Principes et Commandes NodeJS - TypeScript

NodeJS

Principe :

NodeJS permet d'utiliser du JavaScript en dehors d'un navigateur web.

Comme Chrome, il se base sur V8 pour interpréter et optimiser le code JS.

NPM = Node Package Manager. C'est le gestionnaire de projet JavaScript (équiv de ***Maven*** et ***Gradle*** pour Java)

Instalation de NodeJS :

* Via Nodist :

GitHub de Nodist : https://github.com/nullivex/nodist

Nodist permet d'instaler plusieurs versions de Node.JS sur une même machine et de sélectionner la version à utiliser.

Pour instaler Nodist et NodeJS : https://github.com/nullivex/nodist/releases

> télécharger et exécuter .exe

> "node -v" et "npm -v" en ligne de commande pour vérifier que cela a bien fonctionné et que la variable Path a été MàJ

* Ou instalation simple :

Télécharger NodeJS sur <https://nodejs.org/en/download/>

L'installer à la racine du lecteur C:

Vérifier que la variable d'environnement pointant sur le dossier 'npm' a bien été créée. Sinon la rajouter au Path.

Instalation de NPM et Angular-CLI

Dans l'invite de commande : (Rappel pour l'ouvrir : Window+R => cmd)

Installer la dernière version de NPM (*Node Package Manager*) en global avec la commande : npm install -g npm@latest

Installer le CLI (*Command Line Interface*) Angular en global avec la commande : npm install -g @angular/cli

Contenu du projet :

package.json

fichier de config (équiv. à *POM.xml* de Maven ou *build.gradle* de gradle) qui définit :

- nom (obligatoire)

- version (obligatoire)

- dépendances

- raccourcit en ligne de commandes ("script": {})

dossier ***node\_modules***

contient toutes les dépendances liées au projet et chargé par npm

ATTENTION : à ne pas charger sur **GitHub**

Détails importants sur le fonctionnement global et principe d'importation/exportation d'éléments d'un fichier .js à un autre :

Lors du chargement d'un fichier .js, Node place le contenu dans une fonction afin de préserver le global space name et de mettre à disposition 5 éléments accessibles partout dans l'appli de la manière suivante :

(function (exports, require, module, \_\_filename, \_\_dirname){

*contenu du fichier.js original*

});

exports => ref à module.exports

require => permet de charger les modules (depuis un autre fichiers .js ou depuis une dépendances incluses dans ***node\_modules***)

module => ref au module courant pour mettre à disposition l'élément module.export qui permet d'exporter des objets du fichier courant

\_\_filename => ref au fichier .js lui-même

\_\_dirname => ref au dossier courant lui-même

Ainsi, il est possible d'accéder aux éléments comme suit :

var monObjet = require('monmodule') importe le module contenu dans le dossier de dépendances node\_modules

var monObjet = require('./monFichier.js') importe les éléments déclarés module.exports = ... dans le fichier dans le même dossier

var monObjet = require('../monFichier.js') importe les éléments déclarés module.exports = ... dans le fichier dans le dossier parent

fichierA.js

module.exports = monObjet; => rend l'objet monObjet accessible (via un require('fichierA')) depuis un autre fichier JavaScript

fichierB.js

var monObjet = require('./fichierA'); => récupération de monObjet dans un autre fichier (ici dans le même dossier)

Dans la pratique, on aura le plus couramment :

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  import { MonService } from 'src/services/mon-service.ts';  export class MonComponent {  constructor(public monService: MonService) {}  maMethode() {  return monService.faitQuelqueChose();  }  } |

Tests unitaires

Mocha gère les tests unitaires (equiv de ***JUnit*** en Java)

