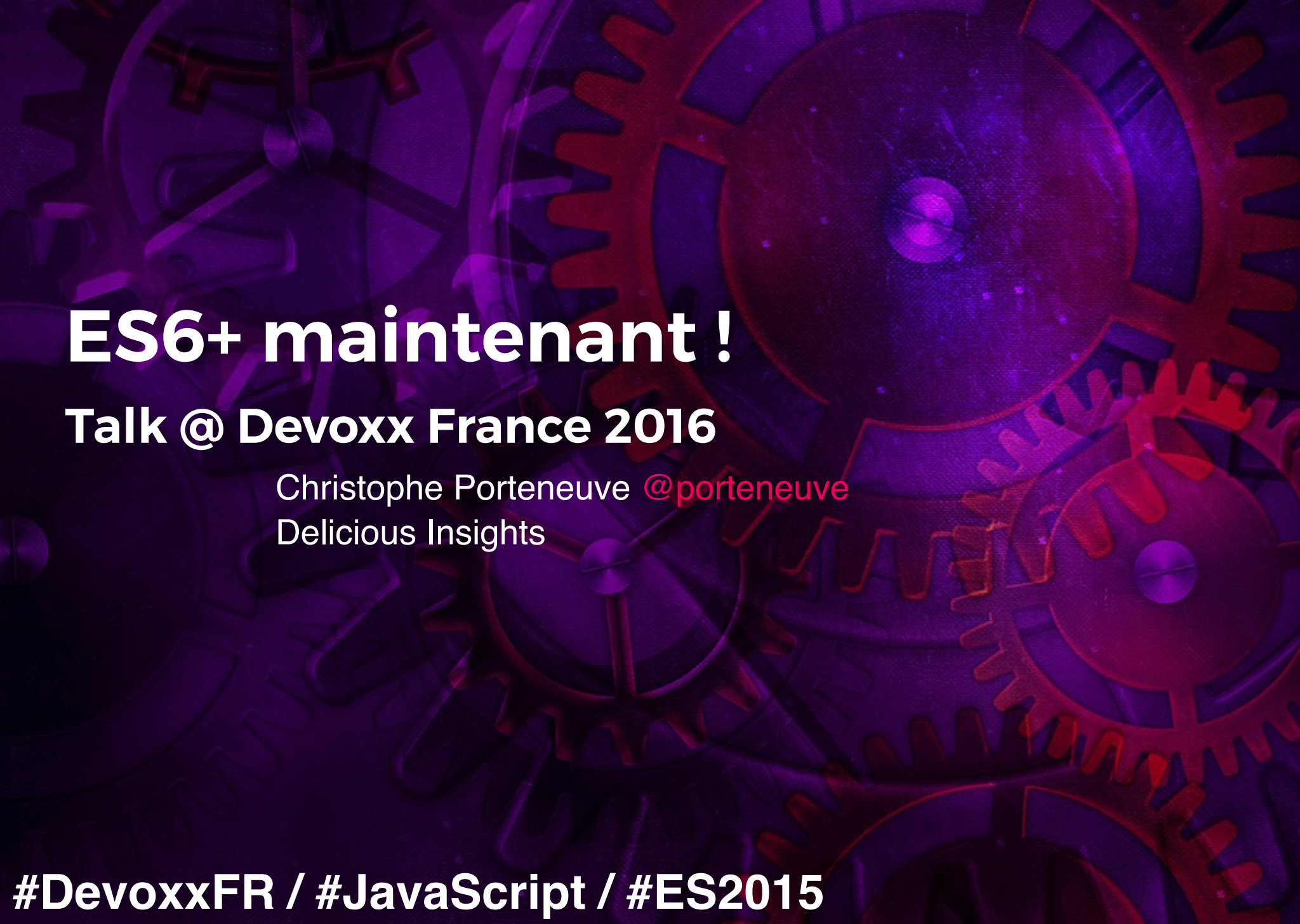


ES6+ maintenant !

