

M1IFo3

Conception d'applications Web



RAPPELS JAVASCRIPT

LIONEL MÉDINI
NOVEMBRE 2020

Plan du cours



- **Introduction**
 - Rich Internet Applications (RIA)
 - Historique
 - JavaScript aujourd'hui
- **Rappels JavaScript**
 - Programmation fonctionnelle
 - Programmation orientée-objet
 - Programmation événementielle
- **Programmation XML en JavaScript**
 - Retour sur l'API DOM (Document Object Model)
 - Utilisation du DOM XML en JavaScript
- **Références**

Introduction



- Objectif : concevoir des applications Web « riches »
 - Web-based
 - ✦ Paradigme client-serveur, HTTP
 - ➔ Programmation côté serveur et côté client
 - Expérience utilisateur proche des applications natives
 - ✦ Interface utilisateur fluide, ergonomique, dynamique
 - ➔ Traitement de l'interface côté client (JavaScript, CSS , DOM)
 - ➔ Échanges client-serveur asynchrones (AJAX)
 - Logique métier complexe
 - ✦ Outils « évolués » de modélisation, conception, développement
 - ➔ IDE, POO, UML, design patterns, méthodes agiles, XP...
- ➔ Où placer la logique métier ? La couche données ?

Introduction



- L'offre côté client (rappel)
 - Moteur de rendu : *cf.* cours 1
 - Moteur de scripting
 - ✦ Objectif : dynamiser les applications Web
 - ✦ Propriétés
 - Manipulation de la page
 - Échanges de données asynchrones
 - Logique applicative côté client
 - Mécanismes de bas niveau
 - ✦ Premières versions :
 - JavaScript : Netscape Navigator 2.0 (mars 1996)
 - JScript : MS Internet Explorer 3.0 (août 1996)
 - Standard commun : ECMAScript-262 (juin 1997)
 - Officiellement supporté par JavaScript, JScript et ActionScript

Introduction



- **ECMAScript-262 : historique des versions**
 - 1st Edition (06/1997)
 - Edition 2 (06/1998)
 - Edition 3 (12/1999)
 - Edition 4 : abandonnée (2008)
 - Edition 5 (12/2009), puis 5.1 (06/2011) → JavaScript 1.8
 - ECMAScript Harmony = ES 2015 = ES6 (06/2015) → JavaScript 2.0
 - ES7 (06/2016)
 - ES.Next... ES10
- **TypeScript**
 - Développé par MicroSoft en 2012
 - Utilisé dans plusieurs bibliothèques et frameworks
- **Nouvelles fonctionnalités à chaque édition**

Source :

<https://en.wikipedia.org/wiki/ECMAScript>

Introduction



- Moteurs ECMAScript / JavaScript / JScript

- 1995 : SpiderMonkey/TraceMonkey/JägerMonkey/IonMonkey (Gecko) : FireFox
 - ↳ 2017 : [Quantum](#) : Servo
- 2000 : KJS (KHTML) : Konqueror
 - ↳ 2002 : WebCore + JavaScriptCore (Webkit) : Chrome, Safari
 - ↳ 2008 : SquirrelFish Extreme (= Nitro) : Chrome, Safari
 - ↳ 2013 : Blink : Chrom[e|ium], Opera
- 2003 -> 2013 : Linear B/Futhark/Carakan (Presto) : Opera
- 1996 -> 2015 : JScript/Chakra (Trident) : Internet explorer
 - ↳ 2015 : Chakra (JavaScript) : MicroSoft Edge

Introduction



- **Caractéristiques des moteurs actuels (côté client)**
 - Interprétés
 - ✦ Compilation JIT en bytecode
 - Implémentent un sous-ensemble de ES6
- **JavaScript côté serveur**
 - Moteur V8 (Google)
 - Implémentation : NodeJS
- **Nouvelles versions (ES6, ES7, TS)**
 - Objectif
 - ✦ Permettre la programmation d'applications structurées côté client
 - Nouveaux éléments syntaxiques
 - Transpilables en ES5

Introduction



- ECMAScript 6
 - Classes & modules
 - Iterators & for/of loops
 - Python-style generators & generator expressions
 - Arrow functions
 - Binary data
 - Typed arrays
 - Collections (maps, sets and weak maps)
 - Promises
 - Number & math enhancements
 - Reflection
 - Proxies (metaprogramming for virtual objects and wrappers)
- ECMAScript 7 (puis 8, 9, 10)
 - Exponentiation operator (**)
 - Array.prototype.includes

Source : <https://en.wikipedia.org/wiki/ECMAScript>

Introduction



- TypeScript
 - Superset de ES6
 - Orienté-objet
 - Typage statique
 - Syntaxe spécifique
 - Fonctionnalités
 - ✦ Type annotations and compile-time type checking
 - ✦ Type inference
 - ✦ Type erasure
 - ✦ Interfaces
 - ✦ Enumerated type
 - ✦ Mixin
 - ✦ Generic
 - ✦ Namespaces
 - ✦ Tuple
 - ✦ Await

