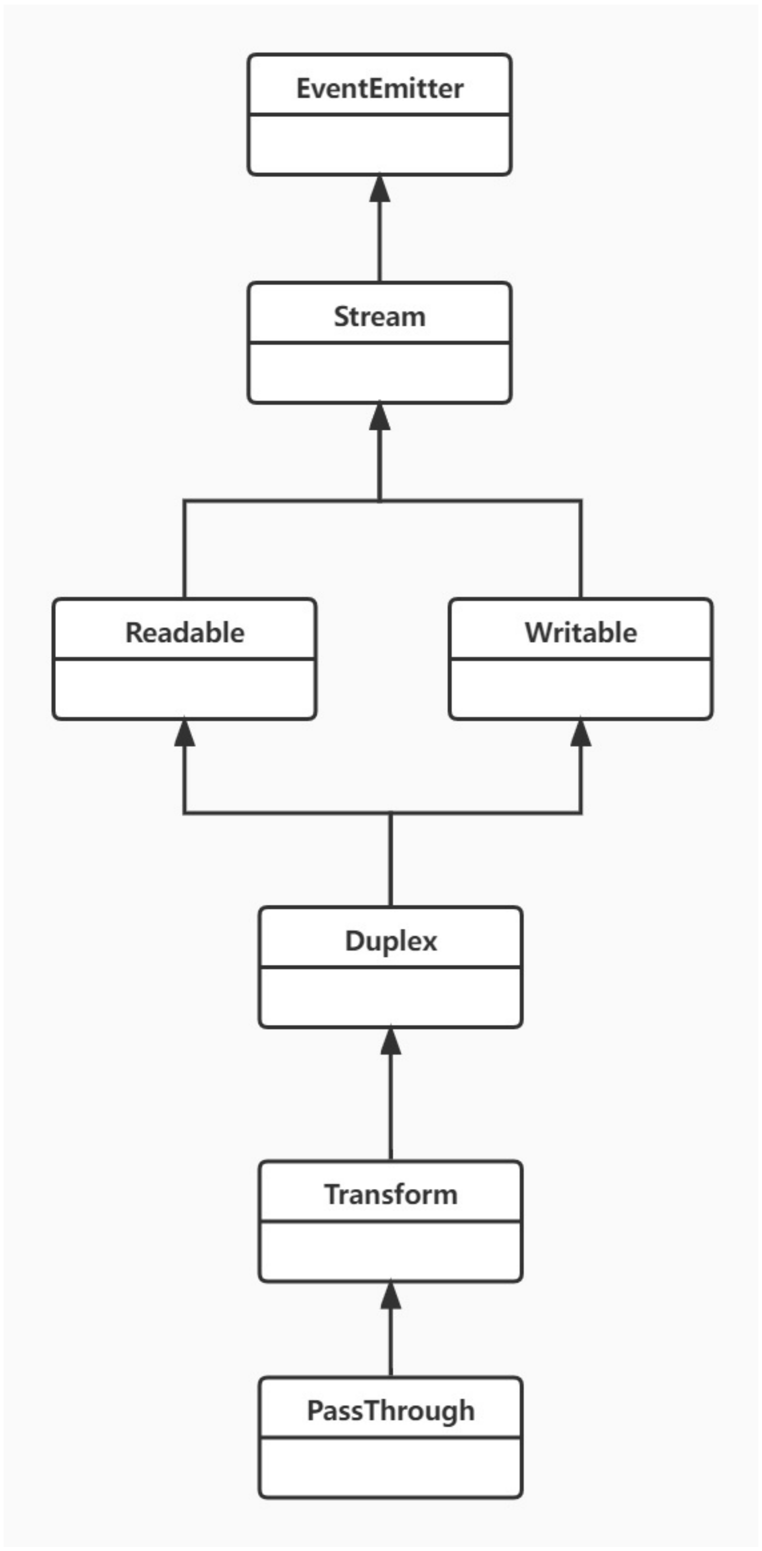
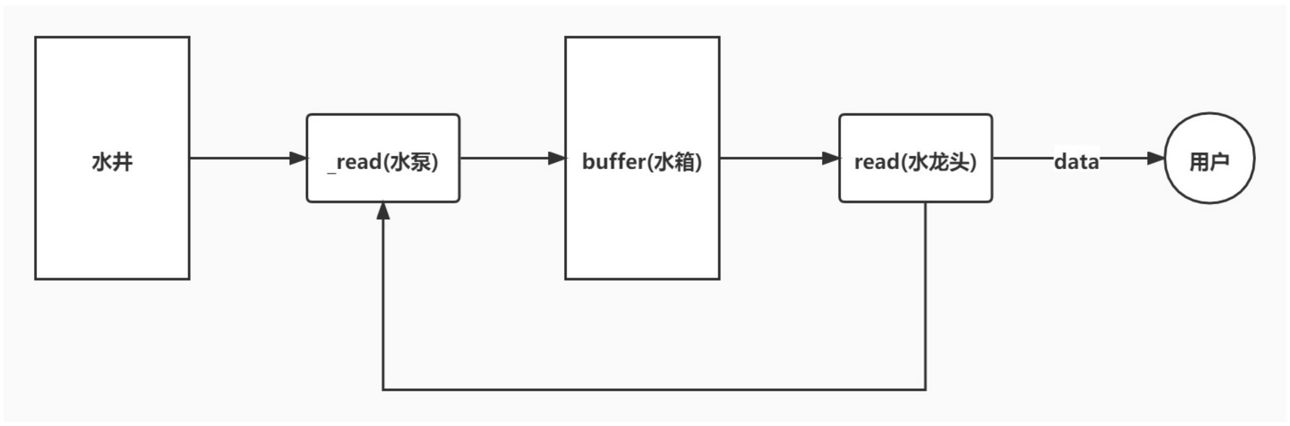
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=90 sentences=394, words=1846

