

JS

ESNext

(re)Découvrir JavaScript



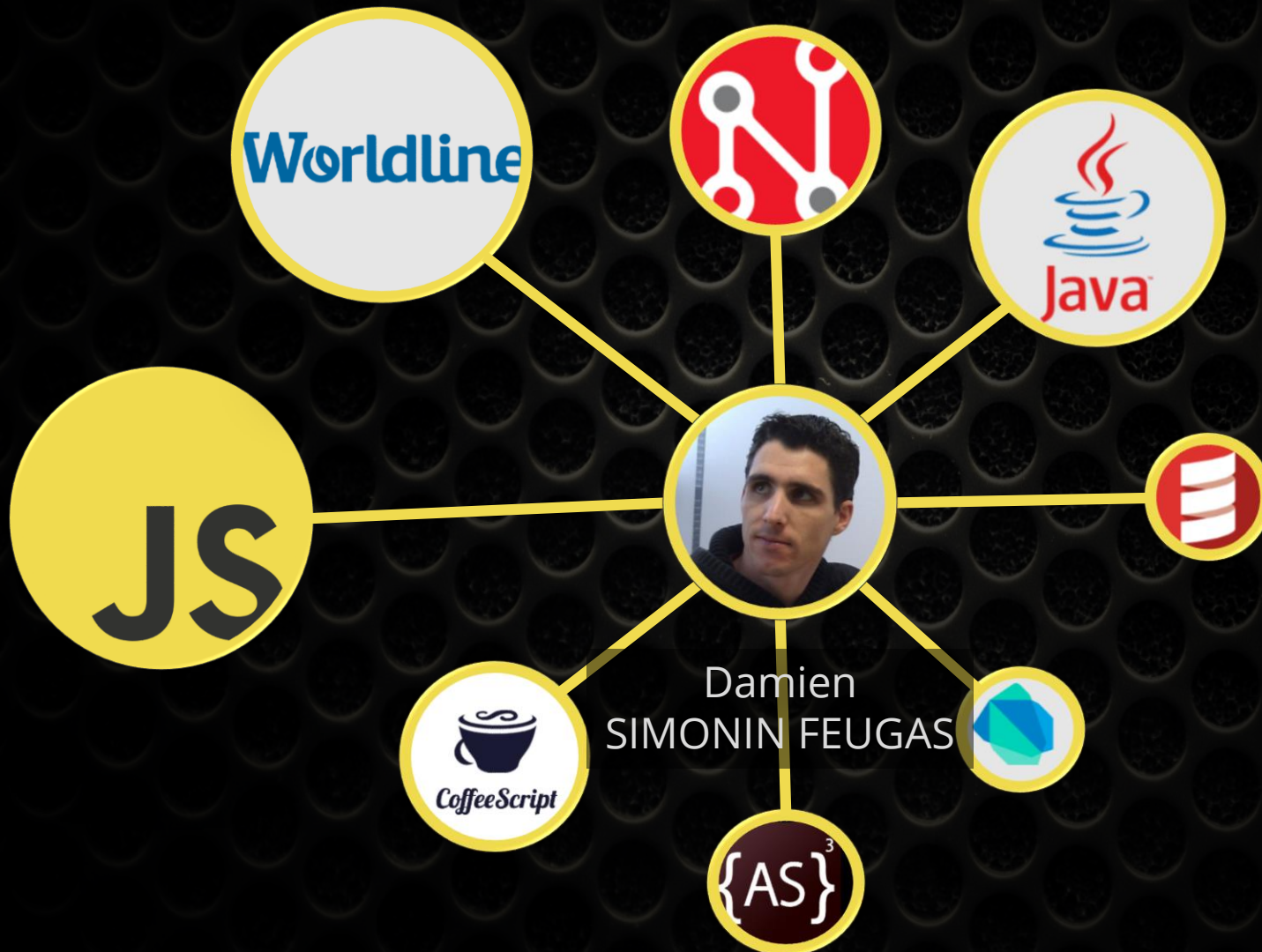
Faisons connaissance

JS

WHO
ARE
YOU?

