**cluster多核处理模块**

Node.js是单线程运行的，不管你的机器有多少个内核，只能用到其中的一个，为了能利用多核计算资源，需要使用多进程来处理应用。cluster模块让我们可以很容易地创建一个负载均衡的集群，自动分配CPU多核资源。  
使用如下所示：

const cluster = require('cluster');

const http = require('http');

const cpuNums = require('os').cpus().length;

if(cluster.isMaster){

for(let i=0;i<cpuNums;i++){

cluster.fork();

}

cluster.on('exit',(worker)=>{

console.log(`worker${worker.id} exit.`)

});

cluster.on('fork',(worker)=>{

console.log(`fork：worker${worker.id}`)

});

cluster.on('listening',(worker,addr)=>{

console.log(`worker${worker.id} listening on ${addr.address}:${addr.port}`)

});

cluster.on('online',(worker)=>{

console.log(`worker${worker.id} is online now`)

});

}else{

http.createServer((req,res)=>{

console.log(cluster.worker.id);

res.writeHead(200);

res.end('hello world');

}).listen(3000,'127.0.0.1');

}

执行结果：

fork：worker1  
fork：worker2  
fork：worker3  
fork：worker4  
worker1 is online now  
worker2 is online now  
worker3 is online now  
worker1 listening on 127.0.0.1:3000  
worker4 is online now  
worker2 listening on 127.0.0.1:3000  
worker3 listening on 127.0.0.1:3000  
worker4 listening on 127.0.0.1:3000

**cluster工作原理**  
如上代码所示，master是控制进程，worker是执行进程，每个worker都是使用child\_process.fork()函数创建的，因此worker与master之间通过IPC进行通信。  
当worker调用用server.listen()方法时会向master进程发送一个消息，让它创建一个服务器socket，做好监听并分享给该worker。如果master已经有监听好的socket，就跳过创建和监听的过程，直接分享。换句话说，所有的worker监听的都是同一个socket，当有新连接进来的时候，由负载均衡算法选出一个worker进行处理。  
**cluster对象的属性和方法**  
cluster.isMaster：标志是否master进程，为true则是  
cluster.isWorker：标志是否worker进程，为true则是  
cluster.worker：获得当前的worker对象，在master进程中使用无效  
cluster.workers： 获得集群中所有存活的worker对象，子啊worker进程使用无效  
cluster.fork()： 创建工作进程worker  
cluster.disconnect([callback])： 断开所有worker进程通信  
\***cluster对象的事件**  
Event: 'fork'： 监听创建worker进程事件  
Event: 'online'： 监听worker创建成功事件  
Event: 'listening'： 监听worker进程进入监听事件  
Event: 'disconnect'： 监听worker断开事件  
Event: 'exit'： 监听worker退出事件  
Event: 'message'：监听worker进程发送消息事件  
使用如下所示：

const cluster = require('cluster');

const http = require('http');

const cpuNums = require('os').cpus().length;

/\*process.env.NODE\_DEBUG='net';\*/

if(cluster.isMaster){

for(let i=0;i<cpuNums;i++){

cluster.fork();

}

cluster.on('exit',(worker)=>{

console.log(`worker${worker.id} exit.`)

});

cluster.on('fork',(worker)=>{

console.log(`fork：worker${worker.id}`)

});

cluster.on('disconnect',(worker)=>{

console.log(`worker${worker.id} is disconnected.`)

});

cluster.on('listening',(worker,addr)=>{

console.log(`worker${worker.id} listening on ${addr.address}:${addr.port}`)

});

cluster.on('online',(worker)=>{

console.log(`worker${worker.id} is online now`)

});

cluster.on('message',(worker,msg)=>{

console.log(`got the worker${worker.id}'s msg：${msg}`);

});

Object.keys(cluster.workers).forEach((id)=>{

cluster.workers[id].send(`hello worker${id}`);

});

}else{

process.on('message',(msg)=>{

console.log('worker'+cluster.worker.id+' got the master msg：'+msg);

});

process.send('hello master, I am worker'+cluster.worker.id);

http.createServer((req,res)=>{

res.writeHead(200);

res.end('hello world'+cluster.worker.id);

}).listen(3000,'127.0.0.1');

}

执行结果如下：

fork：worker1  
fork：worker2  
fork：worker3  
fork：worker4  
worker1 is online now  
worker2 is online now  
got the worker1's msg：hello master, I am worker1  
worker1 got the master msg：hello worker1  
worker1 listening on 127.0.0.1:3000  
worker4 is online now  
got the worker2's msg：hello master, I am worker2  
worker2 got the master msg：hello worker2  
worker3 is online now  
worker2 listening on 127.0.0.1:3000  
got the worker4's msg：hello master, I am worker4  
worker4 got the master msg：hello worker4  
worker4 listening on 127.0.0.1:3000  
got the worker3's msg：hello master, I am worker3  
worker3 got the master msg：hello worker3  
worker3 listening on 127.0.0.1:3000

在win7环境下，cluster负载均衡情况，如下所示：  
服务端代码：

const cluster = require('cluster');

const http = require('http');

const cpuNums = require('os').cpus().length;

if(cluster.isMaster){

var i = 0;

const widArr = [];

for(let i=0;i<cpuNums;i++){

cluster.fork();

}

cluster.on('message',(worker,msg)=>{

if(msg === 'ex'){

i++;

widArr.push(worker.id);

(i>=80)&&(process.exit(0));

}

});

process.on('exit', (code) => {

console.log(analyzeArr(widArr));

});

//统计每个worker被调用的次数

function analyzeArr(arr) {

let obj = {};

arr.forEach((id, idx, arr) => {

obj['work' + id] = obj['work' + id] !== void 0 ? obj['work' + id] + 1 : 1;

});

return obj;

}

}else{

http.createServer((req,res)=>{

console.log(`worker${cluster.worker.id}`);

process.send('ex');

res.writeHead(200);

res.end('hello world'+cluster.worker.id);

}).listen(3000,'127.0.0.1');

}

使用Apache的AB命令进行测试，并发40，总共80：C:\Users\learn>ab -c 40 -n 80 http://127.0.0.1:3000/。  
测试结果：

{ work4: 19, work3: 20, work1: 19, work2: 22 }