二进制数组

二进制数组（`ArrayBuffer`对象、TypedArray视图和`DataView`视图）是JavaScript操作二进制数据的一个接口。这些对象早就存在，属于独立的规格（2011年2月发布），ES6将它们纳入了ECMAScript规格，并且增加了新的方法。

这个接口的原始设计目的，与WebGL项目有关。所谓WebGL，就是指浏览器与显卡之间的通信接口，为了满足JavaScript与显卡之间大量的、实时的数据交换，它们之间的数据通信必须是二进制的，而不能是传统的文本格式。文本格式传递一个32位整数，两端的JavaScript脚本与显卡都要进行格式转化，将非常耗时。这时要是存在一种机制，可以像C语言那样，直接操作字节，将4个字节的32位整数，以二进制形式原封不动地送入显卡，脚本的性能就会大幅提升。

二进制数组就是在这种背景下诞生的。它很像C语言的数组，允许开发者以数组下标的形式，直接操作内存，大大增强了JavaScript处理二进制数据的能力，使得开发者有可能通过JavaScript与操作系统的原生接口进行二进制通信。

二进制数组由三类对象组成。

（**1**）`ArrayBuffer`对象：代表内存之中的一段二进制数据，可以通过"视图"进行操

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

作。“视图”部署了数组接口，这意味着，可以用数组的方法操作内存。

（2）**TypedArray**视图：共包括9种类型的视图，比如`Uint8Array`（无符号8位整数）数组视图，`Int16Array`（16位整数）数组视图，`Float32Array`（32位浮点数）数组视图等等。

（3）`DataView`视图：可以自定义复合格式的视图，比如第一个字节是Uint8（无符号8位整数）、第二、三个字节是Int16（16位整数）、第四个字节开始是Float32（32位浮点数）等等，此外还可以自定义字节序。

简单说，`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视图支持的数据类型一共有9种（`DataView`视图支持除`Uint8C`以外的其他8种）。

数据类型	字节长度	含义	对应的C语言类型
Int8	1	8位带符号整数	signed char
Uint8	1	8位不带符号整数	unsigned char
Uint8C	1	8位不带符号整数（自动过滤溢出）	unsigned char
Int16	2	16位带符号整数	short
Uint16	2	16位不带符号整数	unsigned short

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Int32	4	32位带符号整数	int
Uint32	4	32位不带符号的整数	unsigned int
Float32	4	32位浮点数	float
Float64	8	64位浮点数	double

注意，二进制数组并不是真正的数组，而是类似数组的对象。

很多浏览器操作的API，用到了二进制数组操作二进制数据，下面是其中的几个。

- File API
- XMLHttpRequest
- Fetch API
- Canvas
- WebSockets

---

1. ArrayBuffer对象

概述

`ArrayBuffer`对象代表储存二进制数据的一段内存，它不能直接读写，只能通过视图（TypedArray视图和`DataView`视图)来读写，视图的作用是以指定格式解读二进制数

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

据。

`ArrayBuffer`也是一个构造函数，可以分配一段可以存放数据的连续内存区域。

```
var buf = new ArrayBuffer(32);
```

上面代码生成了一段32字节的内存区域，每个字节的值默认都是0。可以看到，`ArrayBuffer`构造函数的参数是所需要的内存大小（单位字节）。

为了读写这段内容，需要为它指定视图。`DataView`视图的创建，需要提供`ArrayBuffer`对象实例作为参数。

```
var buf = new ArrayBuffer(32);
var dataView = new DataView(buf);
dataView.getUint8(0) // 0
```

上面代码对一段32字节的内存，建立`DataView`视图，然后以不带符号的8位整数格式，读取第一个元素，结果得到0，因为原始内存的`ArrayBuffer`对象，默认所有位都是0。

另一种TypedArray视图，与`DataView`视图的一个区别是，它不是一个构造函数，而是一组构造函数，代表不同的数据格式。

```
var buffer = new ArrayBuffer(12);

var x1 = new Int32Array(buffer);
x1[0] = 1;
var x2 = new Uint8Array(buffer);
x2[0]  = 2;
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
x1[0] // 2
```

上面代码对同一段内存，分别建立两种视图：32位带符号整数（`Int32Array`构造函数）和8位不带符号整数（`Uint8Array`构造函数）。由于两个视图对应的是同一段内存，一个视图修改底层内存，会影响到另一个视图。

TypedArray视图的构造函数，除了接受`ArrayBuffer`实例作为参数，还可以接受普通数组作为参数，直接分配内存生成底层的`ArrayBuffer`实例，并同时完成对这段内存的赋值。

```
var typedArray = new Uint8Array([0,1,2]);
typedArray.length // 3

typedArray[0] = 5;
typedArray // [5, 1, 2]
```

上面代码使用TypedArray视图的`Uint8Array`构造函数，新建一个不带符号的8位整数视图。可以看到，`Uint8Array`直接使用普通数组作为参数，对底层内存的赋值同时完成。

ArrayBuffer.prototype.byteLength

`ArrayBuffer`实例的`byteLength`属性，返回所分配的内存区域的字节长度。

```
var buffer = new ArrayBuffer(32);
buffer.byteLength
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
// 32
```

