Stream in Node.js

Streams are objects that allow you to read data from one source and record data to a destination. In Node.js, there are 4 types of Streams.

What is a stream?

Streams are objects that allow you to read data from one source and record data to a destination. In Node.js, there are 4 types of Streams.

Readable - Is the Stream used for reading operations

Writable - Is the Stream used for recording

Duplex - Is Stream used for both recording and reading purposes

Transform - This is a Duplex Stream type, in that the output is calculated based on the data you have entered.

Each Stream type is an instance of the **EventEmitter** object and throws several events at different times. The following list lists some commonly used events:

data - This event is triggered when data is available for read operation.

end - This event is triggered when there is no more data to read.

error - This event is triggered when any error occurs in reading and writing data.

finish - This event is activated when all data has been transferred to the base system area.

In the next section, I will detail the activities commonly used on Streams.

Read data from Stream in Node.js

First, you create a text file named input.txt with the following content:

```
QTM is a Web page that contains all the scripts
Welcome to the world !!!!!!
```

Create a js file named main.js. In this file, you first declare **fs** Module (this is the Module for File I / O operations) using the require () method. Then using the createReadStream () method takes the parameter that is the name of the text file you created earlier to read the data from.

```
var fs = require ( "fs" );  var data = '' ;  // Create a Readable Stream var re
```

Now run main.js to see the result:

```
$ node main . js
```

Check output:

```
Let's get married
QTM is a Web page that contains all the scripts
Welcome to the world !!!!!!
```

Write data to Stream in Node.js

You also created main.js as above. The difference is that instead of using createReadStream (), you use the createWriteStream () method to get the file to contain the results you need to record:

```
var fs = require ( "fs" );  var data = 'VietNamVoDoi' ;  // Create Writable mo
```

Now run main.js to see the result:

```
$ node main . js
```

Check the result:

```
Let's get married
Please write.
```

Now, open the generated output.txt in the current directory and check the resulting content:

```
VietNamVoDoi
```

Concept of Piping Stream in Node.js

Piping is a technique. With this technique, we provide the output of a Stream to make input for another Stream. There are no restrictions on this Piping activity, ie the process may continue.

To understand more about this concept, you follow the example below. In this example, I read the data from a file, then write that data to another file.

First, create js file named main.js. In this file, you use the two methods discussed above to create the corresponding createReadStream () and createWriteStream () for reading and writing data. Next, use the pipe () method to implement the Piping Stream technique as follows:

```
var fs = require ( "fs" );  // Create a Readable Stream var readerStream = fs
```

Run main.js to see the result:

```
$ node main . js
```

Check the result:

```
Let's get married
```

Open the generated output.txt in your current directory and check the content:

```
QTM is a Web page that contains all the scripts
Welcome to the world !!!!!!
```

Chaining Stream Concept in Node.js

Chaining is a technique for connecting the output of a Stream to another Stream and creating a series of multiple Stream operations. Often it is used with Piping operations.

The following example illustrates how to combine two Piping and Chaining operations. First we compress a file, then extract the file.

Create main.js. In this file, I need to declare the **zlip** Module that provides the createGzip () method for compression.

```
var fs = require ( "fs" );  var zlib = require ( 'zlib' );  // Nen input input
```

Run main.js to see the result:

```
$ node main . js
```

Check the result:

```
The file should be curved.
```

After checking, you will see that input.txt has been compressed and it has created an input.txt.gz in the current directory. Now, try to extract the same file using the createGunzip () method of **zlib** Module as follows:

```
var fs = require ( "fs" );  var zlib = require ( 'zlib' );  // Phase input.txt
```

Run main.js to see the result:

```
$ node main . js
```

Check the result:

```
In the File Bar.
```

According to Tutorialspoint

Previous article: Concept of Buffer in Node.js

Next article: Read the File record in Node.js