

Callback Hell

Un problema que se suele presentar en el uso de las Callback es lo que se conoce como Callback Hell.

Se produce cuando se encadenan varias callbacks, de modo que una función no se ejecutará hasta que termine la anterior.

Para poder ejecutar una función hay que esperar a que una función anterior termine y devuelva un resultado, ya que ese resultado se necesita para poder ejecutar la función siguiente.

Esto provoca que el código sea difícil de comprender y también difícil de mantener. El problema se puede ver incluso más agravado si utilizamos funciones anónimas directamente en el paso de parámetros.

Una forma de minimizar es Callback Hell es evitando el uso de funciones anónimas, aunque esto tampoco soluciona el problema por completo.

En el ejemplo 1 vemos un ejemplo de ejecución secuencial de funciones. Como se ejecutan de forma síncrona, las funciones se ejecutarán en el orden en el que se han escrito en el programa y como podemos comprobar por la consola los mensajes salen en el orden correcto.

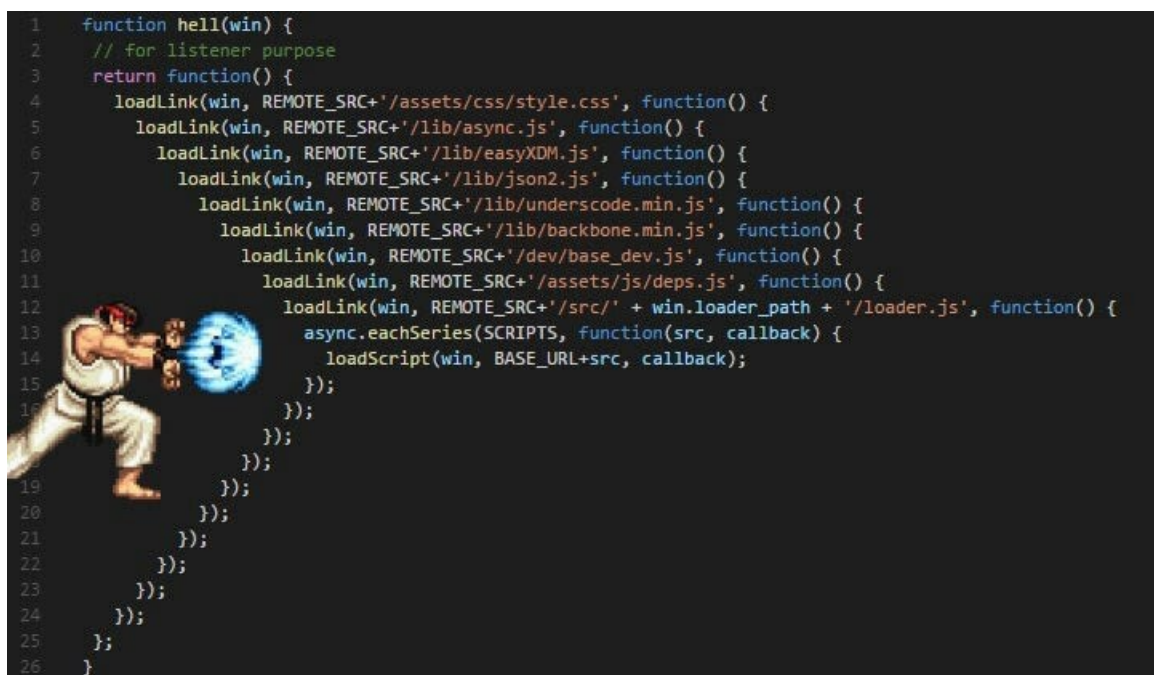
```
Consiguiendo la ternera de Pepe
Haciendo filetes la ternera cruda
Cocinando la ternera fileteada
Sirviendo la ternera cocinada
La ternera está servida en la mesa
> |
```

El problema se plantea cuando tenemos que ejecutar tareas asíncronas. En el ejemplo2 “convertimos” las funciones anteriores en funciones asíncronas, de forma que no sabemos cuánto tiempo se necesitará en la ejecución de cada función y no podemos saber el orden en el que se ejecutarán las funciones. Lo podemos comprobar con los mensajes de la consola.

Cocinando la undefined
Sirviendo la undefined
La undefined en la mesa
Consiguiendo la ternera de Pepe
Haciendo filetes la undefined
>

Con las funciones callback podemos solucionar este problema. Usando funciones callback nos aseguraremos de que las funciones se ejecuten en el orden correcto, aunque no sepamos cuándo se ejecutarán. Puedes ver el ejemplo3.

Aunque el ejemplo3 funciona, la anidación de callbacks provoca lo que se conoce como Callback Hell. Este anidamiento de funciones resulta muy complicado de comprender y de mantener:



```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  });
26 }
```

El problema con una función asíncrona es que no podemos devolver valores de la forma habitual. Cuando se invoca a una función asíncrona, no se sabe cuándo terminará, y si escribimos el código para que devuelva un valor, y pretendemos utilizar el valor devuelto unas cuantas instrucciones más adelante, descubriremos que en realidad no se ha devuelto ningún valor, ya que la función no ha terminado.

Podemos “transformar” cualquier función en “asíncrona” utilizando `setTimeout(miFuncion,1000)`. Al invocar a la función es cuando debemos hacer el `setTimeout`. Es la invocación de la función lo que se hace de forma asíncrona.

La clave de usar callbacks está en que la callback se le pasa como parámetro a la función asíncrona para que así, la función asíncrona pueda ejecutar la callback una vez que haya concluido su tarea.

Una función que hace algo de forma asíncrona debería aceptar un argumento de callback donde ponemos la función por ejecutar después de que se complete.