组**

上面说DataView可以操作ArrayBuffer数据，还有一种操作数据的方法就是类型化数组

| 类型 | 值范围 | 字节数 | 描述 |
| --- | --- | --- | --- |
| [Int8Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Int8Array) | -128~127 | 1 | 8 位有符号整数（补码） |
| [Uint8Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Uint8Array) | 0~255 | 1 | 8 位无符号整数 |
| [Uint8ClampedArray](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Uint8ClampedArray) | 0~255 | 1 | 8 位无符号整数（值会被裁剪） |
| [Int16Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Int16Array) | -32768~32767 | 2 | 16 位有符号整数（补码） |
| [Uint16Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Uint16Array) | 0~65535 | 2 | 16 位有符号整数 |
| [Int32Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Int32Array) | -2147483648~2147483647 | 4 | 32 位有符号整数（补码） |
| [Uint32Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Uint32Array) | 0~4294967295 | 4 | 32 位有符号整数 |
| [Float32Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Float32Array) | -3.4E38~3.4E38 以及 1.2E-38（最小正数） | 4 | 32 位 IEEE 浮点数（7 位有效数字，例如 1.123456） |
| [Float64Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Float64Array) | -1.8E308~1.8E308 以及 5E-324（最小正数） | 8 | 64 位 IEEE 浮点数（16 位有效数字，例如 1.123...15） |
| [BigInt64Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/BigInt64Array) | -263~263 - 1 | 8 | 64 位有符号整数（补码） |
| [BigUint64Array](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/BigUint64Array) | 0~264 - 1 | 8 | 64 位无符号整数 |

- 使用示例

/\*\*

  \* 1. new(length: number): Float32Array; 传递一个 length

  \* 创建一个可以存储8个浮点数的类型化数组

  \* 当传入length时，一个内部的 buffer 会被创建，该缓冲区的大小

  \* 是传入的length乘以数组中每个元素的字节数，并且每个元素的值都是0

  \*/

let float32Array = new Float32Array(8);

console.log(float32Array.BYTES\_PER\_ELEMENT);// 4每个数所占的字节

console.log(float32Array.byteLength);       // 总字节数 4 \* 8 = 64

console.log(float32Array.length);           // 数组中元素的数量

console.log(float32Array.byteOffset);       // 0

/\*\*

  \* 2. new(arrayOrArrayBuffer: ArrayLike<number>): Float32Array;

  \* 第二种方法是传递一个普通的数组，然后通过这个数组来创建实例

  \* 此时类型化数组中的值是已经赋值过的

  \*/

let vertices = [1.0, 1.0, 1.0, 2.0, 2.0, 2.0];

let float32Array = new Float32Array(vertices);

console.log(float32Array.BYTES\_PER\_ELEMENT);// 4每个数所占的字节

console.log(float32Array.byteLength);       // 总字节数 4 \* 6 = 24

console.log(float32Array.length);           // 数组中元素的数量 6

console.log(float32Array.byteOffset);       // 0

- 不能动态增长

类型化数组的长度在new时固定好了，不像array那样可以push

- 属性访问

let vertices = [1.0, 2.0, 3.0];

let float32Array = new Float32Array(vertices);

console.log(float32Array);                      // [1.0, 2.0, 3.0]

console.log(float32Array[0]);                   // 1