Asynchronous JavaScript

Olivier Liechti & Simon Oulevay
COMEM Web Services 2016

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Let's talk about callbacks

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Callbacks accumulate quickly

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
Book.count(function(err, totalCount) {
  // handle error

  Book.count(criteria, function(err, filteredCount) {
    // handle error

    var query = Book
      .find(criteria)
      .sort('title')
      .skip(offset)
      .limit(limit);

    if (req.query.embed == 'publisher') {
      query = query.populate('publisher');
    }

    query.exec(function(err, books) {
      // handle error

      res.set('X-Pagination-Page', page);
      res.set('X-Pagination-Page-Size', pageSize);
      res.set('X-Pagination-Total', totalCount);
      res.set('X-Pagination-Filtered-Total', filteredCount);

      res.send(books);
    });
  });
});
```

Entering the **first** asynchronous callback function.

Entering the **second** asynchronous callback function.

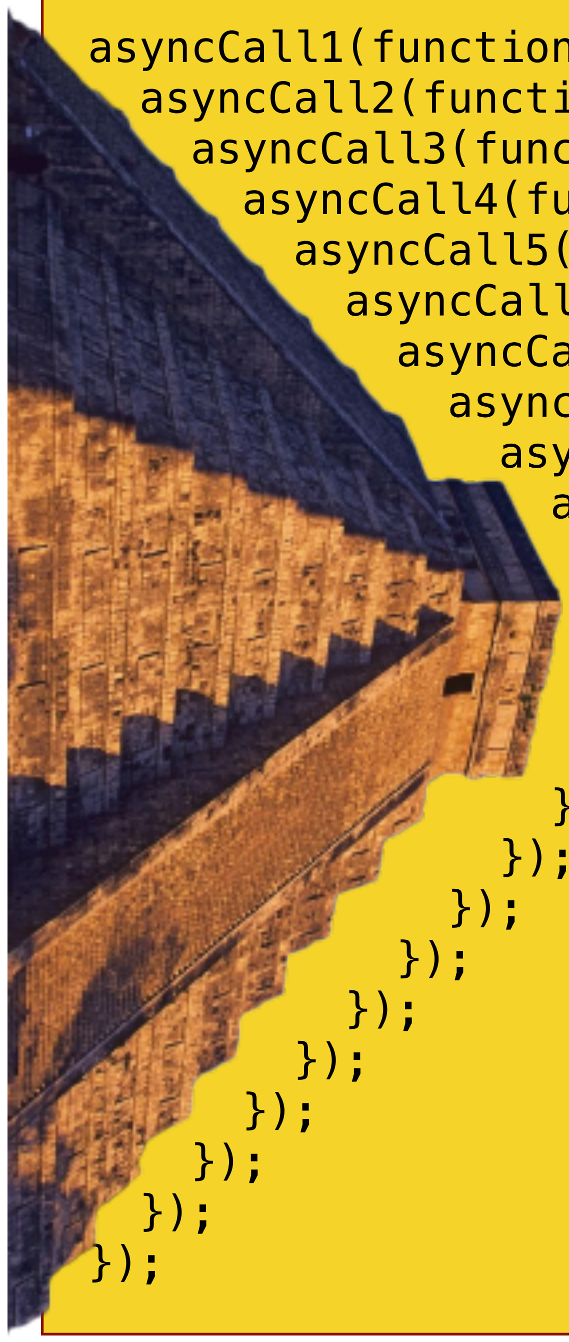
Entering the **third** asynchronous callback function.

What if I have 12 asynchronous calls?

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

We call this
the **pyramid**
of doom

(Also known as the **callback hell**.)



```
asyncCall1(function(err, result1) {
  asyncCall2(function(err, result2) {
    asyncCall3(function(err, result3) {
      asyncCall4(function(err, result4) {
        asyncCall5(function(err, result5) {
          asyncCall6(function(err, result6) {
            asyncCall7(function(err, result7) {
              asyncCall8(function(err, result8) {
                asyncCall9(function(err, result9) {
                  asyncCall10(function(err, result10) {
                    asyncCall11(function(err, result11) {
                      asyncCall12(function(err, result12) {
                        // finally...
                      });
                    });
                  });
                });
              });
            });
          });
        });
      });
    });
  });
});
```

How do I fix it?

Async is a utility module which provides straight-forward, powerful functions for working with **asynchronous JavaScript**.

Async provides around 20 functions that include the usual '**functional**' suspects (map, reduce, filter, each…) as well as some common patterns for **asynchronous control flow** (parallel, series, waterfall…). All these functions assume you follow the **Node.js convention** of providing a single callback as the last argument of your async function.

https://github.com/caolan/async

Running sequential asynchronous calls

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

`waterfall(tasksArray, callback)`