1.流的分类

- Readable 可读流
- Writable 可写流
- Duplex 双工流
- Transform 转换流
- PassThrough 传递流



2.Readable(可读流)



2.1 1.readableStream.js

1.readableStream.js

```
const readableStream = require('./readableStream');
readableStream.on('data', (data) => {
    console.log(data);
    readableStream.pause();
});
```

2.2 readableStream.js

readableStream.js

```
const Readable = require('./Readable');
const readableIterator = (function (count) {
    return {
        next() {
            count++;
            if (count 5) {
                return { done: false, value: count + '' };
            } else {
                return { done: true, value: null }
            }
        }
    }
})(0)

const readableStream = new Readable({
    read() {
        let { done, value } = readableIterator.next();
        if (done) {
            this.push(null);
        } else {
            this.push(value);
        }
    }
});
module.exports = readableStream;
```

2.3 Readable.js

Readable.js

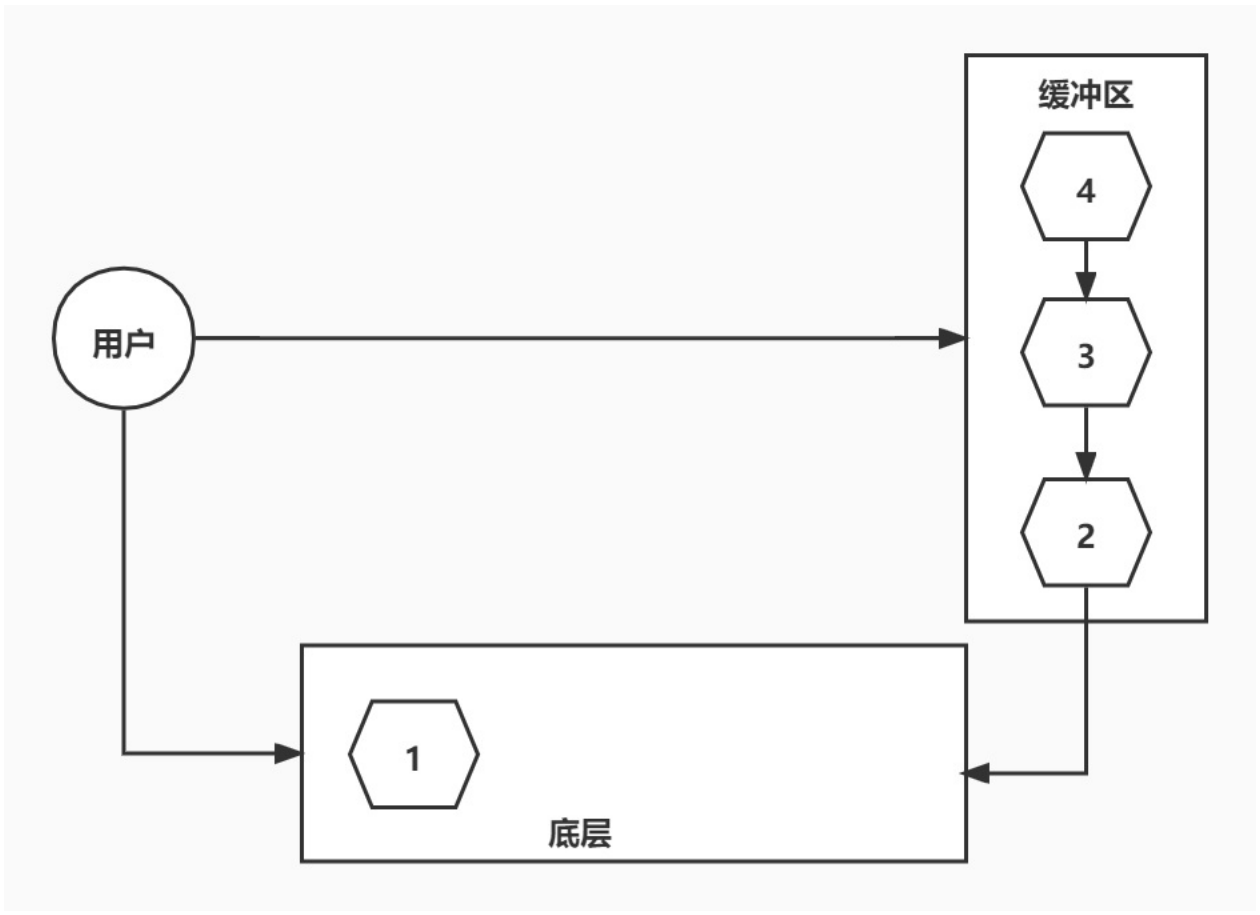
```
const Stream = require('./Stream');
var { inherits } = require('util');
function Readable(options) {
    Stream.call(this, options);
    this._readableState = { ended: false, buffer: [], flowing: false };
    if (options.read) this._read = options.read;
}
inherits(Readable, Stream);
Readable.prototype.on = function (event, fn) {
    Stream.prototype.on.call(this, event, fn);
    if (event === 'data') {
        this.resume();
    }
}
Readable.prototype.resume = function () {
    this._readableState.flowing = true;
    while (this.read());
}
Readable.prototype.pause = function () {
    this._readableState.flowing = false;
}
Readable.prototype.read = function () {
    if (!this._readableState.ended && this._readableState.flowing) {
        this._read();
    }
    let data = this._readableState.buffer.shift();
    if (data) {
        this.emit('data', data);
    }
    return data;
}
Readable.prototype.push = function (chunk) {
    if (chunk === null) {
        this._readableState.ended = true;
    } else {
        this._readableState.buffer.push(chunk);
    }
}
module.exports = Readable;
```

2.4 Stream.js

Stream.js

```
const EventEmitter = require('events');
var { inherits } = require('util');
function Stream(options) {
     this.options = options;
    EventEmitter.call(this);
}
inherits(Stream, EventEmitter);
module.exports = Stream;
```

