Fork me on GitHub

Series map (async.mapSeries)

In the previous example we ran the same async task on multiple items in parallel. Each task downloaded an item then extracted its name.

What if we want to wait for the current item to download first before starting the next download?

Callbacks

With callbacks we need to use an entirely different function - mapSeries:

```
async.mapSeries(ids, function(id, callback) {
    getFromStorage(id, function (err, res) {
        if (err) return callback(err);
        callback(null, res.name);
    })
}, function(err, results) {
    // results is an array of names
});
```

Promises

With promises, we just say what we mean. Create a promise that first waits for the current download to complete, then download the next item, then extract its name.

```
// start with current being an "empty" already-fulfilled promise
var current = Promise.resolve();

Promise.all(ids.map(function(id) {
```

```
        current = current.then(function() {
            return getItem(id) // returns promise
        }).then(function(result) {
            return result.name;
        });
        return current;
    })).then(function(results) {
        // results is an array of names
    })
```

Again, `Promise.all` does the rest.

Notes

This example illustrates how promises compose really well. Instead of using a different utility function every time we encounter a new problem, we can simply express the specifics of that new problem clearly by specifying dependencies between our tasks. The reason why its easier? Because the tasks are values, and because those values have a powerful composition function that creates new tasks depending on old tasks - `.then`.

If we want to control the level of parallelism, then we will need to change our code again