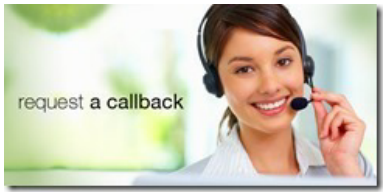
Node.js ABC's - C is for Callbacks

Joe Pruitt, 2014-24-11

With synchronous programming, when you make a call to function, your code is blocked until the method you are calling has processed it's work and returned it's status to you. A good example is the "C" fopen and fread commands.

```
FILE *pFile = fopen("file.txt", "r");  // Wait...
char buf[101];
if ( NULL != pFile ) {
  fread(buf, 1, 100, pFile); // Wait...
  buf[(sizeof buf)-1] = 0;
  printf("%s", buf); // Wait...
  fclose(pFile); // Wait...
}
```



In this code, the vast amount of time used when this code is ran is waiting for the underlying file system to access and read from the file. It is true for most I/O based applications such as ones that utilize databases or connect to external services, your code will spend a majority of it's time sitting around waiting.

One of the key components of Node.js is the concept of non-blocking I/O and asynchronous programming. If you have seen or used the setTimeout() function in JavaScript, you already have seen how a non-blocking call works. With the setTimeout() function, you pass in a function to call and a time after which the function should be called.

```
setTimeout(function() { console.log("Ring-a-ling...")}, 5000);
console.log("I'm Waiting for the phone to ring...");
```

If you run the preceding code, you'll see:

> *I'm waiting for the phone to ring...*
> *Ring-a-ling...*

The code sets a timeout of 5000ms (or 5 seconds), passing in the function to call when it fires, and then it continues with the execution of the script.

The Callback Function

One of the primary patterns you will see for asynchronous programming is by using a callback function you pass to asynchronous functions. It has at least one parameter which is to pass information about whether the call succeeded or failed. It also frequently has a second parameter for passing back additional details about the call (a file system or network socket handle, contents of a object read requests, or the output from a database call).

The following example illustrates how the readFile function in the file system fs core module works with a callback to determine the size of the file.

```
var file = "somefile.txt";
var fs = require("fs");
rs.readFile(file, function(err, data) {
  if ( err ) { throw err; }
  var len = data.toString().split("\n").length -1;   console.log("File Length: " + len);
});
```

Controlling Asynchronous Calls

One can imagine when you have a complicated script that makes many asynchronous calls, you could get into some trouble in handling the callbacks in any kind of predictable order. One handy third-party tool is the "async" module. The async module comes with a series of helper functions for asynchronous calls and response control such as map, reduce, filter, each, etc.

```
async.map(['f1', 'f2', 'f3', 'f4'], fs.stat, function(err, results) {
  // results contains an array of stats for each input item
});

// call functions at the same time and then call callback when they have all completed.
async.parallel([
  function1() { ... },
  function2() { ... },
  function3() { ... }
], callback);

// call functions one at a time in series one at a time.
async.series([
  function1() { ... },
  function2() { ... },
  function3() { ... }
])
```

Synchronous Versions of Core Library Calls

Several of the core libraries include synchronous versions of their asynchronous counterparts. This is primarily evident in the File System module (http://nodejs.org/api/fs.html). Look for functions ending in "Sync" with the omitted callback parameter. These are useful for command line scripting where the processing delays associated with synchronous calls is not a factor.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com