Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Lập trình ứng dụng web bằng NodeJS

Bài 4. ***Buffer và Stream***

http://csc.edu.vn/

2017

Nội dung

1. **Buffer là gì?**
2. Tạo buffer trong NodeJS
3. Sử dụng buffer
4. Stream là gì?
5. Cách sử dụng stream
6. Piping Stream
7. Chaning Stream

1. Buffer là gì?

- Buffer là một vùng nhớ tạm thời mà có thể lưu trữ thông tin trong khi đang xử lý các thông tin khác.

- Khi làm việc với các luồng TCP hoặc hệ thống file, cần thiết phải xử lý các luồng dữ liệu bát phân. Node.js cung cấp các lớp Buffer cho phép lưu trữ các dữ liệu thô như một mảng các số nguyên tương ứng với phần cấp phát bộ nhớ thô bên ngoài.

- Các lớp Buffer trong Node.js là các lớp toàn cục và có thể được truy cập trong ứng dụng mà không cần khai báo các Buffer Module bởi phương thức require() như các Module khác.

Nội dung

1. Buffer là gì?
2. **Tạo buffer trong NodeJS**
3. Sử dụng buffer
4. Stream là gì?
5. Cách sử dụng stream
6. Piping Stream
7. Chaning Stream

2. Tạo buffer trong NodeJS

- Tạo một Buffer có 10 phần tử

```
buf1 = Buffer(10);
```

- Tạo một Buffer từ mảng:

```
buf2 = Buffer([1,2,3,4,5,6,7,8,9999]);
```

- Tạp một buffer có toàn bộ phần tử là 0:

```
buf1 = Buffer.alloc(10);
```

2. Tạo buffer trong NodeJS

- Tạo một buffer từ một chuỗi

```
buf3 = Buffer("Chào mừng các bạn đến với NodeJS", "utf-8");
```

- Với Buffer tạo từ một chuỗi, tham số thứ 2 trong hàm khởi tạo Buffer chính là kiểu mã hóa dữ liệu chuỗi

- Bao gồm các bảng mã: ASCII, UTF-8, BASE64

Nội dung

1. Buffer là gì?
2. Tạo buffer trong NodeJS
3. **Sử dụng buffer**
4. Stream là gì?
5. Cách sử dụng stream
6. Piping Stream
7. Chaning Stream

3. Sử dụng buffer

- Ghi dữ liệu vào buffer
- Đọc dữ liệu từ Buffer
- Ghép nối các Buffer
- So sánh Buffer
- Sao chép buffer
- Chia nhỏ Buffer

3.1 Ghi dữ liệu vào Buffer

- **Để ghi dữ liệu vào buffer, ta dùng phương thức write trong đối tượng buffer đã tạo.**

- **Cấu trúc lệnh:**

  Tên_đối_tượng.write(string[, offset][, length][, encoding])

- **Trong phương thức write có 4 tham số như sau:**
  - String: là một dữ liệu dạng chuỗi được ghi tới buffer.
  - Offset: là chỉ mục để buffer bắt đầu ghi tại đó. Giá trị mặc định là số 0
  - Length: Số lượng các byte được ghi. Mặc định là buffer.length.
  - Encoding: Mã hóa được sử dụng. "utf8" là mã hóa mặc định.

3.1 Ghi dữ liệu vào Buffer

- Ví dụ:

```
buf.write("NODEJS");
```

- Ví dụ hoàn chỉnh:

```
buf = new Buffer(256);
len = buf.write("NODEJS");

console.log(buf);
console.log("Tổng số byte đã ghi vào buffer : "+  len);
```

- Lưu ý: nếu số byte của buffer không đủ chứa chuỗi thì nó sẽ chỉ ghi một phần của chuỗi đó.

3.2 Đọc dữ liệu từ Buffer

- **Để đọc dữ liệu trong buffer chúng ta sử dụng phương thức toString. Cấu trúc như sau:**

  Tên_đối_tượng.toString([encoding][, start][, end])

- **Các tham số bao gồm:**
  - Encoding: Mã hóa buffer sử dụng, mặc định là utf-8
  - Start: vị trí bắt đầu đọc buffer, mặc định là 0
  - End: vị trí kết thúc đọc buffer, mặc định là độ dài của đối tượng buffer hiện hành