```
async.waterfall([
  function(callback) {
    callback(null, 'one', 'two');
  },
  function(arg1, arg2, callback) {
    // arg1 now equals 'one' and arg2 now equals 'two'
    callback(null, 'three');
  },
  function(arg1, callback) {
    // arg1 now equals 'three'
    callback(null, 'done');
  }
], function(err, result) {
  // result now equals 'done'
});
```

Runs the tasks array of functions **in series**, each **passing their results to the next** in the array. However, if any of the tasks pass an error to their own callback, the next function is not executed, and the main callback is **immediately called with the error**.

Running sequential asynchronous calls

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

`waterfall(tasksArray, callback)`

```
async.waterfall([
  function(callback) {
    callback(null, 'one', 'two');
  },
  function(arg1, arg2, callback) {
    // arg1 now equals 'one' and arg2 now equals 'two'
    callback(null, 'three');
  },
  function(arg1, callback) {
    // arg1 now equals 'three'
    callback(null, 'done');
  }
], function(err, result) {
  // result now equals 'done'
});
```

These 3 functions are called **in sequence**, one after the other. Each receives a **callback** function that they should call with either an error or the result.

This is the **final** callback. If one of the tasks produces an **error**, it is called **immediately** with the error. The other tasks are canceled. Otherwise, it is called with the **result** of the last task.

Running sequential asynchronous calls

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

`waterfall(tasksArray, callback)`

```
async.waterfall([
  function(callback) {
    callback(null, 'one', 'two');
  },
  function(arg1, arg2, callback) {
    // arg1 now equals 'one' and arg2 now equals 'two'
    callback(null, 'three');
  },
  function(arg1, callback) {
    // arg1 now equals 'three'
    callback(null, 'done');
  }
], function(err, result) {
  // result now equals 'done'
});
```

With **waterfall**, each **task** receives the **results of the previous task** as arguments.

Running asynchronous calls in parallel

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

`parallel(tasksArray, callback)`

```
async.parallel([
  function(callback){
    setTimeout(function(){
      callback(null, 'one');
    }, 200);
  },
  function(callback){
    setTimeout(function(){
      callback(null, 'two');
    }, 100);
  }
], function(err, results){
  // the results array will equal ['one','two']
  // even though the second function had a
  // shorter timeout.
});
```

Run the tasks array of functions **in parallel**, without waiting until the previous function has completed. If any of the functions pass an error to its callback, the main callback is **immediately called with the error**. Once the tasks have completed, **the results are passed to the final callback as an array**.

Running asynchronous calls in parallel

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

`parallel(tasksArray, callback)`

```
async.parallel([
  function(callback){
    setTimeout(function(){
      callback(null, 'one');
    }, 200);
  },
  function(callback){
    setTimeout(function(){
      callback(null, 'two');
    }, 100);
  }
], function(err, results){
  // the results array will equal ['one','two']
  // even though the second function had a
  // shorter timeout.
});
```

These 2 functions are run **in parallel**. Each receives a **callback** function that they should call with either an error or the result.

This is the **final** callback. If one of the tasks produces an **error**, it is called **immediately** with the error. The other tasks are canceled. Otherwise, it is called with an array containing the **results** of all tasks.

Running asynchronous calls in parallel

`parallel(tasksArray, callback)`

```
async.parallel([
  function(callback){
    setTimeout(function(){
      callback(null, 'one');
    }, 200);
  },
  function(callback){
    setTimeout(function(){
      callback(null, 'two');
    }, 100);
  }
], function(err, results){
  // the results array will equal ['one','two']
  // even though the second function had a
  // shorter timeout.
});
```

Even if the tasks finish out of order, the results will always be given **in the same order as the tasks**.

Back to our example

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
Book.count(function(err, totalCount) {
  // handle error

  Book.count(criteria, function(err, filteredCount) {
    // handle error

    var query = Book
      .find(criteria)
      .sort('title')
      .skip(offset)
      .limit(limit);

    if (req.query.embed == 'publisher') {
      query = query.populate('publisher');
    }

    query.exec(function(err, books) {
      // handle error

      res.set('X-Pagination-Page', page);
      res.set('X-Pagination-Page-Size', pageSize);
      res.set('X-Pagination-Total', totalCount);
      res.set('X-Pagination-Filtered-Total', filteredCount);

      res.send(books);
    });
  });
});
```

We have 3 tasks: **counting all books**, **counting matching books**, and **finding matching books**.