Source : <https://en.wikipedia.org/wiki/TypeScript>

Introduction



- Fonctionnalités en lien avec la spécification HTML5
- Philosophie
 - Rapprocher les fonctionnements des navigateurs de ceux des OS
- Exemples de fonctionnalités
 - Sélecteurs CSS : accès standardisé aux contenus de la page
 - Workers : threads
 - WebSockets : streaming, server push, connexion avec d'autres clients (P2P)
 - WebStorage : émulation BD pour stockage des données de session (sessionStorage) ou d'une application (localStorage)
 - GeoLocation
 - Device APIs...
- Implémentations variables selon les moteurs/navigateurs
- Utilisation simplifiée par de nombreuses bibliothèques
- plus de détails : <http://blog.xebia.fr/2010/03/18/html5-les-api-javascript/> ; <http://html5demos.com/>

Rappels JavaScript / ECMAScript



- **JavaScript : généralités**

- Conçu pour être « simple »

- ✦ Typage dynamique

- Déclarations : var

- ✦ Flexible & permissif

- Conversions de types implicites

- Utilisation de variables hors scope

- Pas/peu d'erreurs à l'exécution

- ➔ Résultats parfois surprenants

- <https://www.toptal.com/javascript/10-most-common-javascript-mistakes>

Rappels JavaScript / ECMAScript



- JavaScript : généralités

- ...Pour les applications complexes

- ✦ Déclarations : utiliser `const` et `let`
 - ✦ Gérer les erreurs avec `try` et `catch`
 - ✦ Utiliser les fonctionnalités des versions récentes
 - ✦ Utiliser le mode strict

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Strict_mode

Rappels JavaScript / ECMAScript



- Caractéristiques du langage
 - Fonctionnel
 - ✦ Fonctions = « citoyens de première classe »
 - Orienté prototype
 - ✦ « object-based » plutôt qu'« object-oriented »
 - ✦ Typage dynamique : types associés aux instances et non aux classes
 - Événementiel
 - ✦ Mécanismes de « callback »
 - ✦ Pattern observer : `addEventListener`
- Fonctionnalités
 - Reflection, E4X : ECMAScript for XML (ECMA-357), JSON...

Rappels JavaScript / ECMAScript



- **Déclarations**

```
const nombre = 1;  
  
let dictionnaire = {name: "foo", one: 1, two: 2};  
  
let tableau = [1, "toto", dictionnaire];  
  
const maFonction = function(x) { return x+1; };
```

Rappels JavaScript / ECMAScript



- **JavaScript Object Notation (JSON)**

- S rialisation/d s rialisation d'objets JS

- ✦ Dictionnaires

- ✦ Tableaux

- Primitives simples

- ✦ `JSON.parse(maVariableString);`

- ✦ `JSON.stringify (maVariableJS);`

Rappels JavaScript / ECMAScript



- Rappels de programmation fonctionnelle
 - S'appuie sur une pile d'appels imbriqués (contexte)
 - Fermeture (closure)
 - ✦ Permet de capturer l'environnement d'une fonction

```
function makeFunc() {  
  var name = "Mozilla";  
  function displayName() {  
    alert(name);  
  }  
  return displayName;  
}
```

```
var myFunc= makeFunc();  
myFunc();
```

Sources

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures>

<https://developer.mozilla.org/fr/docs/JavaScript/Guide/Closures>

Rappels JavaScript / ECMAScript



- Rappels de programmation fonctionnelle
 - S'appuie sur une pile d'appels imbriqués (contexte)
 - Function factory
 - ✦ Permet de passer des paramètres au moment de la création d'une fonction

```
function makeAdder(x) {  
  return function(y) {  
    return x + y;  
  };  
}
```

```
var add5 = makeAdder(5);  
var add10 = makeAdder(10);  
  
console.log(add5(2)); // 7  
console.log(add10(2)); // 12
```

Sources

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures>

<https://developer.mozilla.org/fr/docs/JavaScript/Guide/Closures>

Rappels JavaScript / ECMAScript



- **Programmation orientée-prototype (jusqu'à ES5)**
 - POO sans classe : on ne manipule que des objets
 - Objets représentés sous forme de dictionnaires (tableaux associatifs)
 - Propriétés
 - ✦ Pas de distinction entre les propriétés (attributs/méthodes) d'un objet
 - ✦ On peut remplacer le contenu des propriétés et en ajouter d'autres
 - Réutilisation des comportements (héritage)
 - ✦ se fait en clonant les objets existants, qui servent de prototypes
 - Sources
 - ✦ http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_prototype
 - ✦ http://en.wikipedia.org/wiki/Prototype-based_programming

Rappels JavaScript / ECMAScript



```
// Example of true prototypal inheritance style in JavaScript.
// "ex nihilo" object creation using the literal object notation {}.

var foo = {name: "foo", one: 1, two: 2};
var bar = {three: 3};

// For the sake of simplicity, let us pretend that the following line works
// regardless of the engine used:
// bar.__proto__ = foo;
// bar.[[ prototype ]] = foo
bar = Object.create( foo ); // JS 1.8.5 } // foo is now the prototype of bar.

// If we try to access foo's properties from bar from now on, we'll succeed.
bar.one // Resolves to 1.

// The child object's properties are also accessible.
bar.three // Resolves to 3.

// Own properties shadow prototype properties
bar.name = "bar";
foo.name // unaffected, resolves to "foo"
bar.name // Resolves to "bar"
```

