# Async & JS

A walkthrough common asynchronous patterns for the client, the server and the Internet Of Things

by Andrea Giammarchi

@WebReflection

# quick story about me

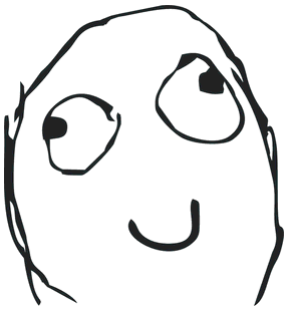
- Created first offline HTML5 Map navigation in 2011 ( async WebSQL based tileServer, like ServiceWorkers, but in production 4 years ago )
- Worked on high traffic fully-async Mobile and Desktop Web applications ( fb, twitter, TweetDeck )
- Maniac about performance on constrained environments such ARM, MIPS, or x86 boards

# quick story about me

- Created first offline HTML5 Map navigation in 2011 ( async WebSQL based tileServer, like ServiceWorkers, but in production 4 years ago )
- Worked on high traffic fully-async Mobile and Desktop Web applications ( fb, twitter, TweetDeck )
- Maniac about performance on constrained environments such ARM, MIPS, or x86 boards
- often complaining about everything I don't understand as developer on es-discuss

# quick story about me

- Created first offline HTML5 Map navigation in 2011 ( async WebSQL based tileServer, like ServiceWorkers, but in production 4 years ago )
- Worked on high traffic fully-async Mobile and Desktop Web applications ( fb, twitter, TweetDeck )
- Maniac about performance on constrained environments such ARM, MIPS, or x86 boards
- often complaining about everything I don't understand as developer on es-discuss



# Buzzboard

- XHR
- Events
- Promises
- Generators
- Standards, fetch, autocomplete, network ...

# XHR

```
function getContentType(url, callback) {
  var xhr = new XMLHttpRequest;
  xhr.onload = function () {
    callback(this.getResponseHeader('content-type'));
  };
  xhr.open('HEAD', url, true);
  xhr.send(null);
}
```

# XHR

```
function getContentType(url, callback) {
  var xhr = new XMLHttpRequest;
  xhr.onload = function () {
    callback(this.getResponseHeader('content-type'));
  };
  xhr.open('HEAD', url, true);
  xhr.send(null);
}

getContentType('?xhr', function (contentType) {
  console.log(contentType);
});
```

# XHR

```
function getContentType(url, callback) {
  var xhr = new XMLHttpRequest;
  xhr.onerror = function (e) { callback(e, null); };
  xhr.onload = function () {
    callback(null, xhr.getResponseHeader('content-type'));
  };
  xhr.open('HEAD', url, true);
  xhr.send(null);
}

getContentType('?xhr', function (err, result) {
  console.log(err || result);
});
```

# Events

```
function getContentType(url, callback) {
  var xhr = new XMLHttpRequest;
  xhr.addEventListener('error', function (e) { callback(e, null); });
  xhr.addEventListener('load', function () {
    callback(null, xhr.getResponseHeader('content-type'));
  });
  xhr.open('HEAD', url, true);
  xhr.send(null);
  return xhr;
}

getContentType('?xhr', function (err, result) {
  console.log(err || result);
}).onload = function (pe) {
  // do something else ...
};
```

# Promises

# Promises

```
function getContentType(url) {
  return new Promise(function (resolve, reject) {
    var xhr = new XMLHttpRequest;
    xhr.onerror = reject;
    xhr.onload = function () {
      resolve(xhr.getResponseHeader('content-type'));
    };
    xhr.open('HEAD', url, true);
    xhr.send(null);
  });
}

getContentType('?promise')
  .then(function (result) { console.log(result); })
  .catch(function (err) { console.warn(err); })
;
```