Talk @ Devoxx France 2016

Christophe Porteneuve @porteneuve
Delicious Insights

#DevoxxFR / #JavaScript / #ES2015

whoami

```
const christophe = {
  age:         38.459958932238195,
  family:      { wife: 'Élodie', son: 'Maxence' },
  city:        'Paris, FR',
  company:     'Delicious Insights',
  trainings:   ['JS Total', 'Node.js', 'Git Total'],
  jsSince:     1995,

  claimsToFame: [
    'Prototype.js',
    'script.aculo.us',
    'JS Attitude',
    'Bien Développer pour le Web 2.0',
    'NodeSchool Paris'
  ]
}
```

D’ES3 à ES7

mai 1995	JavaScript 1.0 (en 10 jours). Netscape 2.0ß3 en 12/95.
juin 1997	ECMAScript (ES1, ECMA-262), standard officiel
déc. 1999	**ES3**. JScript 1.5 à peu près à ce niveau (IE4–8)
déc. 2009 (!)	**ES5**. *Baseline* de compat’ actuelle. IE9+, Node, etc. Peu de gros changements.
juin 2015	**ES6** / ES2015. **Énormément** de nouveautés de **langage pur**.
juin 2016	ES7 / ES2016 (prévu ; versions annuelles désormais, dans le cadre d’ES.Next)

Prise en charge native ?

La table de compat' de Juriy "Kangax" Zaytsev.

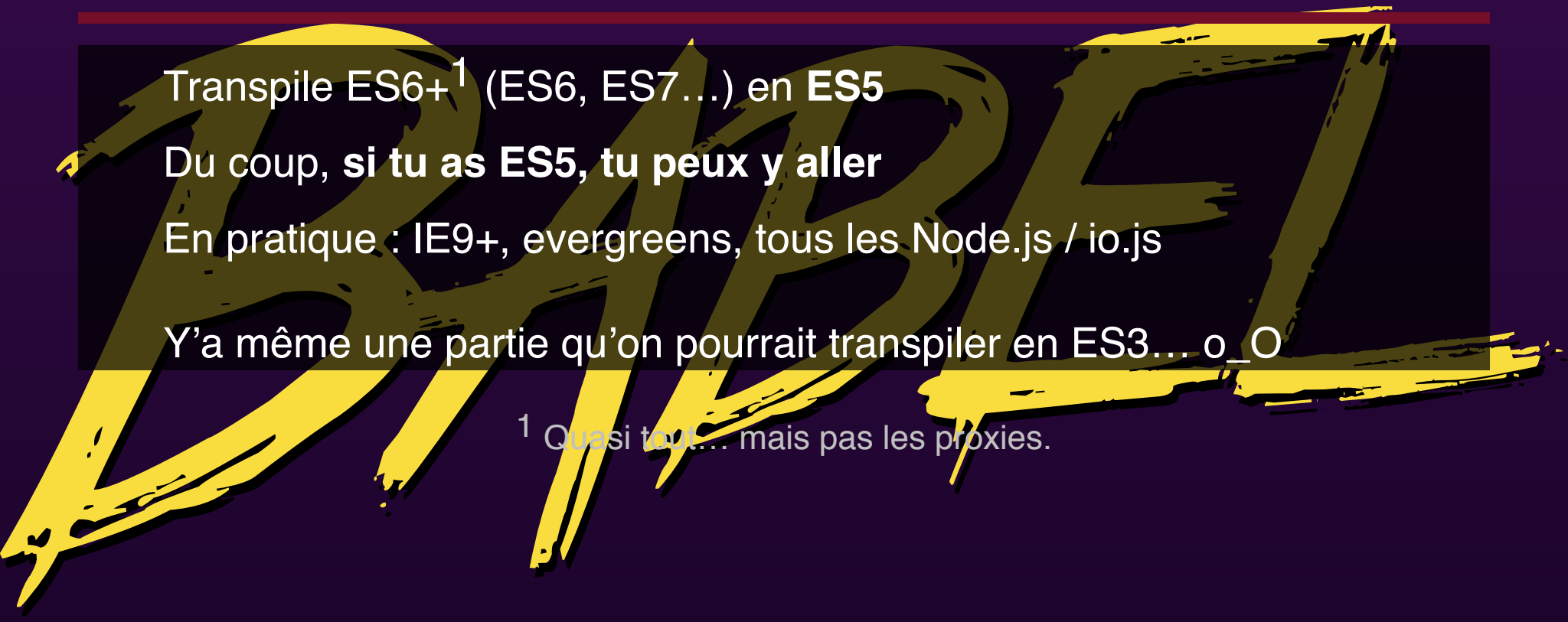
Navigateurs : evergreens **de 90% à 98%**. Safari pourri (53%).