Source : http://en.wikipedia.org/wiki/Prototype-based_programming

Rappels JavaScript / ECMAScript



- Comment programmer « proprement » de l'OO (ES5)
 - Plutôt « object-based » qu'« object-oriented »
 - ➔ Pour programmer en objet, il faut simuler des objets
 - ✦ Créer des constructeurs
 - ✦ Encapsuler les données (avec « `this` »)
 - ✦ Utiliser des « inner functions » à l'intérieur du constructeur
 - Exemple
 - <http://www.sitepoint.com/article/oriented-programming-1/>

Rappels JavaScript / ECMAScript



- Programmer de l'OO (ES6)

- Exemple

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.height * this.width;
  }
}
const square = new Rectangle(10, 10);
console.log(square.area); // 100
```

Rappels JavaScript / ECMAScript



- Programmer de l'OO (ES6)
 - Mots-clés « classiques »
 - ✦ class, constructor, static, this, extends, super
 - Les propriétés sont publiques
 - Pas de classes abstraites ou d'interfaces (pour l'instant)
 - voir TypeScript
 - [Liens vers les spécifications](#)

Rappels JavaScript / ECMAScript



- Fonctions de rappel (callback)

- Définition

- ✦ Fonction qui est passée en paramètre à une autre fonction afin que cette dernière puisse en faire usage

- Exemple : soient une fonction A et une fonction B

- ✦ Lors de l'appel de A, on lui passe en paramètre la fonction B : A(B)
 - ✦ Lorsque A s'exécutera, elle pourra exécuter la fonction B

- Intérêt : faire exécuter du code

- ✦ Sans savoir ce qu'il va faire (défini par un autre programmeur)
 - ✦ En suivant une interface de programmation qui définit
 - Le nombre et le type des paramètres en entrée
 - Le type de la valeur en sortie

- Source :

<http://www.eppershand.net/developpement/algorithmie/explication-utilite-fonctions-callback>

Rappels JavaScript / ECMAScript



- Fonctions de rappel (callback)

- La fonction qui reçoit une callback en paramètre doit respecter son interface

```
fonctionNormale(fonctionCallBack) {... fonctionCallback(argument); ...}
```

- 2 syntaxes pour le passage d'une fonction callback en argument d'une autre fonction

- ✦ Sans paramètre : directement le nom de la fonction

```
fonctionNormale(fonctionCallback);
```

- ✦ Avec paramètre : encapsulation dans une fonction anonyme

