

Advanced Javascript

Thierry Sans

Outline

- Dealing with **asynchronism** (promises and async/await)
- Web workers

Asynchronism

Callback - the building block for asynchronism

```
fs.readFile(filepath, 'utf8', function (err, data) {  
    if (err) console.log(err);  
    return console.log(data);  
});
```

Defining a promise

```
var readFile = function(filepath){  
    return new Promise(function(resolve, reject){  
        fs.readFile(filepath, 'utf8', function (err, data) {  
            if (err) reject(err);  
            return resolve(data);  
        });  
    });  
}
```


Calling a promise

```
readFile(filepath)
  .then(function(data){
    console.log(data);
  })
  .catch(function(err){
    console.log(data);
  });
```

Calling a promise with async/await

```
async function run() {  
    var data = await readFile(filepath);  
    console.log(data);  
};  
  
run().catch(err => console.error(err));
```

Web Workers

<http://afshinm.github.io/50k/>

Web Workers for parallelism

- Create threads in Javascript (now frontend and backend)
- These threads can run in parallel (separate event loop)

What a web worker can/cannot do

✓ XMLHttpRequest

⦿ window

✓ indexedDB

⦿ document (not thread safe)

✓ location (read only)

Create a web worker

doSomething.js

```
function(){  
  "use strict";  
  
  // receive message  
  self.addEventListener('message', function(e){  
    var data = e.data;  
    // send the same data back  
    self.postMessage(data);  
  }, false);  
});
```

Instantiate a web worker

main.js

```
var worker = new Worker('doSomething.js');

// sending a message to the web worker
worker.postMessage({myList:[1, 2, 3, 4]});

// receive message from web worker
worker.addEventListener('message', function(e) {
    console.log(e.data);
}, false);
```