如果要分配的内存区域很大，有可能分配失败（因为没有那么多的连续空余内存），所以有必要检查是否分配成功。

```
if (buffer.byteLength === n) {
  // 成功
} else {
  // 失败
}
```

ArrayBuffer.prototype.slice()

`ArrayBuffer`实例有一个`slice`方法，允许将内存区域的一部分，拷贝生成一个新的`ArrayBuffer`对象。

```
var buffer = new ArrayBuffer(8);
var newBuffer = buffer.slice(0, 3);
```

上面代码拷贝`buffer`对象的前3个字节（从0开始，到第3个字节前面结束），生成一个新的`ArrayBuffer`对象。`slice`方法其实包含两步，第一步是先分配一段新内存，第二步是将原来那个`ArrayBuffer`对象拷贝过去。

`slice`方法接受两个参数，第一个参数表示拷贝开始的字节序号（含该字节），第二个参数表示拷贝截止的字节序号（不含该字节）。如果省略第二个参数，则默认到原`ArrayBuffer`对象的结尾。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

除了`slice`方法，`ArrayBuffer`对象不提供任何直接读写内存的方法，只允许在其上方建立视图，然后通过视图读写。

ArrayBuffer.isView()

`ArrayBuffer`有一个静态方法`isView`，返回一个布尔值，表示参数是否为`ArrayBuffer`的视图实例。这个方法大致相当于判断参数，是否为TypedArray实例或`DataView`实例。

```
var buffer = new ArrayBuffer(8);
ArrayBuffer.isView(buffer) // false

var v = new Int32Array(buffer);
ArrayBuffer.isView(v) // true
```

---

2. TypedArray视图

概述

`ArrayBuffer`对象作为内存区域，可以存放多种类型的数据。同一段内存，不同数据有

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

不同的解读方式，这就叫做"视图"（view）。`ArrayBuffer`有两种视图，一种是TypedArray视图，另一种是`DataView`视图。前者的数组成员都是同一个数据类型，后者的数组成员可以是不同的数据类型。

目前，TypedArray视图一共包括9种类型，每一种视图都是一种构造函数。

- `Int8Array`：8位有符号整数，长度1个字节。
- `Uint8Array`：8位无符号整数，长度1个字节。
- `Uint8ClampedArray`：8位无符号整数，长度1个字节，溢出处理不同。
- `Int16Array`：16位有符号整数，长度2个字节。
- `Uint16Array`：16位无符号整数，长度2个字节。
- `Int32Array`：32位有符号整数，长度4个字节。
- `Uint32Array`：32位无符号整数，长度4个字节。
- `Float32Array`：32位浮点数，长度4个字节。
- `Float64Array`：64位浮点数，长度8个字节。

这9个构造函数生成的数组，统称为TypedArray视图。它们很像普通数组，都有`length`属性，都能用方括号运算符（`[]`）获取单个元素，所有数组的方法，在它们上面都能使用。普通数组与TypedArray数组的差异主要在以下方面。

- TypedArray数组的所有成员，都是同一种类型。
- TypedArray数组的成员是连续的，不会有空位。
- TypedArray数组成员的默认值为0。比如，`new Array(10)`返回一个普通数组，里面没有任何成员，只是10个空位；`new Uint8Array(10)`返回一个TypedArray数组，里面10个成员都是0。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

- TypedArray数组只是一层视图，本身不储存数据，它的数据都储存在底层的`ArrayBuffer`对象之中，要获取底层对象必须使用`buffer`属性。

构造函数

TypedArray数组提供9种构造函数，用来生成相应类型的数组实例。

构造函数有多种用法。

（1）TypedArray(buffer, byteOffset=0, length?)

同一个`ArrayBuffer`对象之上，可以根据不同的数据类型，建立多个视图。

```
// 创建一个8字节的ArrayBuffer
var 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
var 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
var 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
var v3 = new Int16Array(b, 2, 2);
```

上面代码在一段长度为8个字节的内存（`b`）之上，生成了三个视图：`v1`、`v2`和`v3`。

视图的构造函数可以接受三个参数：

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

- 第一个参数（必需）：视图对应的底层`ArrayBuffer`对象。
- 第二个参数（可选）：视图开始的字节序号，默认从0开始。
- 第三个参数（可选）：视图包含的数据个数，默认直到本段内存区域结束。

因此，`v1`、`v2`和`v3`是重叠的：`v1[0]`是一个32位整数，指向字节0～字节3；`v2[0]`是一个8位无符号整数，指向字节2；`v3[0]`是一个16位整数，指向字节2～字节3。只要任何一个视图对内存有所修改，就会在另外两个视图上反应出来。

注意，`byteOffset`必须与所要建立的数据类型一致，否则会报错。

```
var buffer = new ArrayBuffer(8);
var i16 = new Int16Array(buffer, 1);
// Uncaught RangeError: start offset of Int16Array should be a multiple
```

上面代码中，新生成一个8个字节的`ArrayBuffer`对象，然后在这个对象的第一个字节，建立带符号的16位整数视图，结果报错。因为，带符号的16位整数需要两个字节，所以`byteOffset`参数必须能够被2整除。

如果想从任意字节开始解读`ArrayBuffer`对象，必须使用`DataView`视图，因为TypedArray视图只提供9种固定的解读格式。

（2）TypedArray(length)

