Scripting côté client

M1 Architecture de l'Information

Remise à niveau en informatique

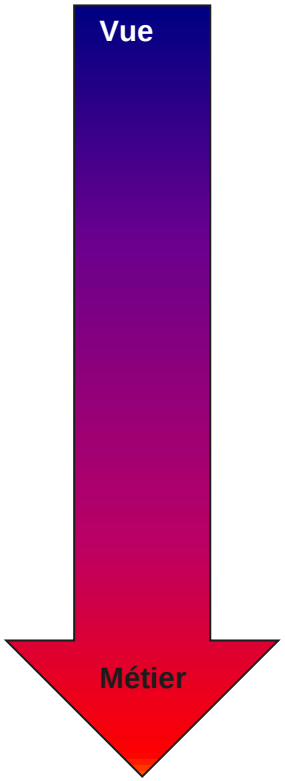
Lionel Médini

Objectifs des cours et des TP (partie Web)

- Découvrir ce qu'est le Web et comment ça marche
  - environnement réseau
  - mécanismes de base du Web
- Être capable de
  - créer une page Web simple
    - langage XHTML
  - mettre en forme une page ou un ensemble de pages
    - langage CSS niveaux 1 et 2
  - rajouter des contenus dynamiques dans une page
    - scripting côté client
  - créer un site Web basique
    - installation d'un serveur
    - publication de contenus
    - scripting côté serveur

Position du problème

- Différentes raisons de placer des traitements dans un navigateur
  - Rendre l'interface dynamique
    - Réagir aux événements utilisateur
  - Rendre le comportement fluide
    - Ne pas interrompre (recharger la page) à chaque action
  - Soulager le réseau
    - Ne pas renvoyer la mise en forme à chaque page, seulement les données
  - Alléger la charge du serveur
    - Déporter des traitements côté client



Aspects techniques

- 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 ?

Rappels de programmation orientée objet (POO)

Introduction : notion de modèle

- Un modèle représente un sujet d'étude (domaine)
  - Composé à partir de cas d'étude du domaine
- Un modèle doit être générique
  - S'applique à tous les éléments du domaine
- Un modèle est subjectif
  - Dépend du point de vue sur le sujet étudié
- Un modèle a une finalité
  - Conçu en fonction d'un objectif opérationnel
- Un modèle est exprimé à l'aide d'un langage
  - Structure la définition du modèle

Rappels de programmation orientée objet (POO)

Objet

- Un « truc » qui fait partie du domaine étudié par un programme
  - ➔ Dépend du point de vue de modélisation
- Un objet peut posséder
  - Un état défini par des variables : attributs — **Statique**
  - Un comportement défini par des fonctions : méthodes — **Dynamique**
- Exemples
  - L'utilisateur « lionel.medini »
  - L'étudiant « qagren »
  - Le bouton « validation » du formulaire « inscription » de la page « http://ens-lyon.fr/nouveaux-etudiants/ »
  - Le module de vérification de la sécurité des mots de passe

Rappels de programmation orientée objet (POO)

- Classe
  - Un ensemble de « trucs » qui ont des caractéristiques communes
    - ➔ Dépend également beaucoup du point de vue de modélisation
  - En d'autres termes : une abstraction des propriétés d'un objet
    - Permet de définir une fois les attributs et les méthodes pour plusieurs objets
    - Un objet est une instance d'une classe s'il correspond à sa définition
  - Exemples
    - Un utilisateur :
      - Possède un login, un mot de passe, peut se connecter…
    - Un étudiant
      - Possède un numéro étudiant, est lié à une formation, peut accéder à ses notes…
    - Un bouton de formulaire
      - Possède une taille, un texte, déclenche une action quand il est cliqué…
    - Un module de vérification de la sécurité des mots de passe
      - Vérifie la sécurité des mots de passe (d'où son nom)

Rappels de programmation orientée objet (POO)

Interface

- Une représentation externe d'une classe d'objets
  - Les informations qu'elle expose
  - Les traitements qu'elle peut effectuer
- Vision « boîte noire »
  - Pas d'indication du fonctionnement interne
  - Informations suffisantes pour l'utilisation
    - Notamment : le prototype des fonctions
- Exemples
  - Mode d'emploi d'un appareil
  - Contrat d'utilisation d'un service
- Une classe réalise (implémente) une interface quand elle permet son fonctionnement

Rappels JavaScript / ECMAScript

- Caractéristiques
  - Interprété
  - Fonctionnel
  - Orienté prototype
    - « object-based » plutôt qu'« object-oriented »
    - Typage dynamique : types associés aux instances et non aux classes
  - Événementiel
    - Mécanismes de « callback » (eventListener)
- Fonctionnalités
  - Reflection
  - E4X : ECMAScript for XML (ECMA-357)
  - JSON
  - …

Rappels JavaScript / ECMAScript

- Programmation orientée prototype
  - 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

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.epershand.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