3.Writable(可写流)



3.1 基本实现

3.1.1 2.writableStream.js

2.writableStream.js

```
let writableStream = require('./writableStream');
writableStream.write('1');
writableStream.write('2');
writableStream.write('3');
writableStream.write('4');
writableStream.write('5');
writableStream.end();
```

3.1.2 writableStream.js

writableStream.js

```
const Writable = require('./Writable');
const writableStream = new Writable({
    write(data, encoding, next) {
        console.log(data.toString(encoding));
        setTimeout(next, 1000);
    }
});
module.exports = writableStream;
```

3.1.3 Writable.js

Writable.js

```
const Stream = require('./Stream');
var { inherits } = require('util');
function Writable(options) {
    Stream.call(this, options);
    this._writableState = {
        ended: false,
        writing: false,
        buffer: []
    };
    if (options.write) this._write = options.write;
}
inherits(Writable, Stream);
Writable.prototype.write = function (chunk) {
    if (this._writableState.ended) {
        return;
    }
    if (this._writableState.writing) {
        this._writableState.buffer.push(chunk);
    } else {
        this._writableState.writing = true;
        this._write(chunk, 'utf8', () => this.next());
    }
}
Writable.prototype.next = function () {
    this._writableState.writing = false;
    if (this._writableState.buffer.length > 0) {
        this._write(this._writableState.buffer.shift(), 'utf8', () => this.next());
    }
}
Writable.prototype.end = function () {
    this._writableState.ended = true;
}
module.exports = Writable;
```

3.2 highWaterMark

3.2.1 3.highWaterMark.js

3.highWaterMark.js

```

const Writable = require('./Writable');
class WritableStream extends Writable {
    _write = (data, encoding, next) => {
        console.log(data.toString());
        setTimeout(next, 1000);
    }
}
const writableStream = new WritableStream({
    highWaterMark: 1
});
writableStream.on('finish', () => {
    console.log('finish');
});
let canWrite = writableStream.write('1');
console.log('canWrite:1', canWrite);
canWrite = writableStream.write('2');
console.log('canWrite:2', canWrite);
canWrite = writableStream.write('3');
console.log('canWrite:3', canWrite);
writableStream.once('drain', () => {
    console.log('drain');
    let canWrite = writableStream.write('4');
    console.log('canWrite:4', canWrite);
    canWrite = writableStream.write('5');
    console.log('canWrite:5', canWrite);
    canWrite = writableStream.write('6');
    console.log('canWrite:6', canWrite);
});

```

3.2.2 Writable.js

Writable.js

```
const Stream = require('./Stream');
var { inherits } = require('util');
function Writable(options) {
    Stream.call(this, options);
    this._writableState = {
        ended: false,
        writing: false,
        buffer: [],
+       bufferSize: 0
    };
    if (options.write) this._write = options.write;
}
inherits(Writable, Stream);
Writable.prototype.write = function (chunk) {
    if (this._writableState.ended) {
        return;
    }
    chunk = Buffer.isBuffer(chunk) ? chunk : Buffer.from(chunk, 'utf8');
    this._writableState.bufferSize += chunk.length;
+   let canWrite = this.options.highWaterMark > this._writableState.bufferSize;
+   if (this._writableState.writing) {
+       this._writableState.buffer.push(chunk);
+   } else {
+       this._writableState.writing = true;
+       this._write(chunk, 'utf8', () => this.next());
+   }
+   return canWrite;
}
Writable.prototype.next = function () {
    this._writableState.writing = false;
+   if (this._writableState.buffer.length > 0) {
+       let chunk = this._writableState.buffer.shift();
+       this._write(chunk, 'utf8', () => {
+           this._writableState.bufferSize -= chunk.length;
+           this.next();
+       })
+   } else {
+       this.emit('drain');
+   }
}
Writable.prototype.end = function () {
    this._writableState.ended = true;
}
module.exports = Writable;
```