# Promises

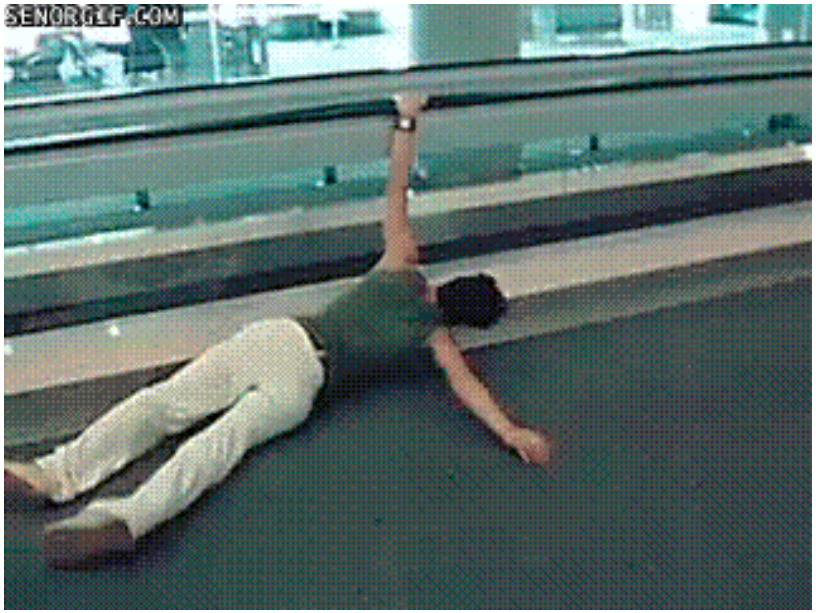
- where is the progress ?

# Promises

- where is the progress ?

# Promises

- where is the progress ?



# Promises

- can I cancel that request ?

# Promises

- can I cancel that request ?

# Promises

- can I cancel that request ?

# Promises

- can I cancel that request ?

# Promises

- Rationale: `xhr.abort();` is an explicit intent that triggers an `'abort'` event. It is not an error, it is not a completed operation, it's very often needed but is not possible via Promises (yet)

# Promises

- Rationale: `xhr.abort();` is an explicit intent that triggers an `'abort'` event. It is not an error, it is not a completed operation, it's very often needed but is not possible via Promises (yet)
- Dozen of different discussions all over the web about how to cancel, when, why, and how again

# Promises

- Rationale: `xhr.abort();` is an explicit intent that triggers an `'abort'` event. It is not an error, it is not a completed operation, it's very often needed but is not possible via Promises (yet)
- Dozen of different discussions all over the web about how to cancel, when, why, and how again
- …fetch on autocomplete as very basic example

# autocomplete

- Shorter is the text, slower the result.

# autocomplete

- Shorter is the text, slower the result.
- Searches start shorter, slower comes later

# autocomplete

- Shorter is the text, slower the result.
- Searches start shorter, slower comes later

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console

Preserve log | Disable cache

| Name<br>Path | Method | Status<br>Text | Type | Initiator | Size<br>Content | Time<br>Latency | Timeline<br>400 ms |
|---|---|---|---|---|---|---|---|
| s?sclient=psy-ab&biw=1600&bih=506&q=t&oq=&gs_l… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 528 B<br>766 B | 97 ms<br>96 ms | |
| s?sclient=psy-ab&biw=1600&bih=506&q=tr&oq=&gs_l… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 493 B<br>778 B | 81 ms<br>79 ms | |
| s?sclient=psy-ab&biw=1600&bih=506&q=tro&oq=&gs… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 620 B<br>1.2 KB | 71 ms<br>70 ms | |
| gen_204?atyp=i&ct=1&cad=1&rsm=6&ei=0kLzVMOm… | GET | 204<br>No Content | text/html | Other | 35 B<br>0 B | 42 ms<br>42 ms | |

# autocomplete

- Shorter is the text, slower the result.
- Searches start shorter, slower comes later