Serveurs : Node LTS (4.4.3) à 48%, Node stable (5.10.1) à 58%, prochain Node LTS (6.0) à **96%** !

Mais évidemment il reste IE (9–11), et puis même 58% ça reste insuffisant tel quel pour être serein-e… Alors on fait quoi, si on n'est pas sur le *bleeding edge* ?

Chiffres au 20/04/2016

Babel



Transpile ES6+[1] (ES6, ES7…) en **ES5**

Du coup, **si tu as ES5, tu peux y aller**

En pratique : IE9+, evergreens, tous les Node.js / io.js

Y’a même une partie qu’on pourrait transpiler en ES3… o_O

[1] Quasi tout… mais pas les proxies.

Babel partout !

Intégré avec **l'univers**

Instructions simples d'utilisation à la volée ou en mode build pour…

- Les **builders** : Grunt, Gulp, Brunch, Webpack, Browserify, jspm…
- Les **runtimes** : Node, RequireJS
- Les **frameworks** : Ember, Rails, Sails…
- Les **harnais de tests** : Mocha, Jasmine, AVA, Karma…
- Les **EDIs et éditeurs** : WebStorm, VS/VSCode, ST, Atom…

ES6, pourquoi ?

Plus **facile**

Plus **puissant**

Plus **fiable**

Plus **performant**

Plus kiffant

Objets & Classes

Là c'est grave plus simple

Littéraux objets

```
function editTodo (id, text) {
  return { type: types.EDIT_TODO, id, text }
}
```

```
function produceFilterTitles (showAll, showActive, showCompleted) {
  return {
    [showAll]: 'All',

    [showActive]: 'Active',
    [showCompleted]: 'Completed'
  }
}
```

Détails

Littéraux objets


```
res.render('entries/index', {
  pageTitle: 'Les bookmarks', entries, entryCount, tags
})
```

```
const coreReducer = combineReducers({
  currentUser, goals, history, today, todaysProgress
})
```

```
return { [action.goalId]: previous + increment }
```

Classes

```
class TodoItem extends Component {
  constructor (props, context) {
    super(props, context)
    this.state = {
      editing: false
    }
  }


  handleDoubleClick () {
    this.setState({ editing: true })
  }
  …
}
```

Détails • Voir sur Babel

Classes (suite)

```
class TodoItem extends Component {
  static MINIMUM_COMPLETION_TIME = 5;

  get completionTime () {
    return (this.checkedAt - this.startedAt) / 1000
  }


  set completionTime (value) {
    this.checkedAt = this.startedAt + Math.max(value, MINIMUM_COMPLETION_TIME) * 1000
  }

  cleanUp () {
    super.cleanUp()
    this.customState = null
  }
}
```

Un confort énorme

sur plein de petits trucs

#DevoxxFR / #JavaScript / #ES2015

Déstructuration

```
const { activeCount } = this.props
…
const { filter: selectedFilter, onShow } = this.props
```

```
var { op: a, lhs: { op: b }, rhs: c } = getASTNode()
```

```
function convertNode ({ nodeType, nodeValue }) {
  switch (nodeType) {
    case ELEMENT_NODE:
      convertElementNode(nodeValue)
      break
    // …
  }
}
```

```
const [, filters] = output.props.children
…
const [,, clear] = output.props.children
```