Faisons connaissance

JS



JS, un langage de programmation :

JS

- **Interprété** - utilise une machine virtuelle
- **Faiblement typé** - valeur sont typées, pas les variables
- **Dynamique** - tout est extensible à l'exécution
- **Multiparadigme** - impératif, fonctionnel, événementiel
- **Mono-thread** - pas de concurrence, pas d'interruption
- **Prototypé** - un objet (prototype) représente la classe



Rapide historique

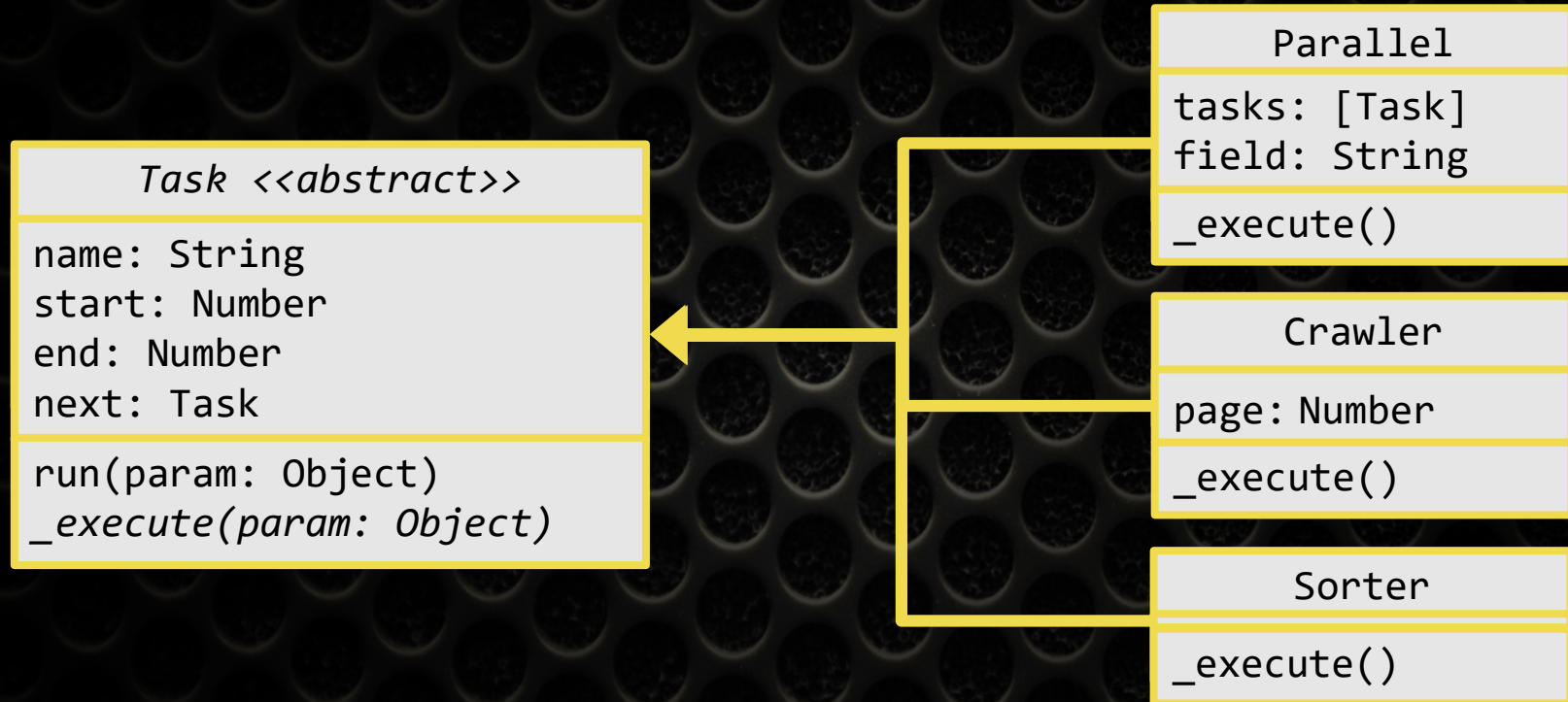
JS



Notre fil rouge

JS

- Ordonnanceur de jobs en NodeJS



- Job d'exemple: crawler les citations sur Chuck Norris

JS

ESNext

Quelques éléments du langage



Portée des variables (block scoping)