视图还可以不通过`ArrayBuffer`对象，直接分配内存而生成。

```
var f64a = new Float64Array(8);
f64a[0] = 10;
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
f64a[1] = 20;
f64a[2] = f64a[0] + f64a[1];
```

上面代码生成一个8个成员的`Float64Array`数组（共64字节），然后依次对每个成员赋值。这时，视图构造函数的参数就是成员的个数。可以看到，视图数组的赋值操作与普通数组的操作毫无两样。

（3）TypedArray(typedArray)

TypedArray数组的构造函数，可以接受另一个TypedArray实例作为参数。

```
var typedArray = new Int8Array(new Uint8Array(4));
```

上面代码中，`Int8Array`构造函数接受一个`Uint8Array`实例作为参数。

注意，此时生成的新数组，只是复制了参数数组的值，对应的底层内存是不一样的。新数组会开辟一段新的内存储存数据，不会在原数组的内存之上建立视图。

```
var x = new Int8Array([1, 1]);
var y = new Int8Array(x);
x[0] // 1
y[0] // 1

x[0] = 2;
y[0] // 1
```

上面代码中，数组`y`是以数组`x`为模板而生成的，当`x`变动的时候，`y`并没有变动。

如果想基于同一段内存，构造不同的视图，可以采用下面的写法。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
var x = new Int8Array([1, 1]);
var y = new Int8Array(x.buffer);
x[0] // 1
y[0] // 1

x[0] = 2;
y[0] // 2
```

（4）TypedArray(arrayLikeObject)

构造函数的参数也可以是一个普通数组，然后直接生成TypedArray实例。

```
var typedArray = new Uint8Array([1, 2, 3, 4]);
```

注意，这时TypedArray视图会重新开辟内存，不会在原数组的内存上建立视图。

上面代码从一个普通的数组，生成一个8位无符号整数的TypedArray实例。

TypedArray数组也可以转换回普通数组。

```
var normalArray = Array.prototype.slice.call(typedArray);
```

数组方法

普通数组的操作方法和属性，对TypedArray数组完全适用。

- `TypedArray.prototype.copyWithin(target, start[, end =`

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

`this.length])`

- `TypedArray.prototype.entries()`
- `TypedArray.prototype.every(callbackfn, thisArg?)`
- `TypedArray.prototype.fill(value, start=0, end=this.length)`
- `TypedArray.prototype.filter(callbackfn, thisArg?)`
- `TypedArray.prototype.find(predicate, thisArg?)`
- `TypedArray.prototype.findIndex(predicate, thisArg?)`
- `TypedArray.prototype.forEach(callbackfn, thisArg?)`
- `TypedArray.prototype.indexOf(searchElement, fromIndex=0)`
- `TypedArray.prototype.join(separator)`
- `TypedArray.prototype.keys()`
- `TypedArray.prototype.lastIndexOf(searchElement, fromIndex?)`
- `TypedArray.prototype.map(callbackfn, thisArg?)`
- `TypedArray.prototype.reduce(callbackfn, initialValue?)`
- `TypedArray.prototype.reduceRight(callbackfn, initialValue?)`
- `TypedArray.prototype.reverse()`
- `TypedArray.prototype.slice(start=0, end=this.length)`
- `TypedArray.prototype.some(callbackfn, thisArg?)`
- `TypedArray.prototype.sort(comparefn)`
- `TypedArray.prototype.toLocaleString(reserved1?, reserved2?)`
- `TypedArray.prototype.toString()`
- `TypedArray.prototype.values()`

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

上面所有方法的用法，请参阅数组方法的介绍，这里不再重复了。

注意，TypedArray数组没有`concat`方法。如果想要合并多个TypedArray数组，可以用下面这个函数。

```
function concatenate(resultConstructor, ...arrays) {
  let totalLength = 0;
  for (let arr of arrays) {
    totalLength += arr.length;
  }
  let result = new resultConstructor(totalLength);
  let offset = 0;
  for (let arr of arrays) {
    result.set(arr, offset);
    offset += arr.length;
  }
  return result;
}

concatenate(Uint8Array, Uint8Array.of(1, 2), Uint8Array.of(3, 4))
// Uint8Array [1, 2, 3, 4]
```

另外，TypedArray数组与普通数组一样，部署了Iterator接口，所以可以被遍历。

```
let ui8 = Uint8Array.of(0, 1, 2);
for (let byte of ui8) {
  console.log(byte);
}
// 0
// 1
// 2
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

字节序

字节序指的是数值在内存中的表示方式。

```
var buffer = new ArrayBuffer(16);
var int32View = new Int32Array(buffer);

for (var i = 0; i < int32View.length; i++) {
  int32View[i] = i * 2;
}
```

上面代码生成一个16字节的`ArrayBuffer`对象，然后在它的基础上，建立了一个32位整数的视图。由于每个32位整数占据4个字节，所以一共可以写入4个整数，依次为0，2，4，6。

如果在这段数据上接着建立一个16位整数的视图，则可以读出完全不一样的结果。

```
var int16View = new Int16Array(buffer);

for (var i = 0; i < int16View.length; i++) {
  console.log("Entry " + i + ": " + int16View[i]);
}
// Entry 0: 0
// Entry 1: 0
// Entry 2: 2
// Entry 3: 0
// Entry 4: 4
// Entry 5: 0
// Entry 6: 6
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
// Entry 7: 0
```