Ajouter la dépendance mocha au projet : npm install mocha --save

Créer un dossier test

Les fichiers sont nomes : **monFichier.spec.js**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | | Utilisation | |
| NodeJS |  | | | | |
|  | en ligne de commande |  | |  |  |
| node | Lance l'application | node application.js | |  |  |
| npm init | initialise le projet | npm init -y | |  |  |
| npm install | télécharge la dépendance en local, cad dans le dossier node\_modules du projet | npm install monModule --save | |  | ATTENTION : doit être joué dans le dossier du projet lui-même ! |
|  | télécharge la dépendance en local en mode développeur | npm install monModule --save-dev | |  |  |
| -g | instaler la dépendance en global, cad dans Node et accessible par tous les projets | npm install -g monModule | |  |  |
|  | désinstalation de la dépendance globale | npm uninstall -g monModule | |  |  |
|  |  |  | |  |  |
|  | **TypeScript** |  | |  |  |
| typescript | instalation de TypeScript en global | npm install -g typescript | | tsc -v tsc --help |  |
| tsconfig.json | fichier qui définit les options de transpilation de TypeScript en JavaScript |  | |  |  |
|  |  |  | |  |  |
|  | Mocha = Tests unitaires |  | |  |  |
| mocha | lancement des tests unitaires  si dans package.json  "scripts": {  "test": "mocha" | npm node\_modules/.bin/mocha  *alors lancement des tests avec :*  mpn mocha | |  |  |
| chai | permet d'ajouter des fonctionnalités aux tests de base | npm install chai --save-dev | var expect = require('chai').expect;  describe('Math.addition', function() {  describe('#without arguments', function() {  it('should return 0 when have no arguments', function() {  expect(function() {  math.addition()}).to.not.throw();  expect(math.addition()).to.equal(0);  });  });  describe('#agrgument is not a number', function() {  it('should ignore argument that is not a number', function() {  expect(function() {  math.addition(2, 4, 'notNumber')}).to.not.throw();  expect(math.addition(2, 4, 'notNumber')).to.equal(6);  });  });  }); | |  |
|  |  |  | |  |  |

TypeScript

Principe :

Langage entre le langage type (comme Java) et le JavaScript.