JS

```
function greet(name, polite) {  
  if (polite) {  
    var greetings = 'Hi '  
  }  
  return greetings + name + '!'  
}
```

```
greet('Tom', true)    'Hi Tom!'
```


Portée des variables (block scoping)

JS

```
function greet(name, polite) {  
  if (polite) {  
    var greetings = 'Hi '  
  }  
  return greetings + name + '!'  
}
```

`greet('Tom', true)` `'Hi Tom!'`

`greet('Tom', false)` `NaN`

Portée des variables (block scoping)

JS

```
function greet(name, polite) {  
  if (polite) {  
    let greetings = 'Hi '  
  }  
  return greetings + name + '!'  
}
```

```
greet('Tom', true)  ReferenceError
```

```
greet('Tom', false) ReferenceError
```

- `let` restreint la portée d'une variable à son bloc

Portée des variables (block scoping)

JS

```
function greet(name, polite) {  
  if (polite) {  
    let greetings = 'Hi '  
  }  
  return greetings + name + '!'  
}
```

```
greet('Tom', true)  ReferenceError
```

```
greet('Tom', false) ReferenceError
```

- **let** restreint la portée d'une variable à son bloc
- **const** déclare une constante (pas de réaffectation)

```
const maxQuotes = 99  
maxQuotes = 100 >> TypeError: Assignment to constant variable.
```

- Utilisez des constantes autant que faire se peut

```
function greet(name, polite = false) {  
  const greetings = polite ? 'Hi ' : ''  
  return greetings + name  
}
```

Classes

JS

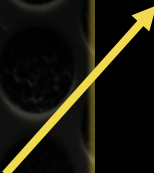
- Déclaration de la classe
 - Tout est public

```
class Task {  
  constructor(name, next = null) {  
    // cf. slide suivant  
  }  
  
  toString() {  
    return this.constructor.name + ' '  
      + this.name  
  }  
  
  get duration() {  
    return this.start  
      ? (this.end || Date.now()) -  
        this.start  
      : -1  
  }  
  
  static display(step) {  
    return task.toString() + (task.next  
      ? ` > ${Task.display(task.next)} `  
      : '')  
  }  
}
```


Classes

JS

- Déclaration de la classe
 - Tout est public
- Un seul constructeur
 - Initialise les attributs




```
class Task {  
  constructor(name, next = null) {  
    // cf. slide suivant  
  }  
  
  toString() {  
    return this.constructor.name + ' '  
      + this.name  
  }  
  
  get duration() {  
    return this.start  
      ? (this.end || Date.now()) -  
        this.start  
      : -1  
  }  
  
  static display(step) {  
    return task.toString() + (task.next  
      ? ` > ${Task.display(task.next)} `  
      : '')  
  }  
}
```

Classes

JS

- Déclaration de la classe
 - Tout est public
- Un seul constructeur
 - Initialise les attributs
- Méthodes d'instance

```
class Task {  
  constructor(name, next = null) {  
    // cf. slide suivant  
  }  
  
  toString() {  
    return this.constructor.name + ' '  
      + this.name  
  }  
  
  get duration() {  
    return this.start  
      ? (this.end || Date.now()) -  
        this.start  
      : -1  
  }  
  
  static display(step) {  
    return task.toString() + (task.next  
      ? ` > ${Task.display(task.next)} `  
      : '')  
  }  
}
```



