

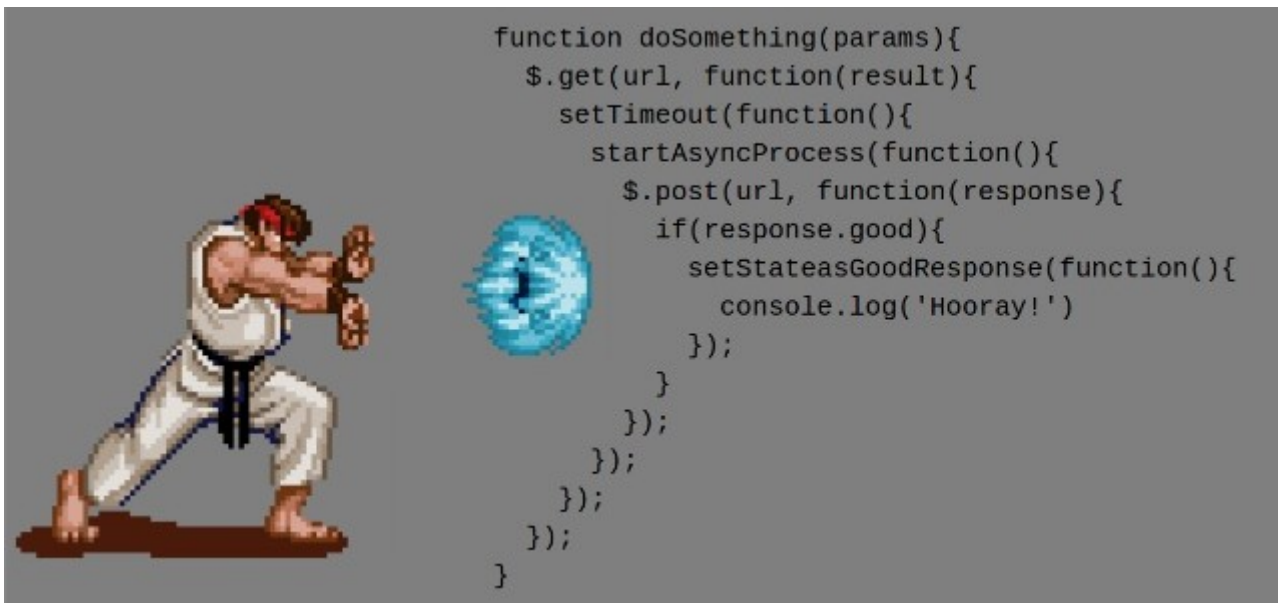
/

“ /

- [\[Basic\]](#) [Promise](#)
- [\[Doc\]](#) [Events \(\)](#)
- [\[Doc\]](#) [Timers \(\)](#)
- [\[Point\]](#) /
- [\[Point\]](#) /

? .

Promise



[Callback Hell](#) [Q](#), [async](#), [EventProxy](#) [Promise](#) , [ES6](#) [JavaScript](#) .

[Promise](#) .

“ [Promise](#) .then .catch ?

We have a problem with promises

, , Promise :

```
let doSth = new Promise((resolve, reject) => {
  console.log('hello');
  resolve();
});

doSth.then(() => {
  console.log('over');
});
```

:

```
hello
over
```

, Promise , then ?

```
setTimeout(10s).then(hello, 10s, ?
```

```
let doSth = new Promise((resolve, reject) => {
  console.log('hello');
  resolve();
});

setTimeout(() => {
  doSth.then(() => {
    console.log('over');
  })
}, 10000);
```

): (

```
setTimeout(function() {
  console.log(1)
}, 0);

new Promise(function executor(resolve) {
  console.log(2);
  for( var i=0 ; i<10000 ; i++ ) {
```

```

    i == 9999 && resolve();
  }
  console.log(3);
}).then(function() {
  console.log(4);
});
console.log(5);

```

, . Promise , Promise , Promise ,

Events

`Events` `Node.js` `core` , `node` `Stream` `Events` `fs` , `net` , `http` `Stream` , `Events`

`EventEmitter` `node` `event` , `emitter` , `(emit)` `cb` `listener` . `D`

“ `Eventemitter` `emit` ?

`Node.js` `Eventemitter` `emit` . :

“ The `EventListener` calls all listeners synchronously in the order in which they were registered. This is important to ensure the proper sequencing of events and to avoid race conditions or logic errors.

`hi 1` `hi 2` ?

```

const EventEmitter = require('events');

let emitter = new EventEmitter();

emitter.on('myEvent', () => {
  console.log('hi 1');
});

emitter.on('myEvent', () => {
  console.log('hi 2');
});

```

```
emitter.emit('myEvent');
```

?

```
const EventEmitter = require('events');

let emitter = new EventEmitter();

emitter.on('myEvent', () => {
  console.log('hi');
  emitter.emit('myEvent');
});

emitter.emit('myEvent');
```

?

```
const EventEmitter = require('events');

let emitter = new EventEmitter();

emitter.on('myEvent', function sth () {
  emitter.on('myEvent', sth);
  console.log('hi');
});

emitter.emit('myEvent');
```

emitter, TCP, .emit .once
emitter, emitter listener

/

“ ? ?

, node .

- console.log

- IO

, IO , setTimeout .

“ , koa , A, A . , ,

Node.js js . , , pop . ① sleep ,

“ sleep ? ①

```
function sleep(ms) {
  var start = Date.now(), expire = start + ms;
  while (Date.now() < expire) ;
  return;
}
```

, libuv (C/C++ libev libevent) .

, , , , js , .

“ reduce? (: reduce)

reduce , n n+1 , n+2 . , .