由于每个16位整数占据2个字节，所以整个`ArrayBuffer`对象现在分成8段。然后，由于x86体系的计算机都采用小端字节序（little endian），相对重要的字节排在后面的内存地址，相对不重要字节排在前面的内存地址，所以就得到了上面的结果。

比如，一个占据四个字节的16进制数`0x12345678`，决定其大小的最重要的字节是"12"，最不重要的是"78"。小端字节序将最不重要的字节排在前面，储存顺序就是`78563412`；大端字节序则完全相反，将最重要的字节排在前面，储存顺序就是`12345678`。目前，所有个人电脑几乎都是小端字节序，所以TypedArray数组内部也采用小端字节序读写数据，或者更准确的说，按照本机操作系统设定的字节序读写数据。

这并不意味大端字节序不重要，事实上，很多网络设备和特定的操作系统采用的是大端字节序。这就带来一个严重的问题：如果一段数据是大端字节序，TypedArray数组将无法正确解析，因为它只能处理小端字节序！为了解决这个问题，JavaScript引入`DataView`对象，可以设定字节序，下文会详细介绍。

下面是另一个例子。

```
// 假定某段buffer包含如下字节 [0x02, 0x01, 0x03, 0x07]
var buffer = new ArrayBuffer(4);
var v1 = new Uint8Array(buffer);
v1[0] = 2;
v1[1] = 1;
v1[2] = 3;
v1[3] = 7;

var uInt16View = new Uint16Array(buffer);
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
// 计算机采用小端字节序
// 所以头两个字节等于258
if (uInt16View[0] === 258) {
  console.log('OK'); // "OK"
}

// 赋值运算
uInt16View[0] = 255;    // 字节变为[0xFF, 0x00, 0x03, 0x07]
uInt16View[0] = 0xff05; // 字节变为[0x05, 0xFF, 0x03, 0x07]
uInt16View[1] = 0x0210; // 字节变为[0x05, 0xFF, 0x10, 0x02]
```

下面的函数可以用来判断，当前视图是小端字节序，还是大端字节序。

```
const BIG_ENDIAN = Symbol('BIG_ENDIAN');
const LITTLE_ENDIAN = Symbol('LITTLE_ENDIAN');

function getPlatformEndianness() {
  let arr32 = Uint32Array.of(0x12345678);
  let arr8 = new Uint8Array(arr32.buffer);
  switch ((arr8[0]*0x1000000) + (arr8[1]*0x10000) + (arr8[2]*0x100) + (
    case 0x12345678:
      return BIG_ENDIAN;
    case 0x78563412:
      return LITTLE_ENDIAN;
    default:
      throw new Error('Unknown endianness');
  }
}
```

总之，与普通数组相比，TypedArray数组的最大优点就是可以直接操作内存，不需要数据类型转换，所以速度快得多。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

BYTES_PER_ELEMENT属性

每一种视图的构造函数，都有一个`BYTES_PER_ELEMENT`属性，表示这种数据类型占据的字节数。

```
Int8Array.BYTES_PER_ELEMENT // 1
Uint8Array.BYTES_PER_ELEMENT // 1
Int16Array.BYTES_PER_ELEMENT // 2
Uint16Array.BYTES_PER_ELEMENT // 2
Int32Array.BYTES_PER_ELEMENT // 4
Uint32Array.BYTES_PER_ELEMENT // 4
Float32Array.BYTES_PER_ELEMENT // 4
Float64Array.BYTES_PER_ELEMENT // 8
```

这个属性在TypedArray实例上也能获取，即有`TypedArray.prototype.BYTES_PER_ELEMENT`。

ArrayBuffer与字符串的互相转换

`ArrayBuffer`转为字符串，或者字符串转为`ArrayBuffer`，有一个前提，即字符串的编码方法是确定的。假定字符串采用UTF-16编码（JavaScript的内部编码方式），可以自己编写转换函数。

```
// ArrayBuffer转为字符串，参数为ArrayBuffer对象
function ab2str(buf) {
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
  return String.fromCharCode.apply(null, new Uint16Array(buf));
}

// 字符串转为ArrayBuffer对象，参数为字符串
function str2ab(str) {
  var buf = new ArrayBuffer(str.length * 2); // 每个字符占用2个字节
  var bufView = new Uint16Array(buf);
  for (var i = 0, strLen = str.length; i < strLen; i++) {
    bufView[i] = str.charCodeAt(i);
  }
  return buf;
}
```

溢出

不同的视图类型，所能容纳的数值范围是确定的。超出这个范围，就会出现溢出。比如，8位视图只能容纳一个8位的二进制值，如果放入一个9位的值，就会溢出。

TypedArray数组的溢出处理规则，简单来说，就是抛弃溢出的位，然后按照视图类型进行解释。

```
var uint8 = new Uint8Array(1);

uint8[0] = 256;
uint8[0] // 0

uint8[0] = -1;
uint8[0] // 255
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

上面代码中，`uint8`是一个8位视图，而256的二进制形式是一个9位的值`100000000`，这时就会发生溢出。根据规则，只会保留后8位，即`00000000`。`uint8`视图的解释规则是无符号的8位整数，所以`00000000`就是`0`。

负数在计算机内部采用"2的补码"表示，也就是说，将对应的正数值进行否运算，然后加`1`。比如，`-1`对应的正值是`1`，进行否运算以后，得到`11111110`，再加上`1`就是补码形式`11111111`。`uint8`按照无符号的8位整数解释`11111111`，返回结果就是`255`。

一个简单转换规则，可以这样表示。

- 正向溢出（overflow）：当输入值大于当前数据类型的最大值，结果等于当前数据类型的最小值加上余值，再减去1。
- 负向溢出（underflow）：当输入值小于当前数据类型的最小值，结果等于当前数据类型的最大值减去余值，再加上1。

请看下面的例子。

