NPM 使用介绍 Node.js 事件循环

Node.js 回调函数

Node.js 异步编程的直接体现就是回调。

异步编程依托于回调来实现，但不能说使用了回调后程序就异步化了。

回调函数在完成任务后就会被调用，Node 使用了大量的回调函数，Node 所有 API 都支持回调函数。

例如，我们可以一边读取文件，一边执行其他命令，在文件读取完成后，我们将文件内容作为回调函数的参数返回。这样在执行代码时就没有阻塞或等待文件 I/O 操作。这就大大提高了 Node.js 的性能，可以处理大量的并发请求。

回调函数一般作为函数的最后一个参数出现：

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

阻塞代码实例

创建一个文件 input.txt ，内容如下：

```
菜鸟教程官网地址：www.runoob.com
```

创建 main.js 文件, 代码如下：

```
var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("程序执行结束!");
```

以上代码执行结果如下：

```
$ node main.js
菜鸟教程官网地址：www.runoob.com

程序执行结束!
```

非阻塞代码实例

创建一个文件 input.txt ，内容如下：

```
菜鸟教程官网地址：www.runoob.com
```

创建 main.js 文件, 代码如下：

```
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!");
```

以上代码执行结果如下：

```
$ node main.js
程序执行结束!
菜鸟教程官网地址：www.runoob.com
```

以上两个实例我们了解了阻塞与非阻塞调用的不同。第一个实例在文件读取完后才执行完程序。 第二个实例我们不需要等待文件读取完，这样就可以在读取文件时同时执行接下来的代码，大大提高了程序的性能。

因此，阻塞是按顺序执行的，而非阻塞是不需要按顺序的，所以如果需要处理回调函数的参数，我们就需要写在回调函数内。

← NPM 使用介绍 Node.js 事件循环 →