```
fonctionNormale(function() { fonctionCallback(arg1); });
```


Rappels JavaScript / ECMAScript



- Programmation événementielle

- ✧ L'objet Event

- Dénote un changement d'état de l'environnement
 - Peut être provoqué par l'utilisateur ou par l'application
- Peut être intercepté à l'aide de code JavaScript
- Possède un **flux d'événement** : propagation dans l'arbre DOM
 - Capture : du nœud Document au nœud visé par l'événement
 - Cible : sur le nœud visé
 - Bouillonnement (bubbling) : remontée jusqu'au nœud document
- Principales propriétés
 - **type** : type de l'événement ("click", "load", "mouseover"...)
 - **target** : élément cible (élément a pour un lien cliqué)
 - **stopPropagation** : arrête le flux d'un événement
 - **preventDefault** : empêche le comportement par défaut (navigation quand un lien est cliqué)

Source : <http://www.alsacreations.com/article/lire/578-La-gestion-des-evenements-en-JavaScript.html>

Rappels JavaScript / ECMAScript



- Programmation événementielle
 - ✧ Deux processus en parallèle
 - Principale : déroulement des traitements et association des événements à des fonctions de callback
 - Callbacks : récupèrent et traitent les événements
 - ✧ Deux syntaxes
 - DOM 0 : attributs HTML / propriétés JavaScript spécifiques onclick, onload...
(<http://www.w3.org/TR/html4/interact/scripts.html#h-18.2.3>)
 - DOM 2 : ajout d'eventListeners en JavaScript
monElement.addEventListener("click", maFonctionCallback, false);
 - Remarques :
 - Le troisième paramètre indique le type de propagation dans l'arbre DOM
 - Internet Explorer utilise la méthode attachEvent() au lieu de addEventListener()

Source : <http://www.alsacreations.com/article/lire/578-La-gestion-des-evenements-en-JavaScript.html>

Rappels JavaScript / ECMAScript



- **Programmation asynchrone**
 - Deux alternatives
 - ✦ Programmation événementielle : event handlers + callbacks
 - Voir slide précédent

Rappels JavaScript / ECMAScript



- Programmation asynchrone

- Deux alternatives

- ✦ Programmation événementielle : event handlers + callbacks
- ✦ Promesses

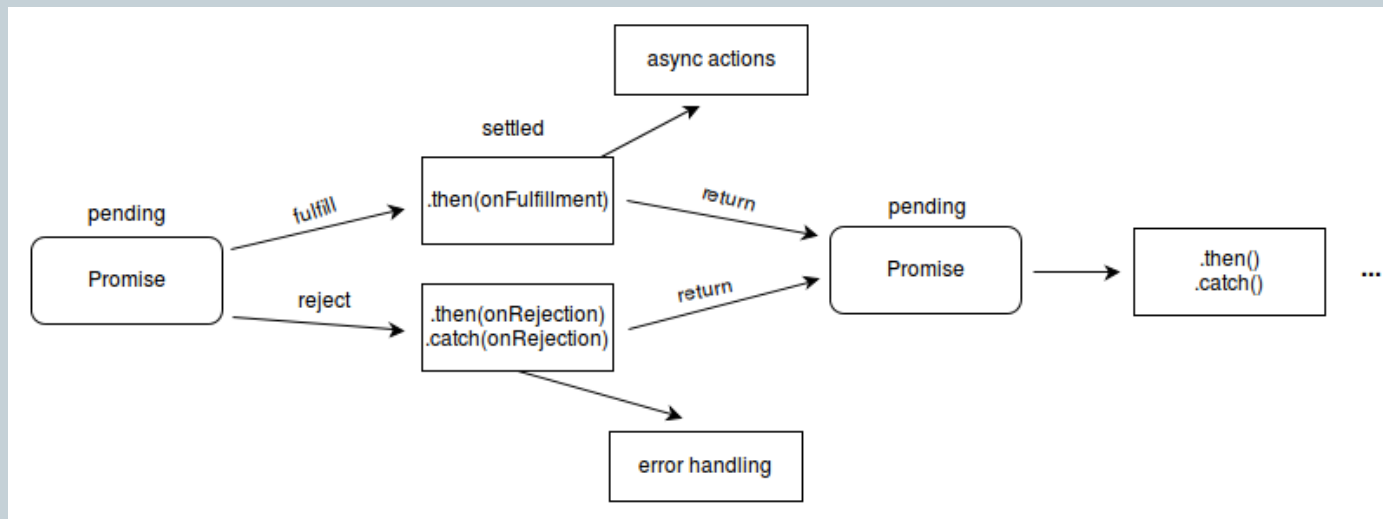
- Une promesse permet de :

- Lancer du code asynchrone ou différé (deferred)
 - requête AJAX, entrée utilisateur, calcul complexe...
- Réagir spécifiquement
 - en cas de succès
 - en cas d'erreur
 - à la fin de l'exécution
- En conservant des performances acceptables
 - multithread

Rappels JavaScript / ECMAScript



- Programmation asynchrone
 - Deux alternatives
 - ✦ Programmation événementielle : event handlers + callbacks
 - ✦ Promesses
 - Workflow (rappel M1Fo1)



Source : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Rappels JavaScript / ECMAScript



- **Programmation asynchrone**
 - Deux alternatives
 - ✦ Programmation événementielle : event handlers + callbacks
 - ✦ Promesses
 - Statuts : une promesse est dans l'un des états suivants
 - Pending : état initial
 - Fulfilled : l'opération s'est terminée correctement
 - Rejected : l'opération s'est terminée avec une erreur
 - Settled : l'opération s'est terminée, peu importe comment

Rappels JavaScript / ECMAScript



- **Programmation asynchrone**

- Deux alternatives

- ✦ Programmation événementielle : event handlers + callbacks

- ✦ Promesses

- Exemple : création simple

```
const promise = new Promise((resolve, reject) => {  
  try {  
    const pi = computePiIn30seconds();  
    resolve(pi);  
  } catch(e) {  
    reject("machine too slow!");  
  }  
});
```

Rappels JavaScript / ECMAScript



- Programmation asynchrone

- Deux alternatives

- ✦ Programmation événementielle : event handlers + callbacks

- ✦ Promesses

- Exemple : création avec paramètres (closure)

```
function maFonctionAsynchrone(nbDecimales) {  
  return promise = new Promise((resolve, reject) => {  
    try {  
      const pi = computePiIn30seconds(nbDecimales);  
      resolve(pi);  
    } catch(e) {  
      reject("machine too slow!");  
    }  
  });  
}
```


Rappels JavaScript / ECMAScript



- Programmation asynchrone

- Deux alternatives

- ✦ Programmation événementielle : event handlers + callbacks

- ✦ Promesses

- Exemple : utilisation

```
maFonctionAsynchrone(150).then(  
  (result) => { console.log(result); }  
).catch(  
  (error) => { console.err(error); }  
).finally(  
  () => { alert("Résultat disponible dans la console"); }  
);
```

Rappels JavaScript / ECMAScript



- Programmation asynchrone

- Deux alternatives

- ✦ Programmation événementielle : event handlers + callbacks
 - ✦ Promesses
 - « Re-synchroniser » une promesse
 - Mots-clés `async/await`