```
var int8 = new Int8Array(1);

int8[0] = 128;
int8[0] // -128

int8[0] = -129;
int8[0] // 127
```

上面例子中，`int8`是一个带符号的8位整数视图，它的最大值是127，最小值是-128。输入值为`128`时，相当于正向溢出`1`，根据"最小值加上余值，再减去1"的规则，就会返回`-128`；输入值为`-129`时，相当于负向溢出`1`，根据"最大值减去余值，再加上1"的规

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

则，就会返回`127`。

`Uint8ClampedArray`视图的溢出规则，与上面的规则不同。它规定，凡是发生正向溢出，该值一律等于当前数据类型的最大值，即255；如果发生负向溢出，该值一律等于当前数据类型的最小值，即0。

```
var uint8c = new Uint8ClampedArray(1);

uint8c[0] = 256;
uint8c[0] // 255

uint8c[0] = -1;
uint8c[0] // 0
```

上面例子中，`uint8C`是一个`Uint8ClampedArray`视图，正向溢出时都返回255，负向溢出都返回0。

TypedArray.prototype.buffer

TypedArray实例的`buffer`属性，返回整段内存区域对应的`ArrayBuffer`对象。该属性为只读属性。

```
var a = new Float32Array(64);
var b = new Uint8Array(a.buffer);
```

上面代码的`a`视图对象和`b`视图对象，对应同一个`ArrayBuffer`对象，即同一段内存。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

TypedArray.prototype.byteLength，TypedArray.prototype.byteOffset

`byteLength`属性返回TypedArray数组占据的内存长度，单位为字节。`byteOffset`属性返回TypedArray数组从底层`ArrayBuffer`对象的哪个字节开始。这两个属性都是只读属性。

```
var b = new ArrayBuffer(8);

var v1 = new Int32Array(b);
var v2 = new Uint8Array(b, 2);
var v3 = new Int16Array(b, 2, 2);

v1.byteLength // 8
v2.byteLength // 6
v3.byteLength // 4

v1.byteOffset // 0
v2.byteOffset // 2
v3.byteOffset // 2
```

TypedArray.prototype.length

`length`属性表示TypedArray数组含有多少个成员。注意将`byteLength`属性和`length`属性区分，前者是字节长度，后者是成员长度。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
var a = new Int16Array(8);

a.length // 8
a.byteLength // 16
```

TypedArray.prototype.set()

TypedArray数组的`set`方法用于复制数组（普通数组或TypedArray数组），也就是将一段内容完全复制到另一段内存。

```
var a = new Uint8Array(8);
var b = new Uint8Array(8);

b.set(a);
```

上面代码复制`a`数组的内容到`b`数组，它是整段内存的复制，比一个个拷贝成员的那种复制快得多。

`set`方法还可以接受第二个参数，表示从`b`对象的哪一个成员开始复制`a`对象。

```
var a = new Uint16Array(8);
var b = new Uint16Array(10);

b.set(a, 2)
```

上面代码的`b`数组比`a`数组多两个成员，所以从`b[2]`开始复制。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

TypedArray.prototype.subarray()

`subarray`方法是对于TypedArray数组的一部分，再建立一个新的视图。

```
var a = new Uint16Array(8);
var b = a.subarray(2,3);

a.byteLength // 16
b.byteLength // 2
```

`subarray`方法的第一个参数是起始的成员序号，第二个参数是结束的成员序号（不含该成员），如果省略则包含剩余的全部成员。所以，上面代码的`a.subarray(2,3)`，意味着b只包含`a[2]`一个成员，字节长度为2。

TypedArray.prototype.slice()

TypeArray实例的`slice`方法，可以返回一个指定位置的新的TypedArray实例。

```
let ui8 = Uint8Array.of(0, 1, 2);
ui8.slice(-1)
// Uint8Array [ 2 ]
```

上面代码中，`ui8`是8位无符号整数数组视图的一个实例。它的`slice`方法可以从当前视图之中，返回一个新的视图实例。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

`slice`方法的参数，表示原数组的具体位置，开始生成新数组。负值表示逆向的位置，即-1为倒数第一个位置，-2表示倒数第二个位置，以此类推。

TypedArray.of()

TypedArray数组的所有构造函数，都有一个静态方法`of`，用于将参数转为一个TypedArray实例。

```
Float32Array.of(0.151, -8, 3.7)
// Float32Array [ 0.151, -8, 3.7 ]
```

下面三种方法都会生成同样一个TypedArray数组。

```
// 方法一
let tarr = new Uint8Array([1,2,3]);

// 方法二
let tarr = Uint8Array.of(1,2,3);

// 方法三
let tarr = new Uint8Array(3);
tarr[0] = 1;
tarr[1] = 2;
tarr[2] = 3;
```