3.2 Đọc dữ liệu từ Buffer

- Ví dụ:

```
buf.toString('utf-8');
```

- Ví dụ hoàn chỉnh:

```
buf = new Buffer(16);
for (var i = 0 ; i <= 15 ; i++) {
  buf[i] =  i + 97;
}

console.log(buf.toString('ascii'));
console.log(buf.toString('utf-8'));
console.log(buf.toString('utf-8',0,5));
```

3.3 ghép nối các buffer

- Để ghép nối các buffer lại với nhau, NodeJS cung cấp phương thức concat với cấu trúc:

```
Buffer.concat(list[, totalLength])
```

- Trong đó list là mảng các buffer cần được ghép nối lại với nhau

- totalLength là tổng độ dài cho phép của buffer sau khi ghép nối. Nếu không truyền vào tham số này, mặc định sẽ tạo ra buffer có độ dài bằng tổng độ dài tất cả buffer trong mảng

3.3 ghép nối các buffer

- Ví dụ:

```
buf3 = Buffer.concat([buf1, buf2]);
```

- Ví dụ hoàn chỉnh:

```
buf1 = new Buffer("Xin chào các bạn<br/>");
buf2 = new Buffer("Đến với lập trình NodeJS");

buf3 = Buffer.concat([buf1, buf2]);

console.log(buf3.toString("utf-8"));
```

3.4 So sánh Buffer

- Để so sánh các buffer xem buffer nào có bộ mã được sắp xếp sau (dựa trên bảng mã ASCII) thì chúng ta dung phương thức compare

- Ví dụ:

```
var result = buffer1.compare(buffer2);
```

- Ví dụ hoàn chỉnh:

```
var buffer1 = new Buffer('a');
var buffer2 = new Buffer('A');

var result = buffer1.compare(buffer2);

if(result < 0) {
   console.log(buffer1 +" dung truoc " + buffer2);
}else if(result == 0){
   console.log(buffer1 +" cung thu tu voi " + buffer2);
}else {
   console.log(buffer1 +" dung sau " + buffer2);
}
```

3.5 Sao chép Buffer

- **Sao chép Buffer chúng ta sẽ sử dụng phương thức copy như sau:**

  Buffer.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd])

  - targetBuffer: là Buffer được sao chép ra
  - targetStart: vị trí bắt đầu ghi sao chép tính theo buffer được copy ra, mặc định là 0
  - sourceStart: vị trí lấy dữ liệu ghi vào tính theo buffer dùng để copy
  - sourceEnd: vị trí ngừng lấy dữ liệu ghi vào tính theo buffer dung để copy

3.5 Sao chép Buffer

- Ví dụ:

```
buf.copy(buf2, 8, 8, 10);
```

- Ví dụ hoàn chỉnh:

```
const buf = Buffer.alloc(26);
const buf2 = Buffer.alloc(26);
const buf3 = Buffer.alloc(26);

for (let i = 0 ; i < 26 ; i++) {
  buf[i] = i + 97;
}

buf.copy(buf2, 8, 8, 10);
buf.copy(buf3, 0, 8, 10);

console.log(buf2.toString());
console.log(buf3.toString());
```

3.6 Chia nhỏ Buffer

- Chia nhỏ buffer trong NodeJS có cung cấp phương thức slice, cấu trúc như sau:

  Buffer.slice([start[, end]])

- start: vị trí bắt đầu cắt, mặc định là 0

- end: vị trí kết thúc cắt, nếu không nhập vào NodeJS sẽ cắt từ vị trí bắt đầu đến hết buffer.

3.6 Chia nhỏ Buffer

- Ví dụ:

```
var buf2 = buf1.slice(0,8);
```

- Ví dụ hoàn chỉnh:

```
var buf1 = new Buffer('Hôm nay em đi học');

var buf2 = buf1.slice(0,8);
var buf3 = buf1.slice(9);

console.log("Noi dung cua buf2 la: " + buf2.toString());
console.log("Noi dung cua buf3 la: " + buf3.toString());
```

Nội dung

1. Buffer là gì?
2. Tạo buffer trong NodeJS
3. Sử dụng buffer
4. **Stream là gì?**
5. Cách sử dụng stream
6. Piping Stream
7. Chaning Stream

4. Stream là gì?

- **Stream là một luồng dữ liệu.**
- **Trong NodeJS, có rất nhiều module được xây dựng có sử dụng stream để thao tác đọc, ghi dữ liệu một cách dễ dàng.**
- **Hiện có 4 loại Stream trong NodeJS như:**
  - Readable: Stream dùng cho hoạt động đọc
  - Writeable: Stream dung cho hoạt động ghi
  - Duplex: dùng cho cả hoạt động ghi và đọc
  - Transform: dung là Duplex Stream nhưng dữ liệu đầu ra sẽ có thể hiệu chỉnh, thay đổi so với dữ liệu đầu vào