```
async function maFonctionSynchrone() {  
  try {  
    const result = await maFonctionAsynchrone(150);  
    console.log(resultat);  
  } catch(error) {  
    console.err(error);  
  }  
  alert("Résultat disponible dans la console");  
}
```

Outils de programmation avec XML



- Définitions

- Qu'est-ce qu'un parser ?

- ✦ « Un module logiciel [...] utilisé pour lire les documents XML et pour accéder à leur contenu et à leur structure. »

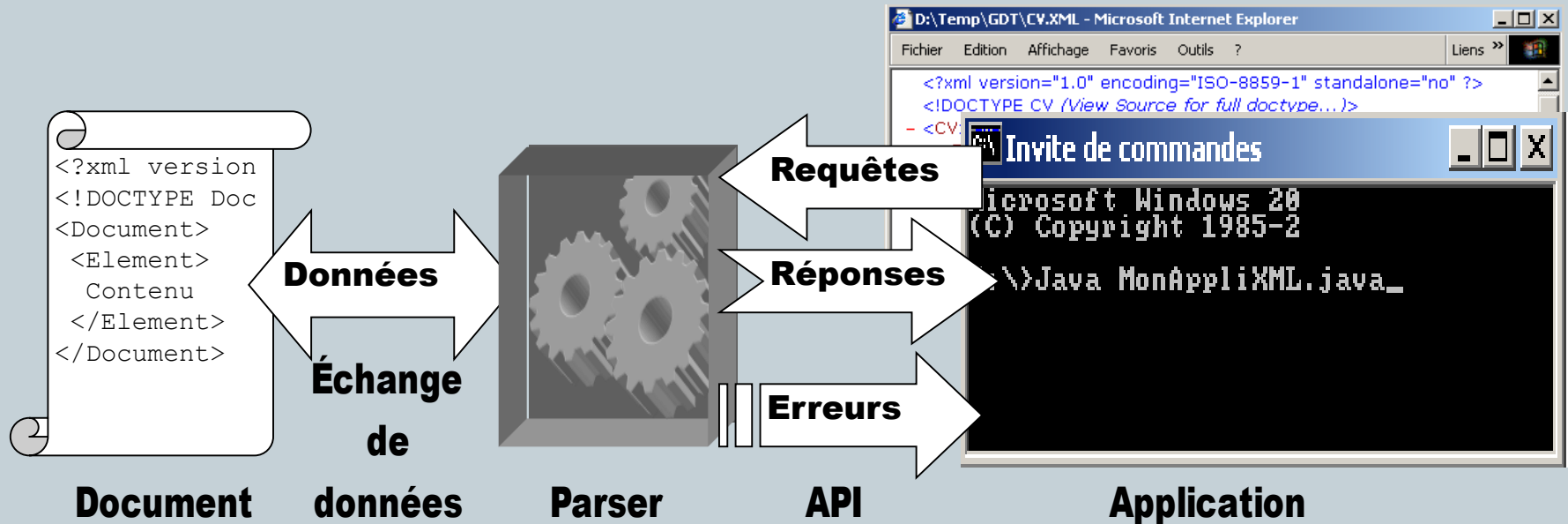
- Qu'est-ce qu'une application ?

- ✦ « On suppose qu'un processeur XML effectue son travail pour le compte d'un autre module, appelé l'application. »

http://babel.alis.com/web_ml/xml/REC-xml.fr.html#dt-xml-proc

Outils de programmation avec XML

- Communications entre parsers et applications
 - Rappel : Application Programming Interface
 - ✦ Outils
 - ✦ Protocole de communication
 - Schéma des échanges de données



L'API DOM (Document Object Model)



- Généralités
 - Modèle objet de document
 - Motivations
 - ✦ Rendre les applications W3 dynamiques
 - ✦ Accéder aux documents HTML et XML depuis un langage de programmation
 - Utilisations courantes
 - ✦ Intégré aux navigateurs
 - ✦ Utilisé en programmation comme API XML
 - Origine : DOM working group (W3C)