Classes

JS

- Déclaration de la classe
 - Tout est public
- Un seul constructeur
 - Initialise les attributs
- Méthodes d'instance
- Des getter/setters

```
new Task('t1').duration
```

```
class Task {  
  constructor(name, next = null) {  
    // cf. slide suivant  
  }  
  
  toString() {  
    return this.constructor.name + ' '  
      + this.name  
  }  
  
  get duration() {  
    return this.start  
      ? (this.end || Date.now()) -  
        this.start  
      : -1  
  }  
  
  static display(step) {  
    return task.toString() + (task.next  
      ? ` > ${Task.display(task.next)} `  
      : '')  
  }  
}
```

Classes

JS

- Déclaration de la classe
 - Tout est public

- Un seul constructeur
 - Initialise les attributs

- Méthodes d'instance

- Des getter/setters

```
new Task('t1').duration
```

- Méthodes de classe

```
class Task {  
  constructor(name, next = null) {  
    // cf. slide suivant  
  }  
  
  toString() {  
    return this.constructor.name + ' '  
      + this.name  
  }  
  
  get duration() {  
    return this.start  
      ? (this.end || Date.now()) -  
        this.start  
      : -1  
  }  
  
  static display(step) {  
    return task.toString() + (task.next  
      ? ` > ${Task.display(task.next)} `  
      : '')  
  }  
}
```


Paramètres par défaut et objet littéraux

JS

- Valeur par défaut pour les paramètres

```
const t1 = new Task('t1')  
new Task('t2', t1)  
new Task('t3', undefined)
```

```
constructor(name, next = null) {  
  // ...  
}
```

Paramètres par défaut et objet littéraux

JS

- Valeur par défaut pour les paramètres

```
const t1 = new Task('t1')
new Task('t2', t1)
new Task('t3', undefined)
```

```
constructor(name, next = null) {
  // ...
}
```

- Littéral objet : structure anonyme entre accolades

```
constructor(name, next = null) {
  const state = 'success'
  const attrs = {
    name,
    next,
    end: null,
    start: null,
    [state]: null
  }
  Object.assign(this, attrs)
}
```


Paramètres par défaut et objet littéraux

JS

- Valeur par défaut pour les paramètres

```
const t1 = new Task('t1')
new Task('t2', t1)
new Task('t3', undefined)
```

```
constructor(name, next = null) {
  // ...
}
```

- Littéral objet : structure anonyme entre accolades
- Notation raccourcie: utilise les variables du scope

```
constructor(name, next = null) {
  const state = 'success'
  const attrs = {
    name,
    next,
    end: null,
    start: null,
    [state]: null
  }
  Object.assign(this, attrs)
}
```

Paramètres par défaut et objet littéraux

JS

- Valeur par défaut pour les paramètres

```
const t1 = new Task('t1')
new Task('t2', t1)
new Task('t3', undefined)
```

```
constructor(name, next = null) {
  // ...
}
```

- Littéral objet : structure anonyme entre accolades
- Notation raccourcie: utilise les variables du scope

```
constructor(name, next = null) {
  const state = 'success'
  const attrs = {
    name: name,
    next: next,
    end: null,
    start: null,
    [state]: null
  }
  Object.assign(this, attrs)
}
```


Paramètres par défaut et objet littéraux

JS

- Valeur par défaut pour les paramètres

```
const t1 = new Task('t1')
new Task('t2', t1)
new Task('t3', undefined)
```

```
constructor(name, next = null) {
  // ...
}
```

- Littéral objet : structure anonyme entre accolades
- Notation raccourcie: utilise les variables du scope
- Propriété calculées: le nom est le résultat de l'expression

```
constructor(name, next = null) {
  const state = 'success'
  const attrs = {
    name,
    next,
    end: null,
    start: null,
    [state]: null
  }
  Object.assign(this, attrs)
}
```

Interpolation (template literals)

JS

- Utilisation de variable à l'intérieur des chaînes

```
toString() {  
  const className = this.constructor.name  
  return `${className} ${this.name}`.trim()  
}
```


Interpolation (template literals)

