第四章：使用 Buffer 处理,编码,解码二进制数据

本章内容：

- 为什么需要用缓冲
- 用字符串创建缓冲
- 把缓冲转换成字符串
- 处理缓冲数据
- 缓冲数据的切分和复制

JavaScript很擅长处理字符串，但是因为它最初的设计是用来处理HTML文档，因此它并不太擅长处理二进制数据。JavaScript没有byte类型，没有结构化的类型（structured types），甚至没有字节数组，只有数字和字符串。（原文：JavaScript doesn't have a *byte* type — it just has numbers — or structured types, or even byte arrays: It just has strings.）

因为 Node 基于 JavaScript，它自然可以处理类似 HTTP 这样的文本协议，但是你也可以用它来跟数据库交互，处理图片或文件上传等，可以想象，如果仅仅用字符串来做这些事得有多困难。早些时候，Node 通过将 byte 编码成文本字符来处理二进制数据，但这种方式后来被证明并不可行，既浪费资源，又缓慢，又不灵活，而且难以维护。

Node 有一个二进制缓冲实现 Buffer，这个伪类（pseudo-class）提供了一系列处理二进制数据的 API，简化了那些需要处理二进制数据的任务。缓冲的长度由字节数据的长度决定，而且你可以随机的设置和获取缓冲内的字节数据。

注意：Buffer类有一个特殊的地方，缓冲内的字节数据所占用的内存不是分配在 JavaScrp It VM 内存堆上的，也就是说这些对象不会被 JavaScript 的垃圾回收算法处理，取而代之的是一个不会被修改的永久内存地址，这也避免了因缓冲内容的内存复制所造成的 CPU 浪费。

创建缓冲

你可以用一个 UTF-8 字符串创建缓冲，像这样：

```
var buf = new Buffer('Hello World!');
```

也可以用指定编码的字符串创建缓冲：

```
var buf = new Buffer('8b76fde713ce', 'base64');
```

可接受的字符编码和标识如下：

- *ascii*——ASCI，仅适用于 ASCII 字符集。
- *utf8*——UTF-8，这种可变宽编码适用于 Unicode 字符集的任何字符，它已经成了 Web 世界的首选编码，也是 Node 的默认编码类型。
- *base64*——Base64，这种编码基于 64 个可打印 ASCII 字符来表示二进制数据，Base64 通常用于在字符文档内嵌入可以被转化成字符串的二进制数据，在需要时又可以完整无损的转换回原来的二进制格式。

如果没有数据来初始化缓冲，可以用指定的容量大小来创建一个空缓冲：

```
var buf = new Buffer(1024); //  创建一个 1024 字节的缓冲
```

获取和设置缓冲数据

创建或接收一个缓冲对象后，你可能要查看或者修改它的内容，可以通过[]操作符来访问缓冲的某个字节：

```
var buf = new Buffer('my buffer content');
//  访问缓冲内第10个字节
console.log(buf[10]);  // -> 99
```

注意：当你（使用缓冲容量大小来）创建一个已初始化的缓冲时，一定要注意，缓冲的数据并没有被初始化成 0，而是随机数据。

```
var buf = new Buffer(1024);
console.log(buf[100]); // -> 5 (某个随机值)
```

你可以这样修改缓冲里任何位置的数据：

```
buf[99] = 125; //  把第 100 个字节的值设置为 125
```

注意：在某些情况下，一些缓冲操作并不会产生错误，比如：

- **缓冲内的字节最大值为 255，如果某个字节被赋予大于 256 的数字，将会用 256 对其取模，然后将结果赋给这个字节。**
- **如果将缓冲的某个字节赋值为 256，它的实际值将会是 0（译者注：其实跟第一条重复，256%256=0）**
- **如果用浮点数给缓冲内某个字节赋值，比如 100.7，实际值将会是浮点数的整数部分——100**
- **如果你尝试给一个超出缓冲容量的位置赋值，赋值操作将会失败，缓冲不做任何修改。**

你可以用 length 属性获取缓冲的长度：

```
var buf = new Buffer(100);
console.log(buf.length); // -> 100
```

还可以使用缓冲长度迭代缓冲的内容，来读取或设置每个字节：

```
var buf = new Buffer(100);
for(var i = 0; i < buf.length; i++) {
    buf[i] = i;
}
```

上面代码新建了一个包含 100 个字节的缓冲，并从 0 到 99 设置了缓冲内每个字节。

切分缓冲数据

一旦创建或者接收了一个缓冲，你可能需要提取缓冲数据的一部分，可以通过指定起始位置来切分现有的缓冲，从而创建另外一个较小的缓冲：

```
var buffer = new Buffer("this is the content of my buffer");
var smallerBuffer = buffer.slice(8, 19);
console.log(smallerBuffer.toString()); // -> "the content"
```

注意，当切分一个缓冲的时候并没有新的内存被分配或复制，新的缓冲使用父缓冲的内

存，它只是父缓冲某段数据（由起始位置指定）的引用。这段话含有几个意思。

首先，如果你的程序修改了父缓冲的内容，这些修改也会影响相关的子缓冲，因为父缓冲和子缓冲是不同的 JavaScript 对象，因此很容易忽略这个问题，并导致一些潜在的 bug。

其次，当你用这种方式从父缓冲创建一个较小的子缓冲时，父缓冲对象在操作结束后依然会被保留，并不会被垃圾回收，如果不注意的话，很容易会造成内存泄露。

注意：如果你担心因此产生内存泄露问题，你可以使用 copy 方法来替代 slice 操作，下面将会介绍 copy。

复制缓冲数据

你可以像这样用 copy 将缓冲的一部分复制到另外一个缓冲：

```
var buffer1 = new Buffer("this is the content of my buffer");
var buffer2 = new Buffer(11);
var targetStart = 0;
var sourceStart = 8;
var sourceEnd = 19;
buffer1.copy(buffer2, targetStart, sourceStart, sourceEnd);
console.log(buffer2.toString()); // -> "the content"
```

上面代码，复制源缓冲的第 9 到 20 个字节到目标缓冲的开始位置。

解码缓冲数据

缓冲数据可以这样转换成一个 UTF-8 字符串：

```
var str = buf.toString();
```

还可以通过指定编码类型来将缓冲数据解码成任何编码类型的数据。比如，你想把一个缓冲解码成base64字符串，可以这么做：

```
var b64Str = buf.toString("base64");
```

使用toString函数，你还可以把一个UTF-8字符串转码成base64字符串：

```
var utf8String = 'my string';
var buf = new Buffer(utf8String);
var base64String = buf.toString('base64')
```

小结

有时候，你不得不跟二进制数据打交道，但是原生 JavaScript 又没有明确的方式来做这件事，于是 Node 提供了 Buffer 类，封装了一些针对连续内存块的操作。你可以在两个缓冲之间切分或复制内存数据。

你也可以把一个缓冲转换成某种编码的字符串，或者反过来，把一个字符串转化成缓冲，来访问或处理每个 bit。

译者注：

本文对应原文第二部分第三章：Node Core API Basics：Using Buffers to Manipulate,Encode, and Decode Binary Data

本系列文章列表和翻译进度，请移步：Node.js 高级编程：用 Javascript 构建可伸缩应用（〇）