4.pipe(管道)

4.1 3.pipe.js

3.pipe.js

```
const readableStream = require('./readableStream');
const writableStream = require('./writableStream');
readableStream.pipe(writableStream);
```

4.2 Readable.js

Readable.js

```
const Stream = require('./Stream');
var { inherits } = require('util');
function Readable(options) {
    Stream.call(this, options);
    this._readableState = { ended: false, buffer: [], flowing: false };
    if (options.read) this._read = options.read;
}
inherits(Readable, Stream);
Readable.prototype.on = function (event, fn) {
    Stream.prototype.on.call(this, event, fn);
    if (event
        this.resume();
    }
}
Readable.prototype.resume = function () {
    this._readableState.flowing = true;
    while (this.read());
}
Readable.prototype.pause = function () {
    this._readableState.flowing = false;
}
Readable.prototype.read = function () {
    if (!this._readableState.ended && this._readableState.flowing) {
        this._read();
    }
    let data = this._readableState.buffer.shift();
    if (data) {
        this.emit('data', data);
    }
    return data;
}
Readable.prototype.push = function (chunk) {
    if (chunk
        this._readableState.ended = true;
    } else {
        this._readableState.buffer.push(chunk);
    }
}
+Readable.prototype.pipe = function (dest) {
+    this.on('data', (chunk) => {
+        dest.write(chunk);
+    })
+    this.on('end', () => {
+        dest.end();
+    });
+}
module.exports = Readable;
```

5.Duplex()双工流)

5.1 4.duplexStream.js

4.duplexStream.js

```
const duplexStream = require('./duplexStream');
duplexStream.pipe(duplexStream);
```

5.2 duplexStream.js

duplexStream.js

```
const Duplex = require('./Duplex');
const readableIterator = (function (count) {
    return {
        next() {
            count++;
            if (count 5) {
                return { done: false, value: count + '' };
            } else {
                return { done: true, value: null }
            }
        }
    }
})(0)
const duplexStream = new Duplex({
    read() {
        let { done, value } = readableIterator.next();
        if (done) {
            this.push(null);
        } else {
            this.push(value);
        }
    },
    write(data, encoding, next) {
        console.log(data.toString(encoding));
        setTimeout(next, 1000);
    }
});
module.exports = duplexStream;
```

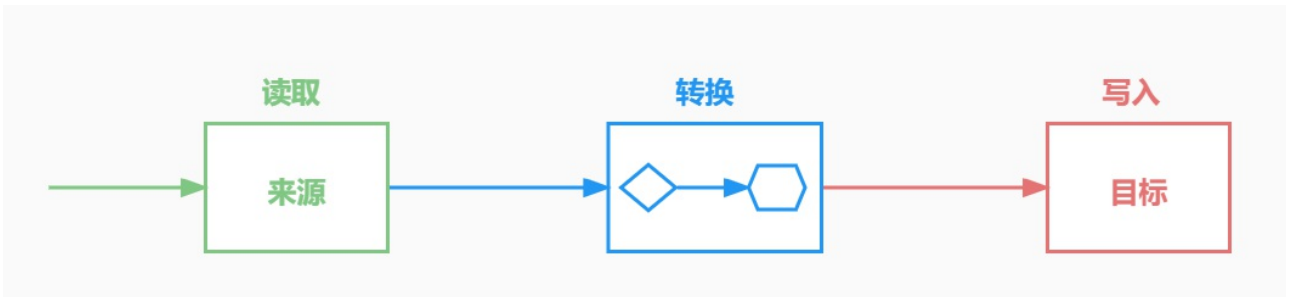
5.3 Duplex.js

Duplex.js

```
const Readable = require('./Readable');
const Writable = require('./Writable');
var { inherits } = require('util');
inherits(Duplex, Readable);
const keys = Object.keys(Writable.prototype);
for (let v = 0; v < keys.length; v++) {
    const method = keys[v];
    if (!Duplex.prototype[method]) {
        Duplex.prototype[method] = Writable.prototype[method];
    }
}
function Duplex(options) {
    Readable.call(this, options);
    Writable.call(this, options);
}

module.exports = Duplex;
```