Détails

Déstructuration

IRL

```
const [
  endpoint = '/users/me',
  output = 'response.json'
] = process.argv.slice(2)
```

```
const HistoryDayGoal = ({ goal: { name, units }, stats: [progress, target] }) => {
  // …

}
```

```
Promise.all([
  Entry.tags(), Entry.count().exec(), Entry.getEntries(req.query)
])
.then(([tags, entryCount, entries]) => {
  res.render('entries/index', { pageTitle: 'Les bookmarks', entries, entryCount, tags })
})
```

#DevoxxFR / #JavaScript / #ES2015

Rest & Spread (itérables)

```
function winners (first, runnerUp, ...others) {
  console.log(first, runnerUp, others)
}
winners('alice', 'bob', 'claire', 'david')
// => 'alice', 'bob', ['claire', 'david']

var arr1 = ['one', 'two']

var arr2 = ['three', 'four']
arr1.push(...arr2) // => 4
arr1 // => ['one', 'two', 'three', 'four']

var nodeListAsArray = [...document.querySelectorAll('h1')]
```

```
return [{ id: …, completed: …, text: … }, ...state]
```

Détails

Rest & Spread (objets)

```
var defaults = { first: 'John', last: 'Doe', age: 42 }
var trainer = { last: 'Smith', age: 35 }
trainer = { ...defaults, ...trainer, age: 36 }
// => { first: 'John', last: 'Smith', age: 36 }
```

```
<TodoItem key={todo.id} todo={todo} {...actions} />
```

Actuellement en *stage 2* pour la spec : ES2017 • Détails

Rest & Spread

IRL

```
componentWillReceiveProps ({ goal }) {
  this.setState({ ...DEFAULT_STATE, ...goal })
}
```

```
case ADD_GOAL: {
  const { name, target, units } = action
  const id = Math.max(...state.map((goal) => goal.id), -1) + 1

  return [...state, { id, name, target, units }]
}
```

Valeurs par défaut

```
function todos (state = initialState, action) {
  // …
}

function convertNode ({ nodeType = ELEMENT_NODE, nodeValue }) {
  // …
}
```

```
var [first, second = 'foo'] = ['yo']
// second === 'foo'

var person = { first: 'Louis', sex: 'M', age: 36 }
var { first, last = 'Doe', age = 42 } = person
// first === 'Louis', last === 'Doe', age === 36
```

Détails

Valeurs par défaut

IRL

```
export default function goals (state = [], action) {
```

```
function goalTrackerReducer (state = coreReducer(undefined, {}), action) {
```

```
export default function today (state = moment().format('YYYY-MM-DD'), action) {
```

```
const DeleteSettingDialog = ({ goal = {}, onCancel, onDelete, open }) => {
```

Template strings

```
var person = { first: 'Thomas', last: 'Anderson', age: 25, nickname: 'Neo' }

// Interpolation de JS quelconque
console.log(`${person.first} aka ${person.nickname}`)
// => 'Thomas aka Neo'

// Multi-ligne !

var markup = `<li>
  ${person.first} ${person.last}, age ${person.age}
</li>`
```

Détails

`let` & `const`

```
const { activeCount } = this.props
…
const { filter: selectedFilter, onShow } = this.props
```

```
const ADD_TODO = 'ADD_TODO'
const DELETE_TODO = 'DELETE_TODO'
const EDIT_TODO = 'EDIT_TODO'
```

Portée : le bloc (même implicite)

```
for (let index = 0; index < 10; ++index)
  setTimeout(() => console.log(index), 10)
  // => Effectivement de 0 à 9, pas 10 x 10, malgré l'asynchrone
```

"`const` is the new `var`". `let` seulement si besoin.

Détails

#DevoxxFR / #JavaScript / #ES2015

`const` ≠ immutable

Attention ! `const` n'est pas récursif / profond !

```
const author = { name: 'Bob', language: 'js' }
author.language = 'es6'

const days = ['tuesday', 'wednesday', 'thursday']
days.push('friday')
```

Pour geler en profondeur, voir deep-freeze.

