Stream

王红元
coderwhy

实力IT教育

认识Stream

- 什么是流呢？
  - 我们的第一反应应该是流水，源源不断的流动；
  - 程序中的流也是类似的含义，我们可以想象当我们从一个文件中读取数据时，文件的二进制（字节）数据会源源不断的被读取到我们程序中；
  - 而这个一连串的字节，就是我们程序中的流；
- 所以，我们可以这样理解流：
  - 是连续字节的一种表现形式和抽象概念；
  - 流应该是可读的，也是可写的；
- 在之前学习文件的读写时，我们可以直接通过 readFile或者 writeFile方式读写文件，为什么还需要流呢？
  - 直接读写文件的方式，虽然简单，但是无法控制一些细节的操作；
  - 比如从什么位置开始读、读到什么位置、一次性读取多少个字节；
  - 读到某个位置后，暂停读取，某个时刻恢复读取等等；
  - 或者这个文件非常大，比如一个视频文件，一次性全部读取并不合适；

文件读写的Stream

- 事实上Node中很多对象是基于流实现的：
  - http模块的Request和Response对象；
  - process.stdout对象；
- 官方：另外所有的流都是EventEmitter的实例：
- Node.js中有四种基本流类型：
  - Writable：可以向其写入数据的流（例如 fs.createWriteStream()）。
  - Readable：可以从中读取数据的流（例如 fs.createReadStream()）。
  - Duplex：同时为Readable和的流Writable（例如 net.Socket）。
  - Transform：Duplex可以在写入和读取数据时修改或转换数据的流（例如zlib.createDeflate()）。
- 这里我们通过fs的操作，讲解一下Writable、Readable，另外两个大家可以自行学习一下。

Readable

- 之前我们读取一个文件的信息：

```
fs.readFile('./foo.txt', (err, data) => {
  console.log(data);
})
```

- 这种方式是一次性将一个文件中所有的内容都读取到程序（内存）中，但是这种读取方式就会出现我们之前提到的很多问题：
  - 文件过大、读取的位置、结束的位置、一次读取的大小；
- 这个时候，我们可以使用 createReadStream，我们来看几个参数，更多参数可以参考官网：
  - start：文件读取开始的位置；
  - end：文件读取结束的位置；
  - highWaterMark：一次性读取字节的长度，默认是64kb；

Readable的使用

- 创建文件的Readable

```
const read = fs.createReadStream("./foo.txt", {
  start: 3,
  end: 8,
  highWaterMark: 4
});
```

- 我们如何获取到数据呢？
  - 可以通过监听data事件，获取读取到的数据；

```
read.on("data", (data) => {
  console.log(data);
});
```

- 也可以做一些其他的操作：监听其他事件、暂停或者恢复

```
read.on('open', (fd) => {
  console.log("文件被打开");
})

read.on('end', () => {
  console.log("文件读取结束");
})

read.on('close', () => {
  console.log("文件被关闭");
})
```

```
read.pause();
setTimeout(() => {
  read.resume();
}, 2000);
```

Writable

- 之前我们写入一个文件的方式是这样的：

```
fs.writeFile('./foo.txt', "内容", (err) => {
  ..

});
```

- 这种方式相当于一次性将所有的内容写入到文件中，但是这种方式也有很多问题：
  - 比如我们希望一点点写入内容，精确每次写入的位置等；
- 这个时候，我们可以使用 createWriteStream，我们来看几个参数，更多参数可以参考官网：
  - flags：默认是w，如果我们希望是追加写入，可以使用 a或者 a+；
  - start：写入的位置；

Writable的使用

- 我们进行一次简单的写入

```
const writer = fs.createWriteStream("./foo.txt", {
  flags: "a+",
  start: 8
});

writer.write("你好啊", err => {
  console.log("写入成功");
});
```

- 你可以监听open事件：

```
writer.on("open", () => {
  console.log("文件打开");
})
```

close的监听

- 我们会发现，我们并不能监听到 close 事件：
  - 这是因为写入流在打开后是不会自动关闭的；
  - 我们必须手动关闭，来告诉Node已经写入结束了；
  - 并且会发出一个 finish 事件的；
- 另外一个非常常用的方法是 end：end方法相当于做了两步操作： write传入的数据和调用close方法；

```
writer.close();

writer.on("finish", () => {
  console.log("文件写入结束");
})

writer.on("close", () => {
  console.log("文件关闭");
})
```

```
writer.end("Hello World");
```

pipe方法

- 正常情况下，我们可以将读取到的 输入流，手动的放到 输出流中进行写入：

```
const reader = fs.createReadStream('./foo.txt');
const writer = fs.createWriteStream('./bar.txt');

reader.on("data", (data) => {
  console.log(data);
  writer.write(data, (err) => {
    console.log(err);
  });
});
```

- 我们也可以通过pipe来完成这样的操作：

```
reader.pipe(writer);
```