JS

- Utilisation de variable à l'intérieur des chaînes

```
toString() {  
  const className = this.constructor.name  
  return `${className} ${this.name}`.trim()  
}
```

- Utilisation d'expression à l'intérieur des chaînes

```
static display(task) {  
  return task.toString() + ( task.next  
    ? ` > ${Task.display(task.next)} `  
    : ' ' )  
}
```

Interpolation (template literals)

JS

- Utilisation de variable à l'intérieur des chaînes

```
toString() {  
  const className = this.constructor.name  
  return `${className} ${this.name}`.trim()  
}
```

- Utilisation d'expression à l'intérieur des chaînes

```
static display(task) {  
  return task.toString() + ( task.next  
    ? ` > ${Task.display(task.next)} `  
    : ' ' )  
}
```


- "Exécution" d'une chaîne interpolée (tagged literals)

```
const locale = 'fr', company = 'Acme', nb = 3  
l18n`Welcome to ${company}, you are visitor number ${nb}`  
>> Bienvenue à Acme, vous êtes le 3ème visiteur
```

Traitement asynchrone (async/await)

JS

- Déclarer une fonction asynchrone



```
async _execute () {  
    // traitement asynchrone: accès réseau, disque,  
    // processus...  
}
```

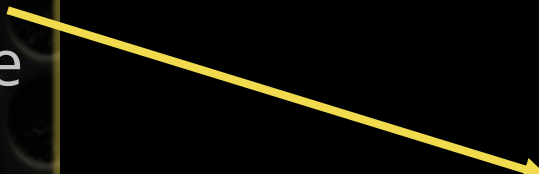

Traitement asynchrone (async/await)

JS

- Déclarer une fonction asynchrone
- Attendre le résultat d'une fonction asynchrone

```
async _execute () {  
  // traitement asynchrone: accès réseau, disque,  
  // processus...  
}
```

```
async run (params = {}) {
```



```
  const results = await this._execute(params)  
  return this.next  
    ? await this.next.run({ ...params, ...results })  
    : results
```

```
}
```

Traitement asynchrone (async/await)

JS

- Déclarer une fonction asynchrone
- Attendre le résultat d'une fonction asynchrone
- Récupérer une erreur

```
async _execute () {  
  // traitement asynchrone: accès réseau, disque,  
  // processus...  
}
```

```
async run (params = {}) {  
  let failure = null  
  
  try {  
    const results = await this._execute(params)  
    return this.next  
      ? await this.next.run({ ...params, ...results })  
      : results  
  } catch (err) {  
    failure = err  
    throw err  
  }  
  
}
```

Traitement asynchrone (async/await)

JS

- Déclarer une fonction asynchrone
- Attendre le résultat d'une fonction asynchrone
- Récupérer une erreur
- Clause terminale

```
async _execute () {  
  // traitement asynchrone: accès réseau, disque,  
  // processus...  
}
```

```
async run (params = {}) {  
  let failure = null  
  this.success = null  
  this.start = Date.now()  
  try {  
    const results = await this._execute(params)  
    return this.next  
      ? await this.next.run({ ...params, ...results })  
      : results  
  } catch (err) {  
    failure = err  
    throw err  
  } finally {  
    this.end = Date.now()  
    this.success = failure === null  
  }  
}
```


Modules - 1/2

JS

- Un module peut exporter un symbole par défaut (fichier *task.js*)

```
export default class Task {  
  // ...  
}
```

Modules - 1/2

JS

- Un module peut exporter un symbole par défaut (fichier *task.js*)

```
export default class Task {  
  // ...  
}
```

- Un module peut importer des symboles externes (fichier *parallel.js*)

```
import Task from './task'  
  
export default class Parallel extends Task {  
  // ...  
}
```

Modules - 1/2

JS

- Un module peut exporter un symbole par défaut (fichier *task.js*)

```
export default class Task {  
  // ...  
}
```

- Un module peut importer des symboles externes (fichier *parallel.js*)

```
import Task from './task'  
  
export default class Parallel extends Task {  
  // ...  
}
```