These tasks have no dependency on each other: they don't need to wait for each other's result before being executed. They could be executed **in parallel**.

Back to our example

```
Book.count(function(err, totalCount) {
  // handle error

  Book.count(criteria, function(err, filteredCount) {
    // handle error

    var query = Book
      .find(criteria)
      .sort('title')
      .skip(offset)
      .limit(limit);

    if (req.query.embed == 'publisher') {
      query = query.populate('publisher');
    }

    query.exec(function(err, books) {
      // handle error

      res.set('X-Pagination-Page', page);
      res.set('X-Pagination-Page-Size', pageSize);
      res.set('X-Pagination-Total', totalCount);
      res.set('X-Pagination-Filtered-Total', filteredCount);

      res.send(books);
    });
  });
});
```

We will extract each of the three asynchronous operations into **its own function**, and we will use async's **parallel** operation to execute them in parallel.

```
async.parallel([
  countAllBooks,
  countFilteredBooks,
  findMatchingBooks
], sendResponse);
```

We will also create a **final callback function (sendResponse)** that will handle sending the response (the error or the result).

Counting all books (task 1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
Book.count(function(err, totalCount) {
  if (err) {
    res.status(500).send(err);
    return;
  }

  // ...
});
```

```
function countAllBooks(callback) {
  Book.count(function(err, totalCount) {
    if (err) {
      callback(err);
    } else {
      callback(undefined, totalCount);
    }
  });
}
```

Our new function takes a **callback** as argument.

If there's an error, instead of handling the response here, we simply call the callback with the error. Async's **parallel** will automatically forward it to the **final callback function**.

If all went well, we call the callback with the **result**.

Counting matching books (task 2)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
Book.count(criteria, function(err, filteredCount) {
  if (err) {
    res.status(500).send(err);
    return;
  }

  // ...
});
```

```
function countFilteredBooks(callback) {
  Book.count(criteria, function(err, filteredCount) {
    if (err) {
      callback(err);
    } else {
      callback(undefined, filteredCount);
    }
  });
}
```

This second task also takes a callback as argument.

Again, we simply give the error to the callback if there is one.

If all went well, we call the callback with the **result**.

Finding matching books (task 3)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
var query = Book
  .find(criteria).sort('title').skip(offset).limit(limit);

if (req.query.embed == 'publisher') {
  query = query.populate('publisher');
}

query.exec(function(err, books) {
  if (err) {
    res.status(500).send(err);
    return;
  }

  // ...
});
```

```
function findMatchingBooks(callback) {
  var query = Book
    .find(criteria).sort('title').skip(offset).limit(limit);

    if (req.query.embed == 'publisher') {
      query = query.populate('publisher');
    }

    query.exec(function(err, books) {
      if (err) {
        callback(err);
      } else {
        callback(undefined, books);
      }
    });
  }
}
```

This third task also takes a callback as argument.

Again, we simply give the error to the callback if there is one.

If all went well, we call the callback with the **result**.

The final callback function

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

If **any** of the tasks **failed**, the final callback function will receive the **error**.

Otherwise, it will receive the **results** from the **3 tasks**.

```
function sendResponse(err, results) {
  if (err) {
    res.status(500).send(err);
    return;
  }

  var totalCount = results[0],
      filteredCount = results[1],
      books = results[2];

  res.set('X-Pagination-Page', page);
  res.set('X-Pagination-Page-Size', pageSize);
  res.set('X-Pagination-Total', totalCount);
  res.set('X-Pagination-Filtered-Total', filteredCount);

  res.send(books);
}
```

We handle sending an error to the user here. This code was repeated 3 times before.

We handle sending all response data here, including the pagination headers.

Begone, pyramid of doom

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
router.get('/', function(req, res, next) {

  var criteria = {};

  // Build filtering criteria...
  // Prepare pagination data...

  function countAllBooks(callback) {
    // ...
  }

  function countFilteredBooks(callback) {
    // ...
  }

  function findMatchingBooks(callback) {
    // ...
  }

  function sendResponse(err, results) {
    // ...
  }

  async.parallel([
    countAllBooks,
    countFilteredBooks,
    findMatchingBooks
  ], sendResponse);
});
```

All our task functions are **"flat"** now. We have gotten rid of the nested callbacks.

Finally, async's **parallel** will handle the control flow for us.

More…

Full example on GitHub