#node.js

回调函数

> 一般作为函数的最后一个参数：

```
>function foo1(value,callback) { }
>function foo2(value,callback1,callback2) { }
```

阻塞代码实例

```
>var fs = require("fs");
>var data = fs.readFileSync('input.txt');
>console.log(data.toString());
>console.log("程序执行结束!");
```

非阻塞代码实例

```
>var fs = require("fs");
>fs.readFile('input.txt', function (err, data) {
>   if (err) return console.error(err);
>   console.log(data.toString());
>});
>console.log("程序执行结束!");
```

事件循环

事件驱动程序

> webSever一直接受请求不等待任何读写操作
> 生成一个主循环来监听事件，当检测到事件时触发回调函数

```
//引入events模块
var events = require('events');
//创建 eventEmitter 对象
var eventEmitter = new events.EventEmitter();
//绑定事件及事件的处理程序
eventEmitter.on('eventName',eventHandler);
//触发事件
eventEmitter.emit('eventName');
```

实例

```
//引入events模块
var events = require('events');
//创建eventEmitter 对象
var eventEmitter = new events.EventEmitter();
//创建事件处理程序
var  connectHandler = function connected() {
    console.log(`连接成功。`);
  //触发 data_received事件
```

```
  eventEmitter.emit('data_received');
}
//绑定connection 事件处理程序
eventEmitter.on('connection',connecHandler);
//使用匿名函数绑定data_received事件
eventEmitter.on('data.received',function(){
    console.log('数据接收成功');
});
//触发connection事件
eventEmitter.emit('connection');
console.log("程序执行完毕")
```

EventEmitter

> EventEmitter 对象如果在实例时发生错误，会触发error事件。当添加新的监听器时，newListener事件会被触发，当监听器被移除时，removeListener事件被触发。

```
var EventEmitter = require('events').EventEmitter;
var event = new EventEmitter();
event.on('some_event', function() {
    console.log('some_event 事件触发');
});
setTimeout(function() {
    event.emit('some_event');
}, 1000);
```

> EventEmitter的每个事件由一个事件名和若干参数组成，事件名是一个字符串，通常表达一定的语义。对于每个事件，EventEmitter支持若干个事件监听器。当事件发生时，注册到这个事件的监听器被依次调用，事件参数作为回调函数参数传递。

```
var events = require('events');
var emitter = new events.EventEmitter();
emitter.on('someEvent', function(arg1, arg2) {
    console.log('listener1', arg1, arg2);
});
emitter.on('someEvent', function(arg1, arg2) {
    console.log('listener2', arg1, arg2);
});
emitter.emit('someEvent', 'arg1 参数', 'arg2 参数');
```

方法

序号	方法	描述
1	addListener(event,listener)	为指定事件添加一个事件监听器到监听数组的尾部
2	on(event,listener)	未指定事件注册一个监听器，接受一个字符串event和一个回调函数
3	once(event,listener)	为指定事件注册一个单次监听器，即最多只会触发一次，触发后立即解除该监听器

序号	方法	描述
4	removeListener(event,listener)	移除指定事件的监听器，监听器必须是该事件已经注册过的监听器。
5	removeAllListeners([event])	移除所有事件的所有监听器，如果指定事件，则移除指定事件的所有监听器
6	setMaxListeners(n)	EventEmitter默认最多10个监听器（超出会警告），此函数用于设置最大值
7	listeners(event)	返回指定事件的监听器数组
8	emit(event,[arg1],[arg2]...)	按参数的顺序执行每个监听器，如果事件有注册监听返回true,否则返回false

类方法

序号	方法	描述
1	listenerCount(emittier,event)	返回指定事件的监听器数量

实例

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// 监听器 #1
var listener1 = function listener1() {
   console.log('监听器 listener1 执行。');
}

// 监听器 #2
var listener2 = function listener2() {
  console.log('监听器 listener2 执行。');
}

// 绑定 connection 事件，处理函数为 listener1
eventEmitter.addListener('connection', listener1);

// 绑定 connection 事件，处理函数为 listener2
eventEmitter.on('connection', listener2);

//注意类方法的调用方式
var eventListeners = require('events').EventEmitter.listenerCount(eventEmitter,'connection');
console.log(eventListeners + " 个监听器监听连接事件。");

// 处理 connection 事件
eventEmitter.emit('connection');

// 移除监绑定的 listener1 函数
eventEmitter.removeListener('connection', listener1);
```

```
29  console.log("listener1 不再受监听。");
30
31  // 触发连接事件
32  eventEmitter.emit('connection');
33
34  eventListeners = require('events').EventEmitter.listenerCount(eventEmitter,'connection');
35  console.log(eventListeners + " 个监听器监听连接事件。");
36
37  console.log("程序执行完毕。");
```

error事件

> EvenEmitter定义了一个特殊的事件error，它包含了错误的语义，我们在遇到异常的时候通常会触发error事件。当error被触发时，EventEmiter规定如果没有响应的监听器，Node.js会把它当作异常，退出程序并输出错误信息。我们一般要为会触发的error事件的对象设置监听器，避免崩溃。

```
1  var events = require('events');
2  var emitter = new events.EventEmitter();
3  emitter.emit('error')
```