6.Transform(转换流)



6.1 5.transformStream.js

5.transformStream.js

```
const readableStream = require('./readableStream');
const transformStream = require('./transformStream');
const writableStream = require('./writableStream');
readableStream.pipe(transformStream).pipe(writableStream);
```

6.2 transformStream.js

transformStream.js

```
const Transform = require('./Transform');
const transformStream = new Transform({
    transform(buffer, encoding, next) {
        let transformed = buffer.toString(encoding) + '{{content}}#x27;;
        next(null, transformed);
    }
});
module.exports = transformStream;
```

6.3 Transform.js

Transform.js

```
const Duplex = require('./Duplex');
var { inherits } = require('util');
inherits(Transform, Duplex);
function Transform(options) {
    Duplex.call(this, options);
    if (options.transform) this._transform = options.transform;
}
Transform.prototype._write = function (chunk, encoding, next) {
    this._transform(chunk, encoding, (err, data) => {
        if (data) {
            this.push(data);
        }
        next(err);
    });
}
Transform.prototype._read = function () {

}
module.exports = Transform;
```

7.objectMode(对象模式)

- 默认情况下，流处理的数据是 Buffer/String类型的值
- 有一个 objectMode标志，我们可以设置它让流可以接受任何 JavaScript对象

7.1 6.objectMode.js

6.objectMode.js

```
const { Readable, Writable } = require('stream');
const readableIterator = (function (count) {
    return {
        next() {
            count++;
            if (count 5) {
                return { done: false, value: { id: count + '' } };
            } else {
                return { done: true, value: null }
            }
        }
    }
})(0)
const readableStream = new Readable({
    objectMode: true,
    read() {
        let { done, value } = readableIterator.next();
        if (done) {
            this.push(null);
        } else {
            this.push(value);
        }
    }
});
const writableStream = new Writable({
    objectMode: true,
    write(data, encoding, next) {
        console.log(data);
        setTimeout(next, 1000);
    }
});
readableStream.pipe(writableStream);
```

8.through2

- through2是一个简单的流处理模块，它提供了一个简单的接口，可以让我们更加方便地处理流

8.1 7.through2.js

```
const fs = require('fs');
const through2 = require('./through2');
const readableStream = require('./readableStream');
const writableStream = require('./writableStream');
const transformStream = through2(function (chunk, encoding, next) {
    let transformed = chunk.toString(encoding) + '{{content}}#x27;;
    next(null, transformed);
});
readableStream.pipe(transformStream).pipe(writableStream);
```

8.2 through2.js

through2.js

```
const fs = require('fs');
const through2 = require('./through2');
const readableStream = require('./readableStream');
const writableStream = require('./writableStream');
const transformStream = through2(function (chunk, encoding, next) {
    let transformed = chunk.toString(encoding) + '{{content}}#x27;;
    next(null, transformed);
});
readableStream.pipe(transformStream).pipe(writableStream);
```

8.3 through2.obj

8.through2.js

```
const fs = require('fs');
const through2 = require('through2');
const fileStream = fs.createReadStream('data.txt', { highWaterMark: 10 });
const all = [];
fileStream.pipe(
    through2.obj(function (chunk, encoding, next) {
        this.push(JSON.parse(chunk))
        next();
    })).on('data', (data) => {
        all.push(data)
    }).on('end', () => {
        console.log(all);
    })
```

8.4 through2.js

through2.js

```
const Transform = require('./Transform');
const { Transform } = require('stream');
function through2(transform) {
    return new Transform({
        transform
    });
}
through2.obj = function (transform) {
    return new Transform({
        objectMode: true,
        transform
    });
}
module.exports = through2;
```

8.5 data.txt

data.txt

```
{"id":1}
{"id":2}
{"id":3}
```