link: null
title: 珠峰架构师成长计划
description: 在Node.js中，使用console对象代表控制台(在操作系统中表现为一个操作系统指定的字符界面，比如 Window中的命令提示窗口)。
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=31 sentences=140, words=795

## 1. 控制台

在Node.js中，使用 console对象代表控制台(在操作系统中表现为一个操作系统指定的字符界面，比如 Window中的命令提示窗口)。

- console.log
- console.info
- console.error 重定向到文件
- console.warn
- console.dir
- console.time
- console.timeEnd
- console.trace
- console.assert

## 2. 全局作用域

- 全局作用域(global)可以定义一些不需要通过任何模块的加载即可使用的变量、函数或类
- 定义全局变量时变量会成为global的属性。
- 永远不要不使用var关键字定义变量，以免污染全局作用域
- setTimeout clearTimeout
- setInterval clearInterval
- unref和ref

```
let test = function(){
  console.log('callback');
}
let timer = setInterval(test,1000);
timer.unref();
setTimeout(function(){
  timer.ref();
},3000)
```

## 3. 函数

- require
- 模块加载过程
- require.resolve
- 模板缓存(require.cache)
- require.main
- 模块导出

```
module.exports, require, module, filename, dirname
```

## 4. process

在node.js里，process 对象代表node.js应用程序，可以获取应用程序的用户，运行环境等各种信息

```
process.argv.forEach(function(item){
  console.log(item);
});
process.on('exit',function(){
  console.log('clear');
});

process.on('uncaughtException',function(err){
  console.log(err);
})

console.log(process.memoryUsage());
console.log(process.cwd());
console.log(__dirname);
process.chdir('..');
console.log(process.cwd());
console.log(__dirname);

function err(){
 throw new Error('&#x62A5;&#x9519;&#x4E86;');
}
err();
```

- process.nextTick()方法将 callback 添加到"next tick 队列"。 一旦当前事件轮询队列的任务全部完成，在next tick队列中的所有callbacks会被依次调用。
- setImmediate预定立即执行的 callback，它是在 I/O 事件的回调之后被触发

```
setImmediate(function(){
  console.log('4');
});
setImmediate(function(){
  console.log('5');
});
process.nextTick(function(){
  console.log('1');
  process.nextTick(function(){
    console.log('2');
    process.nextTick(function(){
      console.log('3');
    });
  });
});

console.log('next');
```

## 5. EventEmitter

在Node.js的用于实现各种事件处理的event模块中，定义了EventEmitter类，所以可能触发事件的对象都是一个继承自EventEmitter类的子类实例对象。