- Chaque fichier (module) est un bloc (scope)

Héritage de classe

JS

- Héritage simple

```
import Task from './task'  
  
export default class Parallel extends Task {
```

Héritage de classe

JS

- Héritage simple

```
import Task from './task'  
  
export default class Parallel extends Task {
```

- Appel du constructeur hérité (avant d'utiliser `this`)

```
  constructor({field, tasks = []} = {}, next = null) {  
    super('', next)  
    Object.assign(this, { tasks, field })  
  }
```

Héritage de classe

JS

- Héritage simple

```
import Task from './task'

export default class Parallel extends Task {
```

- Appel du constructeur hérité (avant d'utiliser `this`)

```
  constructor({field, tasks = []} = {}, next = null) {
    super('', next)
    Object.assign(this, { tasks, field })
  }
```


- Appel des méthodes héritée avec `super`

```
  toString() {
    const sub = this.tasks.map(t => `${Task.display(t)}`).join(' | ')
    return `${super.toString()} (${sub})`
  }
}
```


Destructuration

JS

- Pendant une affectation ou dans une signature



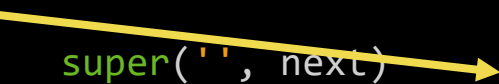
```
constructor(  
  {field, tasks = []} = {},  
  next = null  
) {  
  
  super('', next)  
  Object.assign(this, {tasks, field})  
}
```

Destructuration

JS

- Pendant une affectation ou dans une signature
- Extrait des variables d'un tableau/objet

```
constructor(  
  {field, tasks = []} = {},  
  next = null  
) {  
  
  super('', next)  
  Object.assign(this, {tasks, field})  
}
```

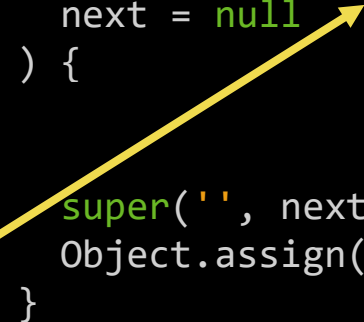


Destructuration

JS

- Pendant une affectation ou dans une signature
- Extrait des variables d'un tableau/objet
- Utilisable avec des valeurs par défaut

```
constructor(  
  {field, tasks = []} = {},  
  next = null  
) {  
  super('', next)  
  Object.assign(this, {tasks, field})  
}
```



Destructuration

JS

- Pendant une affectation ou dans une signature
- Extrait des variables d'un tableau/objet
- Utilisable avec des valeurs par défaut


```
constructor(  
  p1,  
  next = null  
) {  
  const field = p1.field  
  const tasks = p1.tasks || []  
  super('', next)  
  Object.assign(this, {tasks, field})  
}
```

Destructuration

JS

- Pendant une affectation ou dans une signature
- Extrait des variables d'un tableau/objet
- Utilisable avec des valeurs par défaut
- Permet de renommer

```
constructor(  
  {field, tasks = []} = {},  
  next = null  
) {  
  
  super('', next)  
  Object.assign(this, {tasks, field})  
}
```



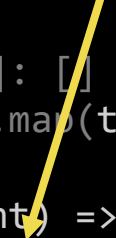
```
const { data: [{ fact, points: score }] = [] } = await new Parallel(  
  { tasks: crawlers, field: 'data' },  
  sorter  
)  
.run()  
console.log(`best fact found (${score} pts) :\n\n${fact}`)
```

Lambdas (Arrow functions)

JS

- Fonction anonyme qui conserve le `this` appelant

```
async _execute (params) {  
  if (!this.tasks.length) return { [this.field]: [] }  
  const results = await Promise.all(this.tasks.map(task => task.run(params)))  
  return {  
    [this.field]: results.reduce((total, current) => {  
      const currentData = current && current[this.field]  
      return total.concat(currentData)  
    }, [])  
  }  
}
```



Lambdas (Arrow functions)