---

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

TypedArray.from()

静态方法`from`接受一个可遍历的数据结构（比如数组）作为参数，返回一个基于这个结构的TypedArray实例。

```
Uint16Array.from([0, 1, 2])
// Uint16Array [ 0, 1, 2 ]
```

这个方法还可以将一种TypedArray实例，转为另一种。

```
var ui16 = Uint16Array.from(Uint8Array.of(0, 1, 2));
ui16 instanceof Uint16Array // true
```

`from`方法还可以接受一个函数，作为第二个参数，用来对每个元素进行遍历，功能类似`map`方法。

```
Int8Array.of(127, 126, 125).map(x => 2 * x)
// Int8Array [ -2, -4, -6 ]

Int16Array.from(Int8Array.of(127, 126, 125), x => 2 * x)
// Int16Array [ 254, 252, 250 ]
```

上面的例子中，`from`方法没有发生溢出，这说明遍历不是针对原来的8位整数数组。也就是说，`from`会将第一个参数指定的TypedArray数组，拷贝到另一段内存之中，处理之后再将结果转成指定的数组格式。

---

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

3. 复合视图

由于视图的构造函数可以指定起始位置和长度，所以在同一段内存之中，可以依次存放不同类型的数据，这叫做"复合视图"。

```
var buffer = new ArrayBuffer(24);

var idView = new Uint32Array(buffer, 0, 1);
var usernameView = new Uint8Array(buffer, 4, 16);
var amountDueView = new Float32Array(buffer, 20, 1);
```

上面代码将一个24字节长度的`ArrayBuffer`对象，分成三个部分：

- 字节0到字节3：1个32位无符号整数
- 字节4到字节19：16个8位整数
- 字节20到字节23：1个32位浮点数

这种数据结构可以用如下的C语言描述：

```
struct someStruct {
  unsigned long id;
  char username[16];
  float amountDue;
};
```

---

4. DataView视图

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

如果一段数据包括多种类型（比如服务器传来的HTTP数据），这时除了建立`ArrayBuffer`对象的复合视图以外，还可以通过`DataView`视图进行操作。

`DataView`视图提供更多操作选项，而且支持设定字节序。本来，在设计目的上，`ArrayBuffer`对象的各种TypedArray视图，是用来向网卡、声卡之类的本机设备传送数据，所以使用本机的字节序就可以了；而`DataView`视图的设计目的，是用来处理网络设备传来的数据，所以大端字节序或小端字节序是可以自行设定的。

`DataView`视图本身也是构造函数，接受一个`ArrayBuffer`对象作为参数，生成视图。

```
DataView(ArrayBuffer buffer [, 字节起始位置 [, 长度]]);
```

下面是一个例子。

```
var buffer = new ArrayBuffer(24);
var dv = new DataView(buffer);
```

`DataView`实例有以下属性，含义与TypedArray实例的同名方法相同。

- `DataView.prototype.buffer`：返回对应的ArrayBuffer对象
- `DataView.prototype.byteLength`：返回占据的内存字节长度
- `DataView.prototype.byteOffset`：返回当前视图从对应的ArrayBuffer对象的哪个字节开始

`DataView`实例提供8个方法读取内存。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

- `getInt8`：读取1个字节，返回一个8位整数。
- `getUint8`：读取1个字节，返回一个无符号的8位整数。
- `getInt16`：读取2个字节，返回一个16位整数。
- `getUint16`：读取2个字节，返回一个无符号的16位整数。
- `getInt32`：读取4个字节，返回一个32位整数。
- `getUint32`：读取4个字节，返回一个无符号的32位整数。
- `getFloat32`：读取4个字节，返回一个32位浮点数。
- `getFloat64`：读取8个字节，返回一个64位浮点数。

这一系列`get`方法的参数都是一个字节序号（不能是负数，否则会报错），表示从哪个字节开始读取。

```
var buffer = new ArrayBuffer(24);
var dv = new DataView(buffer);

// 从第1个字节读取一个8位无符号整数
var v1 = dv.getUint8(0);

// 从第2个字节读取一个16位无符号整数
var v2 = dv.getUint16(1);

// 从第4个字节读取一个16位无符号整数
var v3 = dv.getUint16(3);
```

上面代码读取了`ArrayBuffer`对象的前5个字节，其中有一个8位整数和两个十六位整数。

如果一次读取两个或两个以上字节，就必须明确数据的存储方式，到底是小端字节序还

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

是大端字节序。默认情况下，`DataView`的`get`方法使用大端字节序解读数据，如果需要使用小端字节序解读，必须在`get`方法的第二个参数指定`true`。

```
// 小端字节序
var v1 = dv.getUint16(1, true);

// 大端字节序
var v2 = dv.getUint16(3, false);

// 大端字节序
var v3 = dv.getUint16(3);
```

DataView视图提供8个方法写入内存。

- `setInt8`：写入1个字节的8位整数。
- `setUint8`：写入1个字节的8位无符号整数。
- `setInt16`：写入2个字节的16位整数。
- `setUint16`：写入2个字节的16位无符号整数。
- `setInt32`：写入4个字节的32位整数。
- `setUint32`：写入4个字节的32位无符号整数。
- `setFloat32`：写入4个字节的32位浮点数。
- `setFloat64`：写入8个字节的64位浮点数。