Dans le même univers : Immutable.js, Mori, Redux

Litéraux étendus

Octaux et binaires (`052` ne marche plus en strict)

```
0o52     // => 42
0b101010 // => 42
```

Encore plus d'Unicode

```
"吉".length == 2                        // Nombre de caractères, pas de codepoints

"\u{20BB7}" == "吉" == "\uD842\uDFB7" // Nouvelle syntaxe d'échappement pour codepoint

"吉".match(/./u)[0].length == 2         // Nouveau flag 'u' des regexp, change les classes

"吉".codePointAt(0) == 0x20BB7          // Nouvelle API codePointAt (en plus de charCodeAt)
```

Détails pour les nombres et l'Unicode

Fonctions fléchées

Ne redéfinit pas `this`*, qui reste donc lexical !

```
<TodoTextInput text={todo.text}
 editing={this.state.editing}
 onSave={(text) => this.handleSave(todo.id, text)} />
```

Raccourcis possibles, idéal pour les prédicats et filtres…

```
TODO_FILTERS = {
  [showAll]: () => true,
  [showActive]: (todo) => !todo.completed,
  [showCompleted]: (todo) => todo.completed
}
…
atLeastOneCompleted = this.props.todos.some((todo) => todo.completed)
```

* Pas plus que `arguments`, `super` et `new.target`. Une fonction fléchée ne peut donc être utilisée comme constructeur et s'abstient en général d'utiliser ces aspects • Détails

Fonctions fléchées


```
[12, 18, 21, 23].map((hour) => [hour, 0, 0])
```

```
return state.map((goal) => goal.id === newGoal.id ? newGoal : goal)
```

```
{goals.map((goal) =>
  <GoalSetting key={goal.id} goal={goal}
    onDeleteClick={() => this.openGoalDeleter(goal)}

    onEditClick={() => this.openGoalEditor(goal)}
  />
)}
```

```
const mapStateToProps = ({ goals, currentUser }) => ({ goals, currentUser })
```

#DevoxxFR / #JavaScript / #ES2015

Des vrais modules

qui défoncent

Exporter

Niveau langage, donc analysables statiquement ; pas de soucis de dépendances circulaires grâce à l'export de *live bindings*.

Mode strict par défaut (comme les corps de classes).

```
// Exports explicites, nommés, à la volée
export function addTodo (text) {
  return { type: types.ADD_TODO, text }
}

export const ADD_TODO = 'ADD_TODO'

// Export par défaut (un seul par module)
export default Footer

// Exports a posteriori
export { Component, Factory, makeHigherOrder }
```

```
// Export renommé
export { each as forEach }

// Ré-export (délégation)
export * from './lib/cool-module'

// ou, plus ciblé :
export {
  makeHigherOrder as wrap,
  Component
} from 'toolkit'
```

Importer

```
// Import intégral dans un « namespace »
import * as types from '../constants/ActionTypes'

// Import nommé de l'export par défaut, imports homonymes d'exports nommés
import React, { PropTypes, Component } from 'react'

// Ou juste un des deux modes :

import classnames from 'classnames'
import { SHOW_ALL, SHOW_COMPLETED, SHOW_ACTIVE } from '../constants/TodoFilters'

// Import renommé
import { makeHigherOrder as wrap } from 'toolkit'
```

Détails • Comparaison CommonJS

Chargement asynchrone

Une API de chargement asynchrone quand on a besoin

```
System.import('toolkit')
  .then(toolkit => …)
  .catch(error => …)

// Plusieurs ? Combinaison de promesses !

Promise.all(['toolkit', 'security', 'advanced'].map(System.import))
.then(([toolkit, security, advanced]) => …)
.catch(error => …)
```

L’asynchrone

en version triviale

#DevoxxFR / #JavaScript / #ES2015

Promesses

Un premier gros exemple…

```
caches.match('/data.json').then(function (response) {
  if (!response) throw Error('No data')
  return response.json()
}).then(function (data) {
  // don't overwrite newer network data

  if (!networkDataReceived) {
    updatePage(data)
  }
}).catch(function () {
  // we didn't get cached data, the network is our last hope:
  return networkUpdate
}).catch(showErrorMessage)
.then(stopSpinner)
```

Promesses ≠ Callbacks++

Trois forces principales

Garanties d'appel unique et exclusif

Capture des exceptions / erreurs

Composabilité

Largement répandu

Plein de libs pré-ES6, jqXHR, APIs web récentes… Et discussion Node.