JS

- Fonction anonyme qui conserve le `this` appelant

```
async _execute (params) {  
  if (!this.tasks.length) return { [this.field]: [] }  
  const results = await Promise.all(this.tasks.map(task => task.run(params)))  
  return {  
    [this.field]: results.reduce((total, current) => {  
      const currentData = current && current[this.field]  
      return total.concat(currentData)  
    }, [])  
  }  
}
```

- Syntaxe avec un paramètre et une instruction

```
toString() {  
  const sub = this.tasks.map(t => `${Task.display(t)}`)  
  ).join(' | ')  
  return super.toString() + ' (' + sub + ')'  
}
```

Modules - 2/2

JS

- Export de symboles nommés (*jobs/chuck/utils.js*)

```
const maxQuotes = 99
export default maxQuotes

export function decode(text) { /* ... */ }
```

Modules - 2/2

JS

- Export de symboles nommés (*jobs/chuck/utils.js*)

```
const maxQuotes = 99
export default maxQuotes

export function decode(text) { /* ... */ }
```

- Import de tous les symboles (*job/chuck/index.js*)

```
import nbQuotes, * as utils from './utils'

// utils.decode(best.fact)
```


Modules - 2/2

JS

- Export de symboles nommés (*jobs/chuck/utils.js*)

```
const maxQuotes = 99
export default maxQuotes

export function decode(text) { /* ... */ }
```

- Import de tous les symboles (*job/chuck/index.js*)

```
import nbQuotes, * as utils from './utils'

// utils.decode(best.fact)
```

- Import sélectif et renommage

```
import { decode as decodeHTML } from './utils'
```

Modules - 2/2

JS

- Export de symboles nommés (*jobs/chuck/utils.js*)

```
const maxQuotes = 99
export default maxQuotes

export function decode(text) { /* ... */ }
```

- Import de tous les symboles (*job/chuck/index.js*)

```
import nbQuotes, * as utils from './utils'

// utils.decode(best.fact)
```

- Import sélectif et renommage

```
import { decode as decodeHTML } from './utils'
```


- Import dynamiques

```
const { default: run } = await import(`../lib/jobs/${args}`)
```

Opérateurs *rest* et *spread*

JS

- Dans une signature, *rest* collecte tous les paramètres restants dans un tableau




```
export class Crawler extends Task {  
  constructor (options, ...args) {  
    const { page } = options  
    args.unshift(`page ${page}`)  
    super(...args)  
    Object.assign(this, options)  
  }  
}
```


Opérateurs *rest* et *spread*

JS

- Dans une signature, *rest* collecte tous les paramètres restants dans un tableau
- Lors d'un appel, *spread* transforme un tableau en une suite de paramètres

```
export class Crawler extends Task {  
  constructor (options, ...args) {  
    const { page } = options  
    args.unshift(`page ${page}`)  
    super(...args)  
    Object.assign(this, options)  
  }  
}
```




Opérateurs *rest* et *spread*

JS

- Dans une signature, *rest* collecte tous les paramètres restants dans un tableau
- Lors d'un appel, *spread* transforme un tableau en une suite de paramètres
- Dans un objet, collecte ou dilate les propriétés

```
export class Crawler extends Task {  
  
  constructor (options, ...args) {  
    const { page } = options  
    args.unshift(`page ${page}`)  
    super(...args)  
    Object.assign(this, options)  
  }  
}
```

```
async run (params = {}) {  
  // ...  
  const results = await this._execute(params)  
  return this.next  
    ? await this.next.run({ ...params, ...results })  
    : results  
  // ...  
}
```



Opérateurs *rest* et *spread*

JS

- Dans une signature, *rest* collecte tous les paramètres restants dans un tableau
- Lors d'un appel, *spread* transforme un tableau en une suite de paramètres
- Dans un objet, collecte ou dilate les propriétés

```
export class Crawler extends Task {  
  
  constructor (options, ...args) {  
    const { page } = options  
    args.unshift(`page ${page}`)  
    super(...args)  
    Object.assign(this, options)  
  }  
}
```

```
async run (params = {}) {  
  // ...  
  const results = await this._execute(params)  
  return this.next  
    ? await this.next.run({ ...params, ...results })  
    : results  
  // ...  
}
```