这一系列`set`方法，接受两个参数，第一个参数是字节序号，表示从哪个字节开始写入，第二个参数为写入的数据。对于那些写入两个或两个以上字节的方法，需要指定第三个参数，`false`或者`undefined`表示使用大端字节序写入，`true`表示使用小端字节序写入。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
// 在第1个字节，以大端字节序写入值为25的32位整数
dv.setInt32(0, 25, false);

// 在第5个字节，以大端字节序写入值为25的32位整数
dv.setInt32(4, 25);

// 在第9个字节，以小端字节序写入值为2.5的32位浮点数
dv.setFloat32(8, 2.5, true);
```

如果不确定正在使用的计算机的字节序，可以采用下面的判断方式。

```
var littleEndian = (function() {
  var buffer = new ArrayBuffer(2);
  new DataView(buffer).setInt16(0, 256, true);
  return new Int16Array(buffer)[0] === 256;
})();
```

如果返回`true`，就是小端字节序；如果返回`false`，就是大端字节序。

---

5. 二进制数组的应用

大量的Web API用到了`ArrayBuffer`对象和它的视图对象。

AJAX

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

传统上，服务器通过AJAX操作只能返回文本数据，即`responseType`属性默认为`text`。`XMLHttpRequest`第二版`XHR2`允许服务器返回二进制数据，这时分成两种情况。如果明确知道返回的二进制数据类型，可以把返回类型（`responseType`）设为`arraybuffer`；如果不知道，就设为`blob`。

```
var xhr = new XMLHttpRequest();
xhr.open('GET', someUrl);
xhr.responseType = 'arraybuffer';

xhr.onload = function () {
  let arrayBuffer = xhr.response;
  // ...
};

xhr.send();
```

如果知道传回来的是32位整数，可以像下面这样处理。

```
xhr.onreadystatechange = function () {
  if (req.readyState === 4 ) {
    var arrayResponse = xhr.response;
    var dataView = new DataView(arrayResponse);
    var ints = new Uint32Array(dataView.byteLength / 4);

    xhrDiv.style.backgroundColor = "#00FF00";
    xhrDiv.innerText = "Array is " + ints.length + "uints long";
  }
}
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Canvas

网页`Canvas`元素输出的二进制像素数据，就是TypedArray数组。

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

var imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
var uint8ClampedArray = imageData.data;
```

需要注意的是，上面代码的`uint8ClampedArray`虽然是一个TypedArray数组，但是它的视图类型是一种针对`Canvas`元素的专有类型`Uint8ClampedArray`。这个视图类型的特点，就是专门针对颜色，把每个字节解读为无符号的8位整数，即只能取值0～255，而且发生运算的时候自动过滤高位溢出。这为图像处理带来了巨大的方便。

举例来说，如果把像素的颜色值设为`Uint8Array`类型，那么乘以一个gamma值的时候，就必须这样计算：

```
u8[i] = Math.min(255, Math.max(0, u8[i] * gamma));
```

因为`Uint8Array`类型对于大于255的运算结果（比如`0xFF+1`），会自动变为`0x00`，所以图像处理必须要像上面这样算。这样做很麻烦，而且影响性能。如果将颜色值设为`Uint8ClampedArray`类型，计算就简化许多。

```
pixels[i] *= gamma;
```

`Uint8ClampedArray`类型确保将小于0的值设为0，将大于255的值设为255。注意，

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

IE 10不支持该类型。

WebSocket

`WebSocket`可以通过`ArrayBuffer`，发送或接收二进制数据。

```
var socket = new WebSocket('ws://127.0.0.1:8081');
socket.binaryType = 'arraybuffer';

// Wait until socket is open
socket.addEventListener('open', function (event) {
  // Send binary data
  var typedArray = new Uint8Array(4);
  socket.send(typedArray.buffer);
});

// Receive binary data
socket.addEventListener('message', function (event) {
  var arrayBuffer = event.data;
  // ...
});
```

Fetch API

Fetch API取回的数据，就是`ArrayBuffer`对象。

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
fetch(url)
.then(function(request){
  return request.arrayBuffer()
})
.then(function(arrayBuffer){
  // ...
});
```

File API

如果知道一个文件的二进制数据类型，也可以将这个文件读取为`ArrayBuffer`对象。

```
var fileInput = document.getElementById('fileInput');
var file = fileInput.files[0];
var reader = new FileReader();
reader.readAsArrayBuffer(file);
reader.onload = function () {
  var arrayBuffer = reader.result;
  // ...
};
```

下面以处理bmp文件为例。假定`file`变量是一个指向bmp文件的文件对象，首先读取文件。

```
var reader = new FileReader();
reader.addEventListener("load", processimage, false);
reader.readAsArrayBuffer(file);
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

然后，定义处理图像的回调函数：先在二进制数据之上建立一个`DataView`视图，再建立一个`bitmap`对象，用于存放处理后的数据，最后将图像展示在`Canvas`元素之中。

```
function processimage(e) {
  var buffer = e.target.result;
  var datav = new DataView(buffer);
  var bitmap = {};
  // 具体的处理步骤
}
```

具体处理图像数据时，先处理bmp的文件头。具体每个文件头的格式和定义，请参阅有关资料。