 - ✦ Début : 1997 ; fin : ...
 - ✦ But : standardiser les tentatives existantes

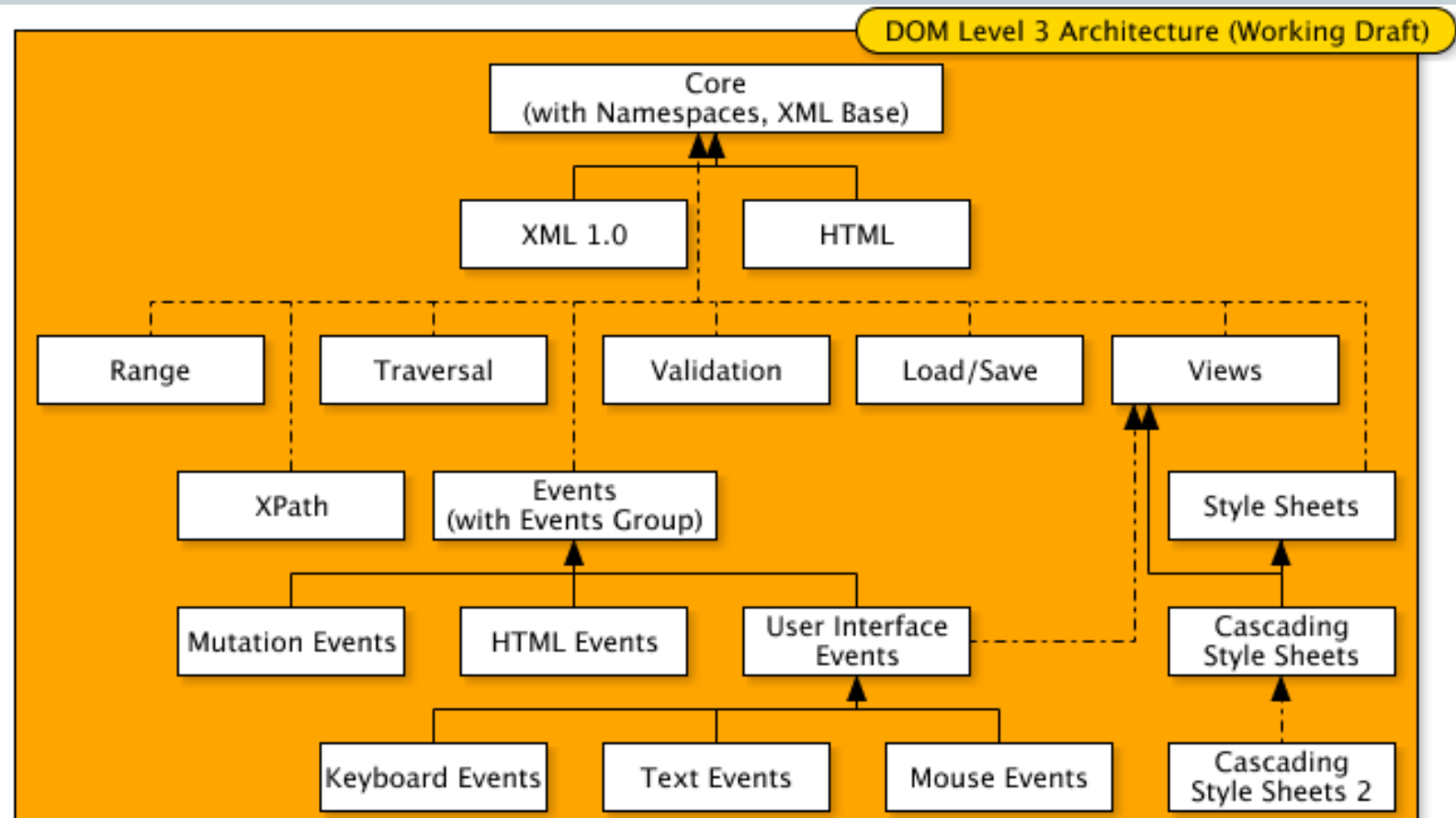
L'API DOM (Document Object Model)



- Principes fondamentaux
 - Représentation arborescente d'un document
 - ✦ Tout le document est chargé en mémoire
 - ✦ Navigation dans la structure arborescente
 - ✦ Représentation des nœuds par des *interfaces*
 - Propriétés
 - Méthodes
 - Recommandations sous forme de niveaux
 - ✦ Niveau 0 : avant...
 - ✦ Niveau 1 : octobre 1998
 - ✦ Niveau 2 : depuis novembre 2000
 - ✦ Niveau 3 : depuis janvier 2004

L'API DOM (Document Object Model)

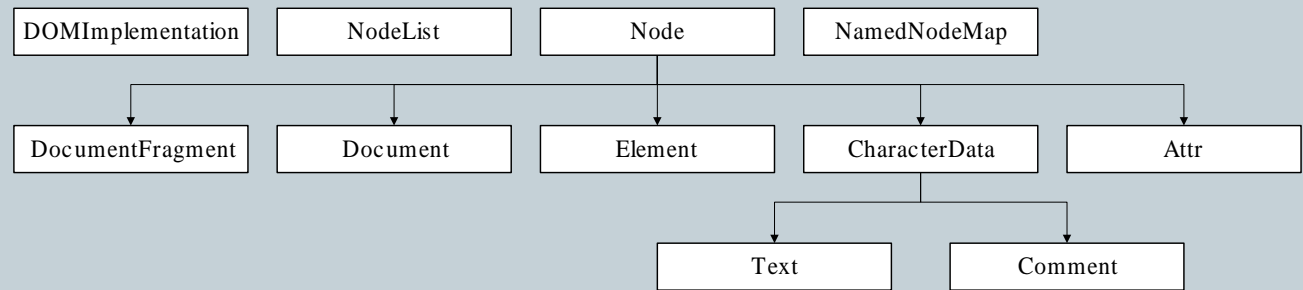
- Fonctionnalités



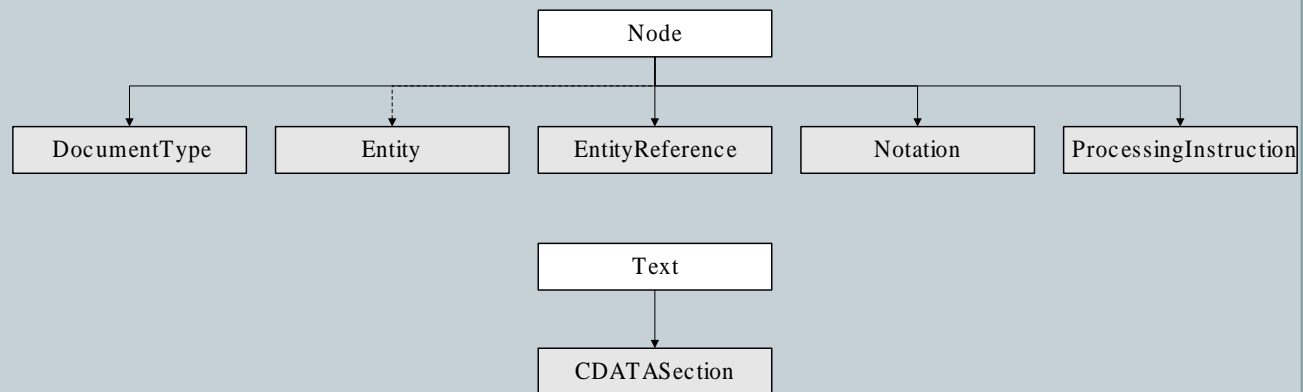
L'API DOM (Document Object Model)

- Interfaces

- DOM Core :



- DOM XML :



L'API DOM (Document Object Model)



- Interfaces DOM (Core et XML) les plus utilisées
 - Node : tout type de nœud de l'arbre DOM
 - ✦ Constantes
Tous les types de nœuds définis (exemple : `node.ELEMENT_NODE`)
 - ✦ Propriétés
`nodeName`, `nodeType`, `nodeValue`, `childNodes`, `textContent`
 - ✦ Méthodes
`insertBefore()`, `replaceChild()`, `removeChild()`, `appendChild()`, `cloneNode()`
 - NodeList : comme son nom l'indique...
 - ✦ Propriétés : `length`, `item(i)`