Le TypeScript est d'abord transpilé en JavaScript avant d'être chargé dans le navigateur ou le serveur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation | |
|  |  | | | |
| Variables |  |  |  |  |
| let | variable remplaçant var | let monObjet;  let monObjet: MonObjet;  let monObjet: MonObjet = new MonObjet();  for(let i = 0; i < 10; i++) {  *instructions*  } | let monObjet: any;  let monNombre: number;  let maString: string;  let maListe: Array<any> = [ ];  let maListe: Array<any> = new Array<any>(); |  |
| const | constante, objet non mutable cad référence mémoire qui ne change pas, mais peut ajouter/suppr les propriétés dynamique comme avec une var | const monObjet: MonObjet = new MonObjet();  const maListe: Array<MonObjet> = [ ]; |  |  |
|  |  |  |  |  |
| String |  |  |  |  |
| ` `  ` ${ } ` | déclaration de chaine de caractère simplifiant la concaténation via les EL | let nb1 = 1, nb2 = 2;  let msg = 'Valeur de';  calculerSomme() {  return `${msg} la somme = ${nb1 + nb2}`;  } |  |  |
|  |  |  |  |  |
| Tuple |  |  |  |  |
| [ , , ] | permet de définir les types d'un nombre défini d'éléments dans un tableau | cont monTuple = [string, number, boolean];  monTuple[1] = 3;  monTuple[0] = 'Ma chaine'; | monTuple[1] = 'Ma chaine' *// => erreur* |  |
|  |  |  |  |  |
| Tableau |  |  |  |  |
| … | tableau contenant plusieurs paramètres | maFunction(arg1, …args) {  maFunction(arg1: string, …args: Array<string>) {  function f (x, y, ...arr) {  return (x + y) \* arr.length;  } | *appel à la fonction avec plusieurs paramètres :*  const valeur = f(1, 2, 3, 4, 5); *// => 6*  *appel à la fonction avec un tableau :*  const monTableau = ['a', 'b', 'c'];  const valeur = f(1, 2, 3, …monTableau); *// => 12* |  |
|  | clone les items d'un tableau dans un autre tableau | const params = ['hello', true, 7];  const other = [1, 2, ...params]; *// => other = [1, 2, 'hello', true, 7]* |  |  |
|  | clone les propriété d'un objet dans un autre objet | |  |  | | --- | --- | | var monObjetA = {  prop1: val1,  prop2: val2  } | var monObjetB = {  ...monObjetA,  prop3: val3  } | |  |  |
|  |  |  |  |  |
| Fonction |  |  |  |  |
|  | paramètres de fonctions avec des valeurs par défauts si non précisées lors de l'appel de la fct | function f (x, y = 2, z = 3) {  return x + y + z;  } |  |  |
| ? | paramètre facultatif | function f (x, y?, z?) {  const result = x;  if(y) { result += y);  if(z) { result += z);  return result;  } |  | ATTENTION : les paramètres facultatifs doivent obligatoirement être placés en dernier dans la liste des paramètres ! |
| Arrow function | (= lambda en Java) | Remarque : fonction anonyme mais ne créant pas de nouveau scope !!! | |  |
| => | 1 argument - retourne 1 valeur  plusieurs arguments - retourne 1 valeur  1 argument - retourne 1 objet  1 argument - réalise plusieurs instructions | (e => e + 1)  ((e, f) => e + f)  (e => ({prop1: val1, prop2: val2}))  (e => {  instruction(s);  return e;  }) |  |  |
| Objet |  |  |  |  |
|  | simplification d'initialisation de variable | var x = 1, y = 2;   |  |  |  | | --- | --- | --- | | monObjet = {  x,  y  } | ⇔ | monObjet = {  x = x,  y = y  } | |  |  |
|  | nom de propriété dynamique | function monCalcul(x, y) {  return x + y;  }  let monObjet = {  prop1: val1,  ['prop' + monCalcul(1, 1)]: val2  } |  |  |
|  | définition de paramètre implicitement | monObjet = {  constructor(public maProp1 : string, private maProp2 : number = 2) ; ⇔  } | monObjet = {  maProp1 : string ;  maProp2 : number = 2 ;  constructor(valeur1 : string, valeur2 : number) {  this.maProp1 = valeur1 ;  if(valeur2) {  this.maProp2 = valeur2 ;  }  }  } |  |
|  | définition de méthode implicitement | monObjet = {  maFonction(x, y) { ⇔  instruction(s);  } | monObjet = {  maFonction: function(x, y) {  instruction(s);  } |  |
|  | constructeur d'objet et héritage | class MonObjet {  constructor (id, x, y) {  this.id = id;  this.move(x, y);  }  move (x, y) {  this.x = x;  this.y = y;  }  } | class MonObjetB extends MonObjet {  constructor (id, x, y, z) {  super(id, x, y);  this.z = z;  }  } |  |
| get  set | définir des getteur et setter manuellement | class MonObjet {  private \_maProp1: string;  get maProp1(): string {  return this.\_maProp1;  }  set maProp1(value: string) {  if(value) {  this.\_maProp1 = value;  }  }  } | *utilisation des getters et setters* ***sans get et set*** *:*  monObjet.maProp1 = 'Ma chaine';  console.log(monObjet.maProp1); |  |
|  |  |  |  |  |
| interface |  |  |  |  |
|  | le nom de la prop sera défini lors de la création de l'objet | interface MonInterface {  maProp1: MonObjet;  maProp2: string;  [maPropDeNomInconnu: string]: any;  maPropOptionnelle1?: number;  maFonction(monParam1: string): void;  } |  |  |