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console

Preserve log | Disable cache

| Name<br>Path | Method | Status<br>Text | Type | Initiator | Size<br>Content | Time<br>Latency | Timeline<br>400 ms |
|---|---|---|---|---|---|---|---|
| s?sclient=psy-ab&biw=1600&bih=506&q=t&oq=&gs_l… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 528 B<br>766 B | 97 ms<br>96 ms | |
| s?sclient=psy-ab&biw=1600&bih=506&q=tr&oq=&gs_l… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 493 B<br>778 B | 81 ms<br>79 ms | |
| s?sclient=psy-ab&biw=1600&bih=506&q=tro&oq=&gs… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 620 B<br>1.2 KB | 71 ms<br>70 ms | |
| gen_204?atyp=i&ct=1&cad=1&rsm=6&ei=0kLzVMOm… | GET | 204<br>No Content | text/html | Other | 35 B<br>0 B | 42 ms<br>42 ms | |

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console

Preserve log | Disable cache

| Name<br>Path | Method | Status<br>Text | Type | Initiator | Size<br>Content | Time<br>Latency | Timeline<br>100 ms 150 ms 200 ms 250 ms |
|---|---|---|---|---|---|---|---|
| s?sclient=psy-ab&site=&source=hp&q=i&oq=&gs_l=&… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 509 B<br>797 B | 111 ms<br>109 ms | |
| s?sclient=psy-ab&site=&source=hp&q=in&oq=&gs_l=… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 510 B<br>808 B | 91 ms<br>90 ms | |
| gen_204?atyp=i&ct=1&cad=1&rsm=6&ei=q0PzVJ_1O… | GET | 204<br>No Content | text/html | Other | 35 B<br>0 B | 44 ms<br>43 ms | |
| s?sclient=psy-ab&site=&source=hp&q=intra&oq=&gs_… | GET | 200<br>OK | application/json | rs=ACT90oFCgXQyU7BNv…<br>Script | 505 B<br>814 B | 61 ms<br>61 ms | |

# Fetch API

- A Promise based XHRish like API whatwg proposal: https://fetch.spec.whatwg.org

# Fetch API

- A Promise based XHRish like API whatwg proposal: https://fetch.spec.whatwg.org

## 5 Fetch API

The `fetch()` method is relatively low-level API for fetching resources. It covers slightly more ground than `XMLHttpRequest`, although it is currently lacking when it comes to reporting progression.

Example

The `fetch()` method makes it quite straightforward to fetch a resource and extract its contents as a `Blob`:

```
fetch("/music/pk/altes-kamuffel.flac")
  .then(res =>
res.blob()).then(playBlob)
```

# Fetch API

- A Promise based XHRish like API whatwg proposal: https://fetch.spec.whatwg.org

## 5 Fetch API

The `fetch()` method is relatively low-level API for fetching resources. It covers slightly more ground than `XMLHttpRequest`, although it is currently lacking when it comes to reporting progression. **… AND YOU CANNOT CANCEL IT !!!**

Example

The `fetch()` method makes it quite straightforward to fetch a resource and extract its contents as a `Blob`:

```
fetch("/music/pk/altes-kamuffel.flac")
  .then(res =>
res.blob()).then(playBlob)
```

# Fetch API

- A Promise based XHRish like API whatwg proposal: https://fetch.spec.whatwg.org

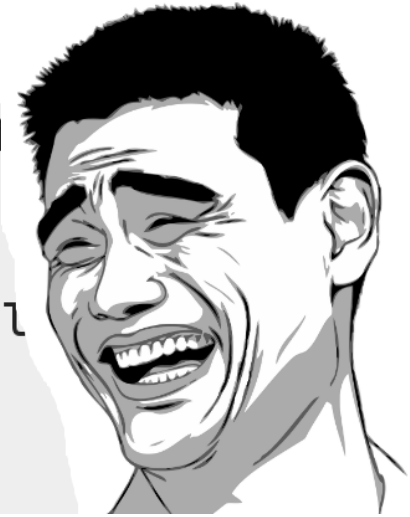
## 5 Fetch API

The `fetch()` method is relatively low-level API for fetching resources. It covers slightly more ground than `XMLHttpRequest`, although it is currently lacking when it comes to reporting progression. **… AND YOU CANNOT CANCEL IT !!!**