JS

ESNext
Time to get *Chucked!*



JS

ESNext

L'écosystème



Les dépendances

JS

- Une application JS peut dépendre de bibliothèques

Dynamiques

- Déclaration : page web
`<script>`
- Importation explicite :
variable globale
- Récupérées par le navigateur
- Hébergées sur un CDN:
CloudFlare, Akamai, Ovh...

Statiques

- Déclaration : `package.json`
- Importation explicite :
`import`, `require()`
- Récupérées en ligne de commande: `npm-cli`, `yarn`
- Hébergée en ligne :
<https://registry.npmjs.org>,
Nexus, Synopia...

Gérer ses dépendances statiques

JS

- Descripteur minimaliste

```
{  
  "name": "change-mind-about-js",  
  "version": "2.0.0",  
  "description": "A conference about modern JavaScript code",  
  "dependencies": {  
    "esm": "^3.0.38",  
    "html-entities": "^1.2.1",  
    "request": "^2.87.0"  
  }  
}
```

- **dependencies** : nécessaire à l'exécution (nom: version)
- **devDependencies** : pour le développement (test, lint...)
- Récupération des dépendances : **npm install** ou **yarn**
- Téléchargées dans le dossier **node_modules/**

JS

ESNext

Les outils pour développer



Transpilation et packaging

JS

- Portabilité et efficacité du code

Transpiler

- Toutes les plateformes n'implémentent pas l'ensemble de la spec
- ES6, ES7, TypeScript → ES5 avec des polyfills
- Utilisation de sourcemaps pour débbugger
- [Babel](#), [TypeScript](#)...

Packager

- Rassembler les fichiers source et les dépendances en quelques bundles (Web)
- Intégrer les ressources externes : images, css (Web)
- Supprimer le code inutilisé
- Offusquer le code
- [Webpack](#), [Rollup](#)...

Assistance au développement

JS

- Qualité/simplicité du code et homogénéité du style

Lint

- [Erreurs possibles](#) : variable inutilisée (typos), getter sans valeur de retour...
- [Bonnes pratiques](#) : égalité stricte, case sans break...
- [Style](#) : point-virgule, indentation...
- Avertissements ou erreurs, extensible avec des règles perso

Formater

- Indentation, espacement, retour à la ligne
- Conçu pour la collaboration : stopper les débats cosmétiques
- Guide style subjectif, à dessein ([Prettier](#), [Standard](#))

JS

ESNext

Pour conclure



JS everywhere

JS

- Un langage devenu incontournable :
 - Non pas car il est le plus expressif, rapide, solide...
- Il est porté par un écosystème vibrant
- Spécification, implémentation...
tout es libre
- Il est présent sur tous les canaux :
 - Le langage natif du Web
 - Apps Mobiles : React Native, Cordova
 - Apps Desktop : Electron, NW.js
 - Server : Node.js
 - IoT : Johnny-5 tourne sur Tessel, Arduino, Raspberry...



JS everywhere

JS

- Un langage dev...
 - Non pas car il... de, solide...
- Il est porté par...
- Spécification, im...
tout es libre
- Il est présent su...
 - Le langage na...
 - Apps Mobiles...
 - Apps Desktop...
 - Server : [Node](#)...
 - IoT : [Johnny-5](#) t... , Raspberry...



JS

Merci pour votre attention et...





Merci pour votre attention et...

...amusez vous bien avec JavaScript !



Crédits photos



- Code disponible sur github:
<http://github.com/feugy/change-mind-about-js>
- Fond « speaker's grid » par Thomas Wang
- Slide 3 « code review » par Mickael Zuskin
- Slide 27 « Hard Rock lives on » par Dustin Gaffke
- Les logos utilisés sont la propriété exclusive de leur propriétaire