| readOnly | équivalent de const pour les propriétés des objets | interface MonInterface {  maProp1: MonObjet;  readonly maProp2: string; |  |  |
|  |  |  |  |  |
| Fonction Générator |  | voir => https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/function\* | |  |
| fonction\* | fonctionne comme le mode debug | function\* monGenerator(i) {  *instruction(s);*  yield i + 1;  *instruction(s);*  yield i + 10;  return result;  }  var gen = monGenerator(10); *=> retourne 11*  gen.next(); *=> retourne 20*  gen.next(); *=> retourne result* |  |  |
|  |  |  |  |  |
| Destructuring | simplifie la syntaxe du code |  |  | Rq: à utiliser avec parcimonie car rend le code moins lisible et plus complexe à comprendre |
| const { } = | ne récupère que les valeurs des props qui m'intéressent de l'objet | const {prop1, prop5 = 'valDefaut', prop6} = maFctQuiRetourneUnObj(); |  | Rq: si prop5 n'est pas trouvée dans l'objet, elle prendra la valeur 'valDefaut' par défaut |
| const { prop: variable | récupère les valeurs et les place dans des constantes pouvant être utilisées directement | const {prop1: a, prop5: b, prop6 } = maFctQuiRetourneUnObj();  if(a > 5) {  ...  } |  |  |
|  | utilisation des propriétés d'un objet ou des éléments d'un tableau dirrectement | function({obj1, obj2}) {  console.log(obj1, obj2);  } | function([obj1, obj2]) {  console.log(obj1, obj2);  } |  |
|  |  |  |  |  |
| Window |  |  |  |  |
|  | changer le titre de l'onglet | window.parent.document.title = 'Nouveau titre'; |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **TypeScript** |  |  |  |
| Codes utiles |  |  |  |  |
|  |  |  |  |  |
| Fonctions outils | plusieurs fonctions outils pratiques | /\*\*  \* Fonction utilitaire determinant si l'objet passé en paramètre est null ou undefined.  \* @param value objet de n'importe quel type  \* @return true si l'objet est null ou undefined, false sinon.  \*/  export function isNullOrUndefined(value: any): boolean {  return value === null || value === undefined;  }  /\*\*  \* Fonction utilitaire déterminant si au moins un des objets passés en paramètre est null ou undefined.  \* @param values nombre variable d'objet de n'importe quel type  \* @return true si au moins un des objets est null ou undefined, false sinon.  \*/  export function anyIsNullOrUndefined(...values): boolean {  return values.some(value => isNullOrUndefined(value));  }  /\*\*  \* Fonction utilitaire déterminant si au moins un des tableaux/liste passés en paramètre est null ou undefined ou vide.  \* @param values nombre variable de tableaux de n'importe quel type  \* @return true si au moins un des tableaux est null ou undefined ou ne contenant aucun élément, false sinon.  \*/  export function anyIsNullOrEmpty(...tables: Array<Array<any>>): boolean {  return tables.some(table => isNullOrUndefined(table) || table.length === 0);  }  /\*\*  \* Fonction utilitaire déterminant si au moins une des chaines passées en paramètre est null ou undefined ou est une chaine vide.  \* @param values nombre variable de string  \* @return true si au moins une chaines est null ou undefined ou est une chaine vide, false sinon.  \*/  export function anyStringIsNullOrEmpty(...values: Array<string>): boolean {  return values.some(value => isNullOrUndefined(value) || value.trim().length === 0);  } | | |
| StringFormat | remplace des tags par des valeurs dans une chaine de caractère à la manière de String.format( ) en Java | /\*\*  \* stringFormat  \* Fonction utilitaire permettant de remplacer des tags dans une chaine de caractère par des valeurs  \* Format des tags : {0}, {1}, {2}, ...  \*  \* Exemple d'utilisation :  \* let maChaine = stringFormat('Ceci est un {0} d\'utilisation de la {1} utilitaire formatString avec {2} valeurs remplacées dans la chaine', 'exemple', 'fonction', 3);  \* console.log(maChaine);  \*  \* Résultat dans la console :  \* Ceci est un exemple d'utilisation de la fonction utilitaire formatString avec 3 valeurs remplacées dans la chaine  \*  \* @param formatString chaine de caractère contenant les tags  \* @param replacements valeurs à insérer dans la chaine  \*/  export function stringFormat(formatString: string, ...replacements: any[]): string {  replacements.forEach((replacement, index) => {  formatString = formatString.replace(`{${index}}`, replacement);  });  return formatString;  } | | |