Example

The `fetch()` method makes it quite straightf
fetch a resource and extract its contents as a

```
fetch("/music/pk/altes-kamuffel
  .then(res =>
res.blob()).then(playBlob)
```



# Fetch API

- Good for actions that do not require progress indication
- Good for operations performed behind the scene
- Good for one shot / quick read of some lightweight data

# Fetch API

- Good for actions that do not require progress indication
- Good for operations performed behind the scene
- Good for one shot / quick read of some lightweight data
- But ****not**** necessarily better than XHR

# Generators

# Generators

# Generators

```
function* getTextContent(url) {
  var xhr = new XMLHttpRequest;
  xhr.open('GET', url, true);
  xhr.send(null);
  while (xhr.readyState != 4) {
    yield Math.floor(((xhr.loaded / xhr.total) || 0) * 100) + '%';
  }
  yield xhr.responseText;
}
```

# Generators

```
function* getTextContent(url) {
  var xhr = new XMLHttpRequest;
  xhr.open('GET', url, true);
  xhr.send(null);
  while (xhr.readyState != 4) {
    yield Math.floor(((xhr.loaded / xhr.total) || 0) * 100) + '%';
  }
  yield xhr.responseText;
}

(function loader(gen) {
  var status = gen.next();
  console.log(status.value);
  if (!status.done) setTimeout(loader, 33, gen);
}(getTextContent('?generator')));
```

# Generators

```
function* getTextContent(url) {
  var xhr = new XMLHttpRequest;
  xhr.open('GET', url, true);
  xhr.send(null);
  while (xhr.readyState != 4) {
    yield Math.floor(((xhr.loaded / xhr.total) || 0) * 100) + '%';
  }
  yield xhr.responseText;
}

(function loader(gen) {
  var status = gen.next();
  console.log(status.value);
  if (!status.done) setTimeout(loader, 33, gen);
}(getTextContent('?generator')));
```

ANTIPATTERN

# Generators

```
var getTextContent = async(function* (url) {
  var value = yield fetch(url);
  return value;
});

getTextContent('?generator')
  .then(function (value) {
    console.log(value);
  })
  .catch(function (error) {
    console.warn(error);
  });
```

# Generators

```
function async(generator) {
  return function () {
    var
      g = generator.apply(this, arguments),
      handle = function (op) {
        var p = Promise.resolve(op.value);
        return op.done ? p : p.then(next, fail);
      },
      next = function (v) { return handle(g.next(v)); },
      fail = function (e) { return handle(g.throw(e)); }
    ;
    try { return next(null); } catch (e) {
      return Promise.reject(e);
    }
  };
} // borrowed and modified from https://www.promisejs.org/generators/
```

# Generators

```
function later() {
  return (later.promise = new Promise(function (resolve, reject) {
    later.resolve = resolve;
    later.reject = reject;
  }));
}
```

# Generators

```
function later() {
  return (later.promise = new Promise(function (resolve, reject) {
    later.resolve = resolve;
    later.reject = reject;
  }));
}

function* createGenerator() {
  console.log('spinned');
  later.value = yield later();
  console.log(later.value);
}
```

# Generators

```
function later() {
  return (later.promise = new Promise(function (resolve, reject) {
    later.resolve = resolve;
    later.reject = reject;
  }));
}

function* createGenerator() {
  console.log('spinned');
  later.value = yield later();
  console.log(later.value);
}

var g = createGenerator(); // nothing logged
```

# Generators

```
function later() {
  return (later.promise = new Promise(function (resolve, reject) {
    later.resolve = resolve;
    later.reject = reject;
  }));
}

function* createGenerator() {
  console.log('spinned');
  later.value = yield later();
  console.log(later.value);
}

var g = createGenerator(); // nothing logged
g.next(); // spinned, {value: Promise, done: false}
```