方法名和参数 描述 addListener(event,listener) 对指定事件绑定事件处理函数 on(event,listener) 对指定事件绑定事件处理函数 once(event,listener) 对指定事件指定只执行一次的事件处理函数 removeListener(event,listener) 对指定事件解除事件处理函数 removeAllListeners([event]) 对指定事件解除所有的事件处理函数 setMaxListeners(n) 指定事件处理函数的最大数量.n为整数值，代表最大的可指定事件处理函数的数量 listeners(event) 获取指定事件的所有事件处理函数 emit(event,[arg1],[arg2],[...]) 手工触发指定事件

```
let EventEmitter = require('./events');
let util = require('util');
util.inherits(Bell,EventEmitter);
function Bell(){
  EventEmitter.call(this);
}
let bell = new Bell();
bell.on('newListener',function(type,listener){
  console.log(`&#x5BF9; ${type}  &#x4E8B;&#x4EF6;&#x589E;&#x52A0;${listener}`);
});
bell.on('removeListener',function(type,listener){
  console.log(`&#x5BF9;${type} &#x4E8B;&#x4EF6;&#x5220;&#x9664;${listener}`);
});
function teacherIn(thing){
  console.log(`&#x8001;&#x5E08;&#x5E26;${thing}&#x8FDB;&#x6559;&#x5BA4;`);
}
function studentIn(thing){
  console.log(`&#x5B66;&#x751F;&#x5E26;${thing}&#x8FDB;&#x6559;&#x5BA4;`);
}
function masterIn(thing){
  console.log(`&#x6821;&#x957F;&#x5E26;${thing}&#x8FDB;&#x6559;&#x5BA4;`);
}
bell.on('&#x54CD;',teacherIn);
bell.on('&#x54CD;',studentIn);
bell.once('&#x54CD;',masterIn);
bell.emit('&#x54CD;','&#x4E66;');
console.log('==============');
bell.emit('&#x54CD;','&#x4E66;');
console.log('==============');
bell.removeAllListeners('&#x54CD;');
console.log('==============');
bell.emit('&#x54CD;','&#x4E66;');
```

```
function EventEmitter(){
  this.events =
{};//&#x4F1A;&#x628A;&#x6240;&#x6709;&#x7684;&#x4E8B;&#x4EF6;&#x76D1;&#x542C;&#x51FD;&#x6570;&#x653E;&#x5728;&#x8FD9;&#x4E2A;&#x5BF9;&#x8C61;&#x91CC;&#x4FDD;&#x
5B58;

//&#x6307;&#x5B9A;&#x7ED9;&#x4E00;&#x4E2A;&#x4E8B;&#x4EF6;&#x7C7B;&#x578B;&#x589E;&#x52A0;&#x7684;&#x76D1;&#x542C;&#x51FD;&#x6570;&#x6570;&#x91CF;&#x6700;&#x591
;&#x6709;&#x591A;&#x5C11;&#x4E2A;
  this._maxListeners = 10;
}
EventEmitter.prototype.setMaxListeners = function(maxListeners){
  this._maxListeners = maxListeners;
}
EventEmitter.prototype.listeners = function(event){
  return this.events[event];
}
//&#x7ED9;&#x6307;&#x5B9A;&#x7684;&#x4E8B;&#x4EF6;&#x7ED1;&#x5B9A;&#x4E8B;&#x4EF6;&#x5904;&#x7406;&#x51FD;&#x6570;&#xFF0C;1&#x53C2;&#x6570;&#x662F;&#x4E8B;&#x4E
F6;&#x7C7B;&#x578B; 2&#x53C2;&#x6570;&#x662F;&#x4E8B;&#x4EF6;&#x76D1;&#x542C;&#x51FD;&#x6570;
EventEmitter.prototype.on = EventEmitter.prototype.addListener = function(type,listener){
  if(this.events[type]){
    this.events[type].push(listener);
    if(this._maxListeners!=0&&this.events[type].length>this._maxListeners){
      console.error(`MaxListenersExceededWarning: Possible EventEmitter memory leak detected. ${this.events[type].length} ${type} listeners added. Use
emitter.setMaxListeners() to increase limit`);
    }
  }else{

//&#x5982;&#x679C;&#x4EE5;&#x524D;&#x6CA1;&#x6709;&#x6DFB;&#x52A0;&#x5230;&#x6B64;&#x4E8B;&#x4EF6;&#x7684;&#x76D1;&#x542C;&#x51FD;&#x6570;&#xFF0C;&#x5219;&#x8D4
;&#x4E00;&#x4E2A;&#x6570;&#x7EC4;
    this.events[type] = [listener];
  }
}
EventEmitter.prototype.once = function(type,listener){
  //&#x7528;&#x5B8C;&#x5373;&#x711A;
 let  wrapper = (...rest)=>{
   listener.apply(this);//&#x5148;&#x8BA9;&#x539F;&#x59CB;&#x7684;&#x76D1;&#x542C;&#x51FD;&#x6570;&#x6267;&#x884C;
   this.removeListener(type,wrapper);
 }
 this.on(type,wrapper);
}
EventEmitter.prototype.removeListener = function(type,listener){
  if(this.events[type]){
    this.events[type] = this.events[type].filter(l=>l!=listener)
  }
}
//&#x79FB;&#x9664;&#x67D0;&#x4E2A;&#x4E8B;&#x4EF6;&#x7684;&#x6240;&#x6709;&#x76D1;&#x542C;&#x51FD;&#x6570;
EventEmitter.prototype.removeAllListeners = function(type){
  delete this.events[type];
}
EventEmitter.prototype.emit = function(type,...rest){
  this.events[type]&&this.events[type].forEach(listener=>listener.apply(this,rest));
}
module.exports = EventEmitter;
```

### util

```
var util = require('util');
//util.inherit();
console.log(util.inspect({name:'zfpx'}));
console.log(util.isArray([]));
console.log(util.isRegExp(/\d/));
console.log(util.isDate(new Date()));
console.log(util.isError(new Error));
```

## 6. node断点调试

V8 提供了一个强大的调试器，可以通过 TCP 协议从外部访问。Nodejs提供了一个内建调试器来帮助开发者调试应用程序。想要开启调试器我们需要在代码中加入debugger标签，当Nodejs执行到debugger标签时会自动暂停（debugger标签相当于在代码中开启一个断点）。

```
var a = 'a';
var b = 'b';

debugger;

var all = a + ' ' + b;
console.log(all);
```

命令 用途 c 继续执行到下一个断点处 next,n 单步执行 step,s 单步进入函数 out,o 退出当前函数 setBreakpoint(10),sb(10) 在第10行设置断点 repl 打开求值环境，ctrl_c退回debug模式 watch(exp) 把表达式添加监视列表 watchers 显示所有表达式的值