(ServiceWorker, Fetch, Streams, Push, Notification, Battery, etc.)

Apprivoiser les promesses

L’excellent tuto FR sur HTML5Rocks

Super visualiseur interactif : Promisees

L’atelier interactif NodeSchool

async / await

Le Saint Graal ? Le futur, en tout cas ! *Stage 3* de la spec (ES2017).

Ne remplace absolument pas les promesses ou les générateurs : les **complémente** à merveille, en revanche.

Code asynchrone, non bloquant, mais qui a « l'air synchrone ».

```
async function processUserNotebook (request, response) {
  try {
    const user = await User.get(request.user)
    const notebook = await Notebook.get(user.notebook)
    response.send(await doSomethingAsync(user, notebook))
  } catch (err) {
    response.send(err)
  }
}
```

async / await

On récupère les structures de contrôle : boucles, conditions…

Et on récupère aussi le `try`/`catch` !

```
async function chainAnimationsAsync (elem, animations) {
  let ret = null
  try {
    for (const anim of animations) {
      ret = await anim(elem)
    }
  } catch (e) { /* ignore and keep going */ }
  return ret
}
```

async / await

Attention toutefois à ne pas recréer le monde synchrone et bloquant, en sériant à tort ce qui peut être parallélisé. Pensez aux primitives de promesses, telles `all` et `race` :

```
async function listEntries (request, response) {
  try {
    const [entryCount, entries, tags] = await Promise.all([
      Entry.count().exec(), Entry.getEntries(request.query), Entry.tags()
    ])
    response.render('entries/index', { entryCount, entries, tags })
  } catch (err) {
    response.send(err)
  }
}
```

Spec • Jake Archibald enfonce le clou • Belle présentation de Ross Boucher

#DevoxxFR / #JavaScript / #ES2015

Métaprogrammation

I see what you did there

#DevoxxFR / #JavaScript / #ES2015

Proxies

À la base, on avait surtout besoin d'un équivalent du `__getattr__` de Python ou du `method_missing` de Ruby… On a brièvement eu une tentative de `__noSuchMethod__`, et finalement on a carrément pondu les proxies.

L'idée : interception possible de **toute manipulation** d'un objet.

- Propriétés (donc méthodes) : lire, écrire, définir, supprimer, tester…
- Fonctions : construction avec `new`, appel…
- Prototypes : lire, redéfinir…

On peut **tout faire !**… mais non émulable avec Babel.
Edge 12+, Chrome 49+, Firefox 38+, Node 6.0.

Proxies

L'idée : on construit un proxy qui enrobe l'objet d'origine.

On définit des *traps* pour chaque type de manipulation. L'API `Reflect` nous fournit l'implémentation par défaut

```
function wrapAsDefensive (obj) {
  return new Proxy(obj, {
    get (target, property, receiver) {
      if (!(property in target)) {
        throw new ReferenceError(`Property ${property} missing in ${JSON.stringify(target)}`)
      }

      return Reflect.get(target, property, receiver)
    }
  })
}
```

Décorateurs

ES2017 (ou plus tard) facilite les *higher-order components* avec ce joli morceau de sucre syntaxique à sémantique variable. *Stage 1*…

```
@autobind
  class TodoItem extends Component {
    @memoize('5min')

    getOverallCounts () {
      // …
    }

    @override
    render () {
      // …
    }
  }
```

Détails • Super article

Explorer ES6+ plus avant

ES6-Features.org

ES6 Katas

ES6 In Depth (MDN)

ES6 In Depth (Nicolás Bevacqua)

Exploring ES6

lebab (ex-xto6)

Merci !

Et que JS soit avec vous

Christophe Porteneuve

@porteneuve

Les formations qui tuent tout

Les slides sont sur bit.ly/devoxx-es6

#DevoxxFR / #JavaScript / #ES2015