L'API DOM (Document Object Model)



- Interfaces DOM (Core et XML) les plus utilisées
 - Document : le nœud racine de l'arbre DOM
 - ✦ Dérive de Node
 - ✦ Propriétés
doctype, documentElement, encoding
 - ✦ Méthodes
createElement(name), createTextNode(), createAttribute(name),
getElementById(id), getElementsByTagName(name)
 - DocumentFragment : partie d'un document ; cf. Node

L'API DOM (Document Object Model)



- Interfaces DOM (Core et XML) les plus utilisées
 - Element : un élément, au sens HTML ou XML
 - ✦ Propriété : tagName
 - ✦ Méthodes
 - getAttribute(name), setAttribute(name, value), hasAttribute(name),
 - getAttributeNode(name), setAttributeNode(node),
 - removeAttribute(name), removeAttributeNode(node),
 - Attr : un attribut...
 - ✦ Propriétés : name, value, ownerElement

L'API DOM (Document Object Model)

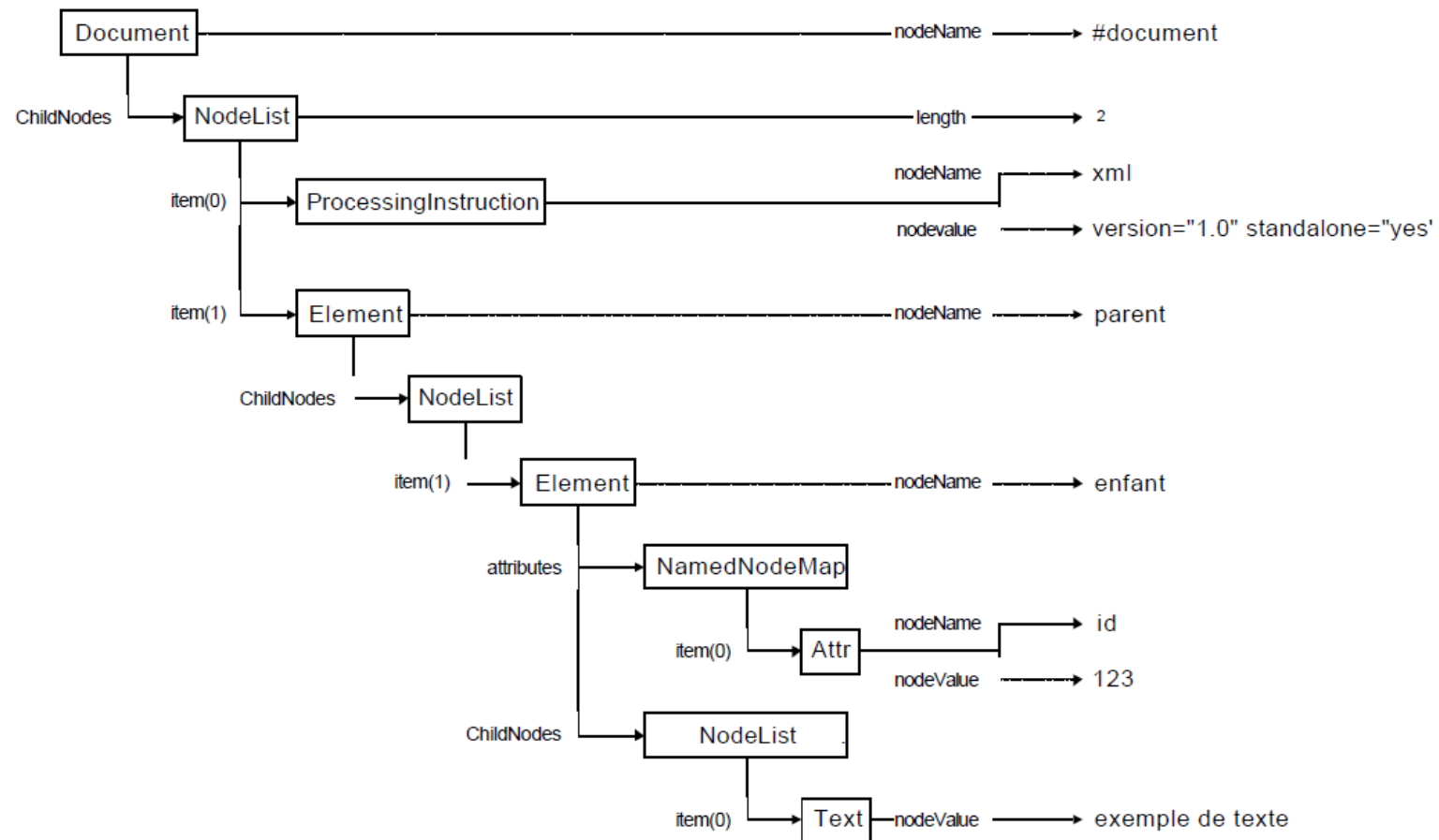


- Interfaces DOM (Core et XML) les plus utilisées
 - Text : nœud textuel (sous-nœud d'un élément)
 - ✦ Propriétés
data, length (héritées de CharacterData)
 - ✦ Méthodes
appendData(), insertData(), deleteData(), replaceData(),
substringData() (héritées de CharacterData)
replaceWholeText()

L'API DOM (Document Object Model)

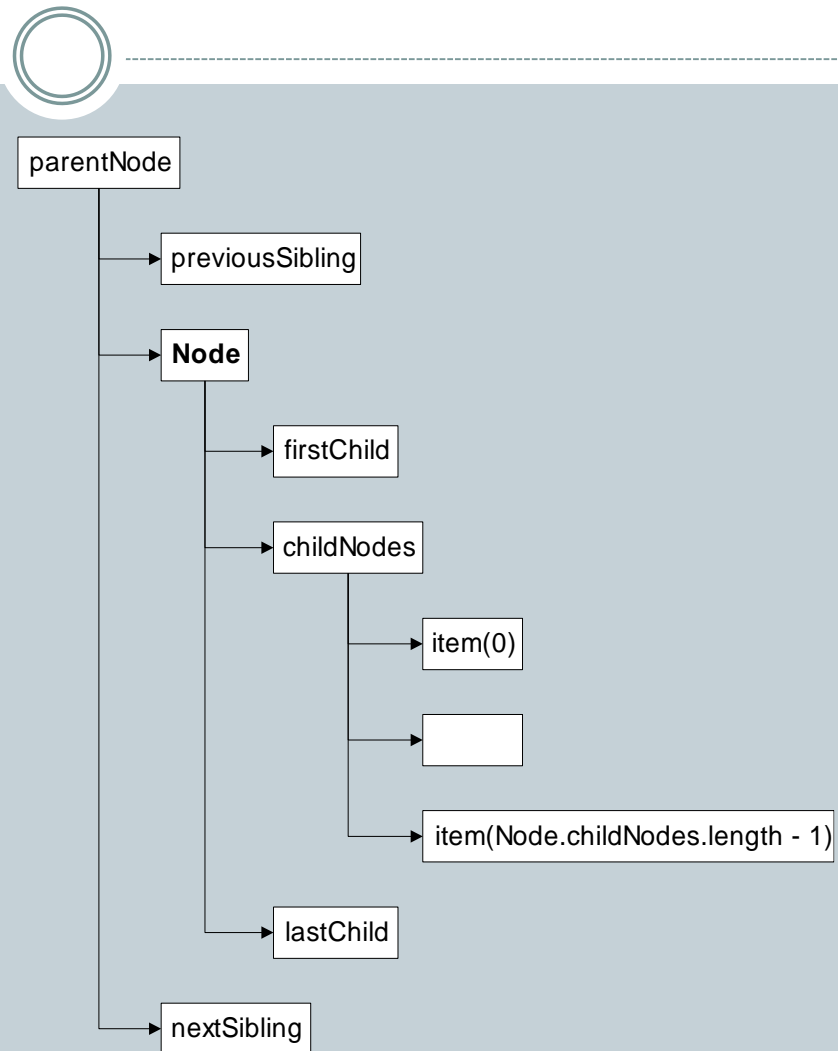


- Hiérarchisation des interfaces d'un document XML