| Map | convertir une map en un objet  convertir un objet JSON en une map | /\*\*  \* convertMapToObject  \* Fonction utilitaire convertissant une map de type <string, any> en un objet JavaScript  \*  \* @param map de type <string, any>  \* @returns {any} objet ayant pour propriété les clé de la map  \*/  export function convertMapToObject(map: Map<string, any>): any {  let object = {};  map.forEach((value, key) => object[key] = value);  return object;  }  /\*\*  \* convertJsonObjectToMap  \* Fonction utilitaire convertissant un jsonObjet en Map<string, any>  \*  \* @param jsonObject  \* @returns {Map<string, any>} ayant pour clé les propriété du jsonObjet  \*/  export function convertJsonObjectToMap(jsonObject: JSON): Map<string, any> {  let map: Map<string, any> = new Map<string, any>();  for (let key in jsonObject) {  map.set(key, jsonObject[key]);  }  return map;  } | | |
|  |  |  | | |
| Deserializable | désérialisation d'un JSON en un objet incluant les méthodes de la classe du même type | *-------------------------------------- interface Deserializable*  interface Deserializable {  getTypes(): Object;  }  */\* Interface servant à marquer les Beans pour la deserialisation avec la fonction deserialize(jsonObject, MonObjet)*  *Exemple d'utilisation :*  *export class CoordonneesDestinataire implements Deserializable {*  *destinataire: Destinataire;*  *adressesPostales: Array<AdressePostale>;*  *numTelelphone: number;*  *adressesCourriels: Array<string>;*  *coordonneeVerifiee: boolean;*  *public getTypes() {*  *return {*  *destinataire: Destinataire,*  *adressesPostales: Array,*  *adressesPostalesArray: AdressePostale,*  *adressesCourriels: Array*  *}*  *}*  *}*  *Dans le retour de getType() :*  *- seules les prorpiétés de type 'object' doivent être indiquées, pas les types primitifs*  *- toutes les propriétés de type Array doivent être indiquées*  *- mais seuls les Array contenant un type 'object' doivent EN PLUS être indiqués avec le format : 'nomProprieteArray: TypeContenuTableau'*  *Remarques :*  *~ les Beans doivent pouvoir être instanciés à vide (constructeur sans paramètre ou uniquement avec paramètres facultatifs !)*  *~ toutes les propriétés de type 'object' doivent implémenter Deserializable (dans l'exemple, Destinataire et AdressePostale doivent implémenter Deserializable)*  *Si un Bean ne contient que des types natifs, getType() retourne alors un objet vide.*  *export class AdressePostale implements Deserializable {*  *ligne1: string;*  *ligne2: string;*  *codePostal: number;*  *ville: string;*  *pays: string;*  *public getTypes() {*  *return {}*  *}*  *}*  *Exemple de déserialisation :*  *obtenirCoordonneesDestinataire(idDestinataire: string): Promise<CoordonneesDestinataire> {*  *return new Promise<CoordonneesDestinataire>((resolve, reject) => {*  *this.diffusionAdapter.getCoordDesinataire(idDestinataire).subscribe((response) => {*  *//déserialisation d'un objet*  *const obj: CoordonneesDestinataire = deserialize(response, CoordonneesDestinataire);*  *resolve(obj);*  *}, (error) => {*  *reject(error);*  *});*  *});*  *}*  *//ou déserialisation d'un tableau d'objet*  *const obj: Array<Document> = deserializeArray(response, Document);*  *\*/*  *----------------------------- méthodes utilitaires de deserialisation utilisant l'interface Deserializable :*  export function deserialize(jsonObject, clazz) {  const instance = new clazz(),  types = instance.getTypes();  for (let prop in jsonObject) {  if (isNullOrUndefined(jsonObject[prop])) {  continue;  }  if (typeof jsonObject[prop] !== 'object') {  instance[prop] = jsonObject[prop];  } else {  if (jsonObject[prop] instanceof Array) {  instance[prop] = deserializeArray(jsonObject[prop], types[prop.concat('Array')]);  } else {  instance[prop] = deserialize(jsonObject[prop], types[prop]);  }  }  }  return instance;  }  export function deserializeArray(jsonArray, arrayContentClazz) {  let table = new Array();  if (!isNullOrUndefined(jsonArray)) {  //s'il s'agit d'un tableau de primitifs, on copie simplement le tableau avec son contenu  if (isNullOrUndefined(arrayContentClazz)) {  table = jsonArray;  } else {  //pour un tableau d'objet, on déserialise chaque objet avant de l'ajouter au tableau  jsonArray.forEach(jsonObject => {  table.push(deserialize(jsonObject, arrayContentClazz));  });  }  }  return table;  } | | |
| ValidableBean |  | *-------------------------------------- interface ValidableBean*  interface ValidableBean {  isValid(): boolean;  }  *----------------------------- méthode utilitaire de validation des objets utilisant l'interface ValidableBean :*  /\*\*  \* Fonction utilitaire spécifique aux Beans implémentant l'interface ValidableBean.  \* Elle vérifie que tous les éléments passés en paramètres sont ni null ni undefined avant d'appeler la méthode isValid() pour chacun d'eux.  \* @param values nombre variable de bean de type ValidableBean  \* @return true si tous les beans sont valide selon leur propre méthode. False sinon.  \*/  export function areAllValid(...beans: Array<ValidableBean>) {  if (anyIsNullOrUndefined(…beans)) {  return false;  }  return beans.every(bean => bean.isValid());  } | | |
|  |  |  |  |  |