```
bitmap.fileheader = {};
bitmap.fileheader.bfType = datav.getUint16(0, true);
bitmap.fileheader.bfSize = datav.getUint32(2, true);
bitmap.fileheader.bfReserved1 = datav.getUint16(6, true);
bitmap.fileheader.bfReserved2 = datav.getUint16(8, true);
bitmap.fileheader.bfOffBits = datav.getUint32(10, true);
```

接着处理图像元信息部分。

```
bitmap.infoheader = {};
bitmap.infoheader.biSize = datav.getUint32(14, true);
bitmap.infoheader.biWidth = datav.getUint32(18, true);
bitmap.infoheader.biHeight = datav.getUint32(22, true);
bitmap.infoheader.biPlanes = datav.getUint16(26, true);
bitmap.infoheader.biBitCount = datav.getUint16(28, true);
bitmap.infoheader.biCompression = datav.getUint32(30, true);
bitmap.infoheader.biSizeImage = datav.getUint32(34, true);
bitmap.infoheader.biXPelsPerMeter = datav.getUint32(38, true);
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
bitmap.infoheader.biYPelsPerMeter = datav.getUint32(42, true);
bitmap.infoheader.biClrUsed = datav.getUint32(46, true);
bitmap.infoheader.biClrImportant = datav.getUint32(50, true);
```

最后处理图像本身的像素信息。

```
var start = bitmap.fileheader.bfOffBits;
bitmap.pixels = new Uint8Array(buffer, start);
```

至此，图像文件的数据全部处理完成。下一步，可以根据需要，进行图像变形，或者转换格式，或者展示在`Canvas`网页元素之中。

留言

10 Comments ECMAScript 6 入门

Recommend 1 Share

Join the discussion…

silence • 2 months ago

关于TypedArray.from()有一段描述不准确：

“这说明遍历是针对新生成的16位整数数组，而不是针对原来的8位整数数组。也的TypedArray数组，拷贝到另一段内存之中（占用内存从3字节变为6字节），然参照这个例子：

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Uint8Array.from(Int16Array.of(264, 260, 258), x => x / 2)
按文中所说的溢出规则：
“正向溢出（overflow）：当输入值大于当前数据类型的最大值，结果等于当前数
1。”
如果是先拷贝生成目标数组然后再map，那么被拷贝的值就发生了溢出，计算的
[4, 2, 1]
而实际结果却正好相反：
[132, 130, 129]
也就是说，from正确的处理过程是先对源数组进行map运算，然后根据map结果

• Reply • Share ›

ruanyf Mod silence • 2 months ago
谢谢指出，已经改正了。

2016-07-05 0:23 GMT+08:00 Disqus <notifications@disqus.net>:

• Reply • Share ›

大钊 • 3 months ago
建议提一下node里面Buffers 和 TypedArray 的区别和联系

• Reply • Share ›

Stuart Zhang • 5 months ago
我有一个问题是关于《ArrayBuffer与String的相互转换》的。

在您提供的程序例程中，String.fromCharCode()与String.prototype.charAt()函数

我的问题是：是否能够把例程中的

1. String.fromCharCode() ==> String.fromCodePoint()
2. String.prototype.charAt ==> String.prototype.codePointAt()

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

在我的修改之后，《ArrayBuffer与String的相互转换》的例程如下：

```
function ab2str(buf) {
  return String.fromCodePoint.apply(null, new Uint16Array(buf)); // 被修改
}

function str2ab(str) {

  const buf = new ArrayBuffer(str.length * 2);
  const bufView = new Uint16Array(buf);
  for (let i = 0, strLen = str.length; i < strLen; i++) {
    bufView[i] = str.codePointAt(i); // 被修改
  }
  return buf;
}
```

• Reply • Share ›

ruanyf Mod ➚ Stuart Zhang • 4 months ago

String.fromCharCode() 返回值会大于65536的。

• Reply • Share ›

Stuart Zhang ➚ ruanyf • 4 months ago

那么，我使用4个字节表示一个字符的话，是否能够解决Char Co

在下面的代码中，我做了如下两个修改

1. Uint16Array（一字符两字节） ==> Uint32Array（一字符四字节
2. str.length * 2 ==> str.length * 4

在我的修改之后，《ArrayBuffer与String的相互转换》的例程如下

```
function ab2str(buf) {
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
  return String.fromCodePoint.apply(null, new Uint32Array(buf)); //
}

function str2ab(str) {
  const buf = new ArrayBuffer(str.length * 4); // 一个字符对应4个字
  const bufView = new Uint32Array(buf); // 修改 Uint16Array 为 Uin
  for (let i = 0, strLen = str.length; i < strLen; i++) {
    bufView[i] = str.codePointAt(i);
  }
  return buf;
}
```

▲ | ▼ • Reply • Share ›

ruanyf Mod ➦ Stuart Zhang • 4 months ago

这样是可以的。

▲ | ▼ • Reply • Share ›

Leo SC • 6 months ago

谢谢你的好文章

▲ | ▼ • Reply • Share ›

kaiye • a year ago

var let arrayBuffer = xhr.response; 多写了个 var

▲ | ▼ • Reply • Share ›

郭南赐 • a year ago

阮老师，请问有没有canvas3d二进制数据读写方面的具体例子呢

▲ | ▼ • Reply • Share ›

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Subscribe Add Disqus to your site Privacy

上一章 下一章

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com