L'API DOM (Document Object Model)

- Déplacement dans une arborescence DOM (interfaces du module Core)



L'API DOM (Document Object Model)



- Conclusion sur le DOM
 - Utilisation du DOM XML en JavaScript
 - ✦ Utilisation directe des propriétés
 - ✦ DOM XML relativement standardisé sur les navigateurs récents
 - Exemple : `document.getElementById()`
 - ✦ En revanche, DOM HTML plus dépendant du navigateur
 - Exemple : `monElement.innerHTML += ...;`
n'interprétait pas le nouveau code HTML sous IE 6 et 7
- Références sur le DOM
 - <http://www.w3.org/DOM/>
 - <http://www.w3schools.com/dom/>

Quelques références



- **Spécifications**

- En règle générale, la vérité est ici : <http://www.w3.org>
- ...Sauf quand elle est ailleurs :
 - ✧ <http://www.ecmascript.org/>
 - ✧ <http://www.typescriptlang.org/>
 - ✧ <http://json.org/>

- **Documentation et tutoriels**

- <https://perso.liris.cnrs.fr/romuald.thion/dokuwiki/doku.php?id=enseignement:lifap5:start>
- <https://developer.mozilla.org/fr>
- [http://msdn.microsoft.com/en-us/library/hbxc2t98\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/hbxc2t98(VS.85).aspx)
- <http://www.xul.fr/ajax-format-json.html>
- <http://es6-features.org/>
- <http://eloquentjavascript.net/>