Nội dung

1. Buffer là gì?
2. Tạo buffer trong NodeJS
3. Sử dụng buffer
4. Stream là gì?
5. **Cách sử dụng stream**
6. Piping Stream
7. Chaning Stream

5. Cách sử dụng Stream

- **Với đối tượng Stream thường có 4 loại event sau được phát ra khi Stream hoạt động:**
  - Data: Sự kiện phát ra khi dữ liệu đã sẵn sàng cho hoạt động đọc dữ liệu
  - End: Sự kiện sẽ phát ra khi đã đọc xong dữ liệu
  - Error: Sự kiện phát ra nếu trong quá trình xử lý mà xảy ra lỗi
  - Finish: Sự kiện hoàn thành công việc chuyển tất cả dữ liệu đến vùng lưu trữ.

5. Cách sử dụng Stream

- Chúng ta sẽ tạm thời lấy ví dụ đọc, ghi dữ liệu tập tin để minh họa cho cách sử dụng stream trong NodeJS.

- Trước hết để đọc, ghi tập tin trong NodeJS cần phải có thư viện “fs” (viết tắt của từ FileSystem) để thực thi với cấu trúc sau:

```
var fs = require("fs");
```

5. Cách sử dụng Stream

- Tạo một Readable stream với dữ liệu từ tập tin nhờ vào thư viện "fs", với ví dụ như sau:

```
var readerStream = fs.createReadStream('gioi_thieu.txt');
```

- Đọc dữ liệu từ Stream đã có bằng cách sau:

```
readerStream.on('data', function(du_lieu) {
   data += du_lieu;
});

readerStream.on('end',function(){
   console.log(data);
});
```

5. Cách sử dụng Stream

- Để ghi dữ liệu vào một tập tin từ Stream ta sẽ tạo một StreamWriter như sau:

```
var writerStream = fs.createWriteStream('output.txt');
```

- Ghi dữ liệu vào tập tin với ví dụ:

```
//ghi dữ liệu vào với mã hóa utf-8
writerStream.write(data,'UTF8');

//đánh dấu kết thúc tập tin
writerStream.end();
```

Nội dung

1. Buffer là gì?
2. Tạo buffer trong NodeJS
3. Sử dụng buffer
4. Stream là gì?
5. Cách sử dụng stream
6. **Piping Stream**
7. Chaning Stream

6. Piping Stream

- Piping Stream là một kỹ thuật xử lý kết quả đầu ra của Stream sẽ là kết quả đầu vào của một Stream khác.

- Để sử dụng kỹ thuật Piping Stream, trong NodeJS có cung cấp cho chúng ta một phương thức là pipe.

6. Piping Stream

- **Ví dụ cấu trúc code phương thức pipe:**

```
readerStream.pipe(writerStream);
```

  - Với ví dụ này dữ liệu đọc từ ReaderStream sẽ được lưu vào writerStream.

- **Ví dụ hoàn chỉnh:**

```
var readerStream = fs.createReadStream('gioi_thieu.txt');

var writerStream = fs.createWriteStream('output.txt');

readerStream.pipe(writerStream);
```

Nội dung

1. Buffer là gì?
2. Tạo buffer trong NodeJS
3. Sử dụng buffer
4. Stream là gì?
5. Cách sử dụng stream
6. Piping Stream
7. **Chaning Stream**

7. Channing Stream

- Chaining là một kỹ thuật để kết nối kết quả đầu ra của một Stream tới một Stream khác và tạo một chuỗi bao gồm nhiều hoạt động Stream. Thường thì nó được sử dụng với các hoạt động Piping.

- Để ví dụ rõ ràng hơn, chúng ta sẽ dùng thư viện “fs” đọc file và thư viện “zlip” nén file để chạy 2 luồng Stream kết hợp với nhau.

7. Chaning Stream

- Ví dụ tường minh như sau:

```
fs.createReadStream('gioi_thieu.txt')
  .pipe(zlib.createGzip())
  .pipe(fs.createWriteStream('file_nen.txt.zip'));
```

- Kết là toàn bộ nội dung của file gioi_thieu.txt được nén thành file_nen.txt.zip

Thảo luận