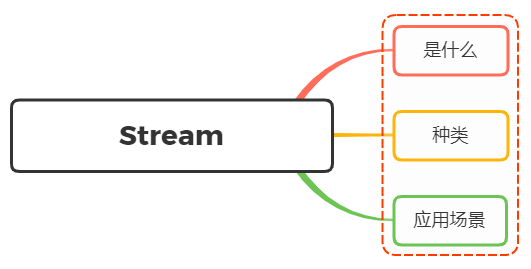
# 面试官：说说对 Node 中的 Stream 的理解？应用场景？



## 一、是什么

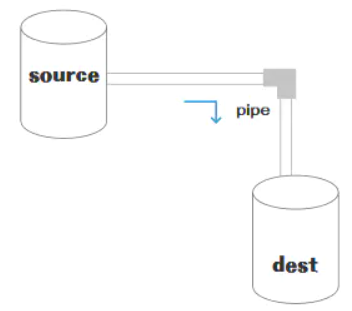
流（Stream），是一个数据传输手段，是端到端信息交换的一种方式，而且是有顺序的,是逐块读取数据、处理内容，用于顺序读取输入或写入输出

Node.js中很多对象都实现了流，总之它是会冒数据（以 Buffer 为单位）

它的独特之处在于，它不像传统的程序那样一次将一个文件读入内存，而是逐块读取数据、处理其内容，而不是将其全部保存在内存中

流可以分成三部分：source、dest、pipe

在source和dest之间有一个连接的管道pipe,它的基本语法是source.pipe(dest)，source和dest就是通过pipe连接，让数据从source流向了dest，如下图所示：



## 二、种类

在NodeJS，几乎所有的地方都使用到了流的概念，分成四个种类：

* 可写流：可写入数据的流。例如 fs.createWriteStream() 可以使用流将数据写入文件
* 可读流： 可读取数据的流。例如fs.createReadStream() 可以从文件读取内容
* 双工流： 既可读又可写的流。例如 net.Socket
* 转换流： 可以在数据写入和读取时修改或转换数据的流。例如，在文件压缩操作中，可以向文件写入压缩数据，并从文件中读取解压数据

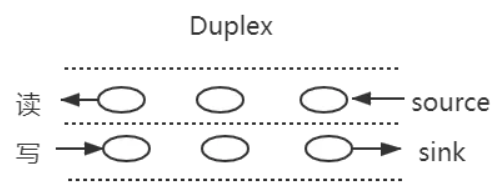
在NodeJS中HTTP服务器模块中，request 是可读流，response 是可写流。还有fs 模块，能同时处理可读和可写文件流

可读流和可写流都是单向的，比较容易理解，而另外两个是双向的

### 双工流

之前了解过websocket通信，是一个全双工通信，发送方和接受方都是各自独立的方法，发送和接收都没有任何关系

如下图所示：

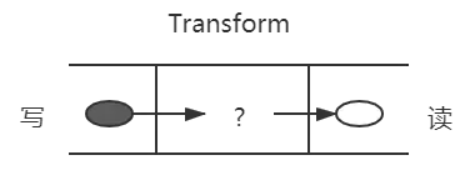


基本代码如下：

const { Duplex } = require('stream');  
  
const myDuplex = new Duplex({  
 read(size) {  
 // ...  
 },  
 write(chunk, encoding, callback) {  
 // ...  
 }  
});

### 双工流

双工流的演示图如下所示：



除了上述压缩包的例子，还比如一个 babel，把es6转换为，我们在左边写入 es6，从右边读取 es5

基本代码如下所示：

const { Transform } = require('stream');  
  
const myTransform = new Transform({  
 transform(chunk, encoding, callback) {  
 // ...  
 }  
});

## 三、应用场景

stream的应用场景主要就是处理IO操作，而http请求和文件操作都属于IO操作

试想一下，如果一次IO操作过大，硬件的开销就过大，而将此次大的IO操作进行分段操作，让数据像水管一样流动，直到流动完成

常见的场景有：

* get请求返回文件给客户端
* 文件操作
* 一些打包工具的底层操作

### get请求返回文件给客户端

使用stream流返回文件，res也是一个stream对象，通过pipe管道将文件数据返回

const server = http.createServer(function (req, res) {  
 const method = req.method; // 获取请求方法  
 if (method === 'GET') { // get 请求  
 const fileName = path.resolve(\_\_dirname, 'data.txt');  
 let stream = fs.createReadStream(fileName);  
 stream.pipe(res); // 将 res 作为 stream 的 dest  
 }  
});  
server.listen(8000);

### 文件操作

创建一个可读数据流readStream，一个可写数据流writeStream，通过pipe管道把数据流转过去

const fs = require('fs')  
const path = require('path')  
  
// 两个文件名  
const fileName1 = path.resolve(\_\_dirname, 'data.txt')  
const fileName2 = path.resolve(\_\_dirname, 'data-bak.txt')  
// 读取文件的 stream 对象  
const readStream = fs.createReadStream(fileName1)  
// 写入文件的 stream 对象  
const writeStream = fs.createWriteStream(fileName2)  
// 通过 pipe执行拷贝，数据流转  
readStream.pipe(writeStream)  
// 数据读取完成监听，即拷贝完成  
readStream.on('end', function () {  
 console.log('拷贝完成')  
})

### 一些打包工具的底层操作

目前一些比较火的前端打包构建工具，都是通过node.js编写的，打包和构建的过程肯定是文件频繁操作的过程，离不来stream，如gulp

## 参考文献

* https://xie.infoq.cn/article/1a9695020828460eb3c4ff1fa
* https://juejin.cn/post/6844903891083984910