# Generators

```
function later() {
  return (later.promise = new Promise(function (resolve, reject) {
    later.resolve = resolve;
    later.reject = reject;
  }));
}

function* createGenerator() {
  console.log('spinned');
  later.value = yield later();
  console.log(later.value);
}

var g = createGenerator(); // nothing logged
g.next(); // spinned, {value: Promise, done: false}

later.promise.then(function (value) {
  g.next(value); // {value: undefined, done: true}
});
```

# Generators

```
function later() {
  return (later.promise = new Promise(function (resolve, reject) {
    later.resolve = resolve;
    later.reject = reject;
  }));
}

function* createGenerator() {
  console.log('spinned');
  later.value = yield later();
  console.log(later.value);
}

var g = createGenerator(); // nothing logged
g.next(); // spinned, {value: Promise, done: false}

later.promise.then(function (value) {
  g.next(value); // {value: undefined, done: true}
});
later.resolve(Math.random());
```

# Generators

```
function async(generator) {
  return function () {
    var
      g = generator.apply(this, arguments),
      handle = function (op) {
        var p = Promise.resolve(op.value);
        return op.done ? p : p.then(next, fail);
      },
      next = function (v) { return handle(g.next(v)); },
      fail = function (e) { return handle(g.throw(e)); }
    ;
    try { return next(null); } catch (e) {
      return Promise.reject(e);
    }
  };
} // borrowed and modified from https://www.promisejs.org/generators/
```

# ...and what about progress?

# Events + Generators + Promises

```
function loadWithProgress(url, onprogress) {
  return new Promise(function (resolve, reject) {
    var xhr = new XMLHttpRequest;
    xhr.addEventListener('load', function (pe) { resolve(xhr); });
    xhr.addEventListener('error', reject);
    xhr.addEventListener('progress', onprogress);
    xhr.open('GET', url, true);
    xhr.send(null);
  });
}
```

# Events + Generators + Promises

```
function loadWithProgress(url, onprogress) {
  return new Promise(function (resolve, reject) {
    var xhr = new XMLHttpRequest;
    xhr.addEventListener('load', function (pe) { resolve(xhr); });
    xhr.addEventListener('error', reject);
    xhr.addEventListener('progress', onprogress);
    xhr.open('GET', url, true);
    xhr.send(null);
  });
}
var load = async(function* (url, onprogress) {
  return yield loadWithProgress(url, onprogress || function (pe) {
    console.log(Math.floor(((pe.loaded / pe.total) || 0) * 100) + '%');
  });
});
```

# Events + Generators + Promises

```
function loadWithProgress(url, onprogress) {
  return new Promise(function (resolve, reject) {
    var xhr = new XMLHttpRequest;
    xhr.addEventListener('load', function (pe) { resolve(xhr); });
    xhr.addEventListener('error', reject);
    xhr.addEventListener('progress', onprogress);
    xhr.open('GET', url, true);
    xhr.send(null);
  });
}
var load = async(function* (url, onprogress) {
  return yield loadWithProgress(url, onprogress || function (pe) {
    console.log(Math.floor(((pe.loaded / pe.total) || 0) * 100) + '%');
  });
});
load('/img/activity-launcher.png').then(function (xhr) {
  console.log(xhr.getResponseHeader('Content-Type'));
});
```

# ...any parallel execution?

# ...any parallel execution?

```
function read(files) {
  return Promise.all(files.map(function (file) {
    return new Promise(function (resolve, reject) {
      fs.readFile(file, function (err, data) {
        if (err) reject(err);
        else resolve(data);
      });
    });
  }));
}

async(function* () {
  var [a, b] = yield read(['a.txt', 'b.txt']);
})();
```

# Async / Await

```
async function getTextContent(url) {
  var value = await fetch(url);
  return value;
}

(async function() {
  console.log(
    getTextContent('?await')
  );
}());
```

# Async & JS

## Thank You!

Andrea Giammarchi

@WebReflection

# Async & JS

## Questions ?

Andrea Giammarchi

@WebReflection