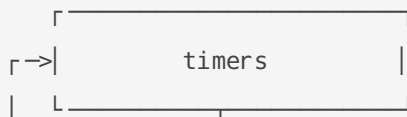
Timers

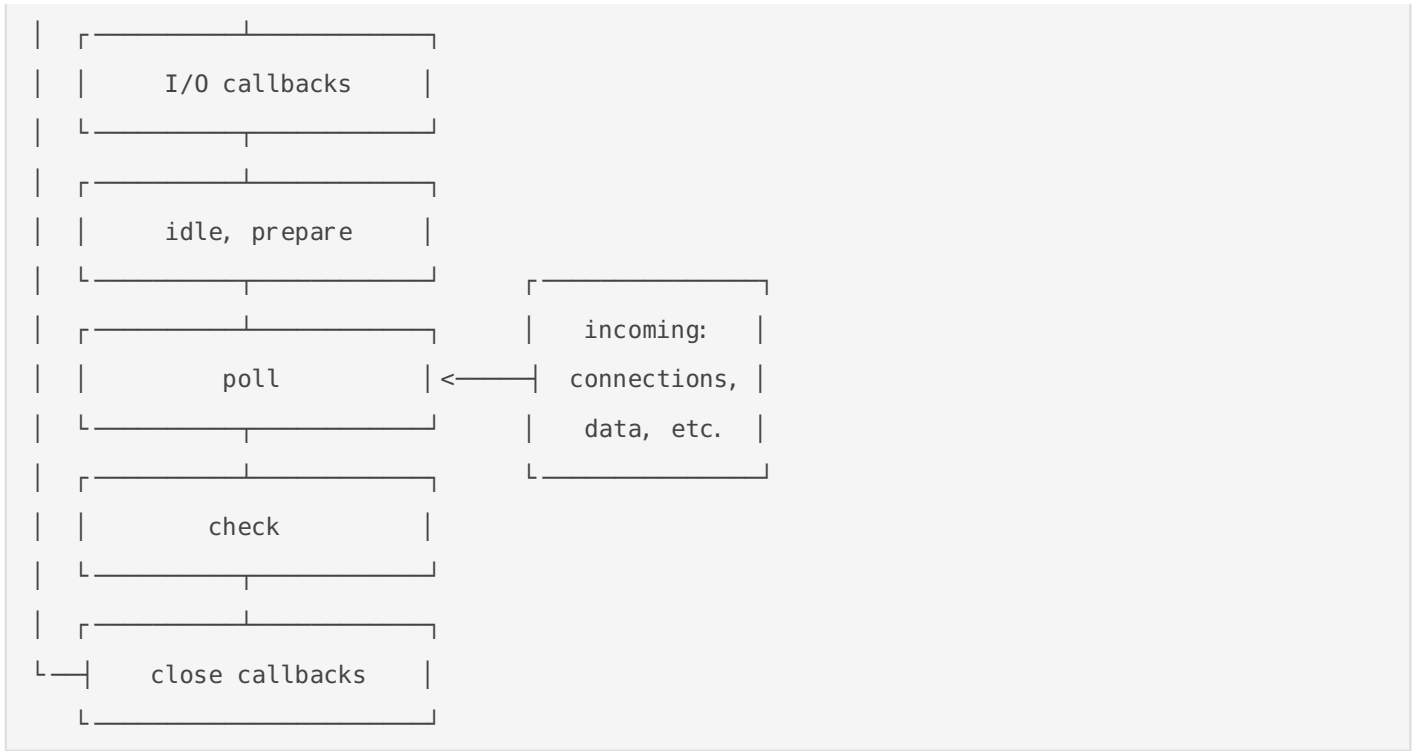
Node.js , .

IO libuv . , readFileSync, execSync , node ,

, setTimeout, nextTick, setTimeout setImmediate

Event loop





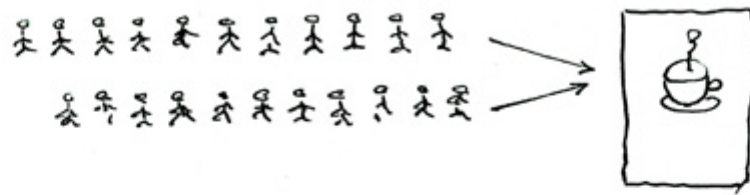
nextTick, Timers, and process.nextTick() The Node.js Event Loop, Timers, and process.nextTick(), queues and schedules

/

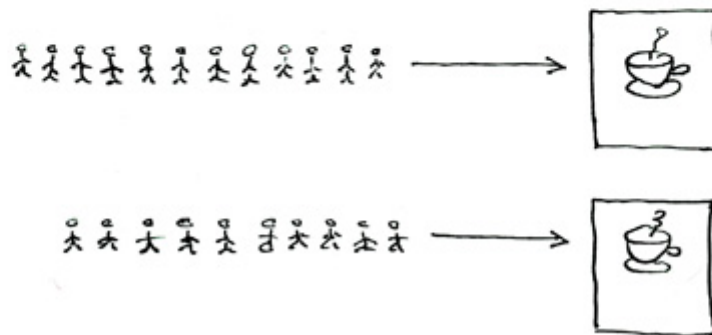
(Parallel) (Concurrent) .

Erlang Joe Armstrong (Concurrent and Parallel)

Concurrent = Two Queues One Coffee Machine



Parallel = Two Queues Two Coffee Machines



© Joe Armstrong 2013

(Concurrent) = 2 1 .

(Parallel) = 2 2 .

Node.js 2 1 Task / , .

node , cluster .

Revision #2

Created 19 July 2021 15:22:34 by

Updated 19 July 2021 15:26:57 by