- 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

Plusieurs manières d'écrire des scripts côté client

- Dans des attributs HTML
  - `<body onload="javascript:mafonction(parametre);">`
  - `<body onload="mafonction(parametre)">`
  - Liés au déclenchement d'un événement
- Dans la page Web
  - ```
    <script language="javascript">
      Contenu du script
    </script>
    ```
- Dans un fichier .js externe
  - ```
    <script language="javascript"
            src="URL_du_fichier.js"></script>
    ```
  - Attention : il faut une balise ouvrante et une balise fermante (pas de balise d'élément vide)

Déclenchement de l'exécution des scripts

- Dépend de leur position dans la page
  - Dans un attribut
    - Déclenchement de l'événement auquel il est lié
    - ➔ Programmation événementielle (EventListener)
  - Dans l'élément `head`
    - Chargement à la lecture de la page
    - Exécution après le chargement
    - ➔ Programmation impérative classique
  - Dans l'élément `body`
    - En fonction de la position de l'élément dans l'arborescence
    - ➔ Mettre une balise `<script>` à la fin du `body` a le même effet que `<body onload="...">`
    - ➔ Programmation impérative classique

Exercice 1

- Recopiez la page structure.html
- Rajoutez-y les scripts suivants
  - Le script téléchargé (par le navigateur) à l'URL http://liris.cnrs.fr/lionel.medini/enseignement/Master_AI/Remise_a_niveau/TP/bonjour.js
  - Le même script, mais avec le code dans l'en-tête de la page Web
    - Changez le message affiché pour le distinguer du précédent
  - Le même script dans le corps de la page, avec une petite différence
    - Remplacez la fonction `alert` par la fonction `console.log`

Exercice 2

- Recopiez la page structure.html
- Rajoutez-y un paragraphe
- Rajoutez-y un script qui envoie une alerte quand on clique sur ce paragraphe
- Rajoutez-y deux scripts qui affichent dans la console le fait que la souris est au-dessus ou sort de ce paragraphe

Parser (analyseur) / processeur

- Définitions
  - Qu'est-ce qu'un parser ?
    - « Un module logiciel […] utilisé pour lire les documents […] et pour accéder à leur contenu et à leur structure. »
  - Qu'est-ce qu'une application ?
    - « On suppose qu'un processeur […] effectue son travail pour le compte d'un autre module, appelé l'application. »

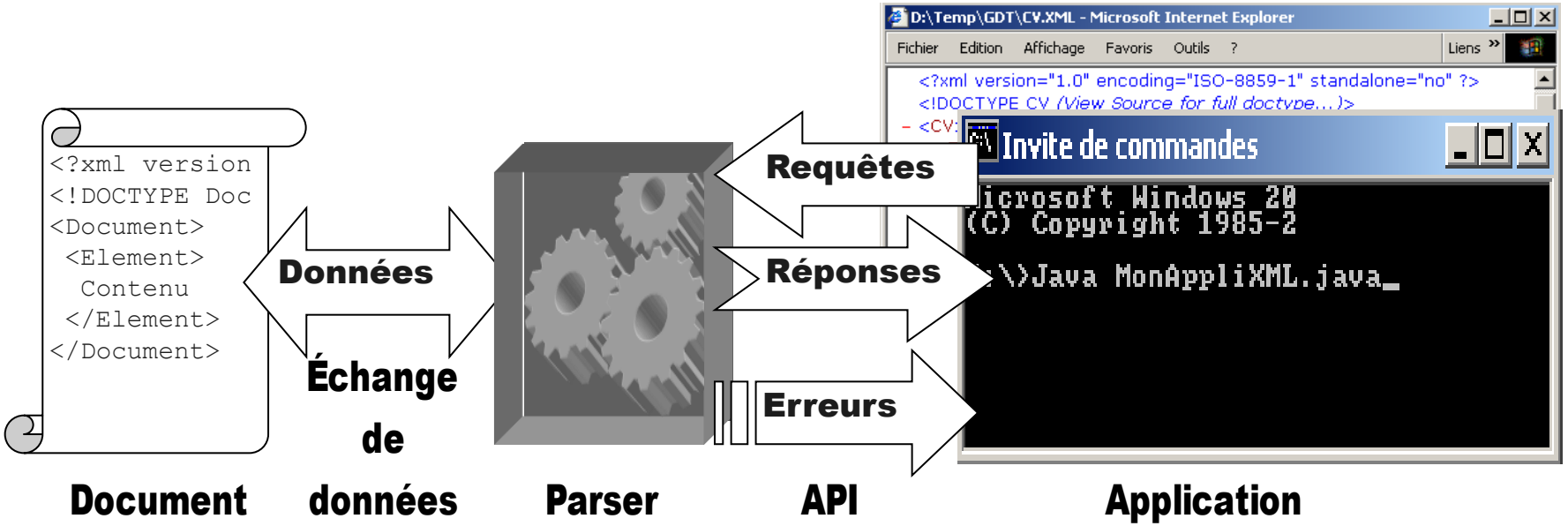
http://babel.alis.com/web_ml/xml/REC-xml.fr.html#dt-xml-proc

Application Programming Interface (API)

Principe général

- Standardisation des échanges entre modules logiciels
  - Outils
  - Protocole de communication

Exemple : communications entre parser et application



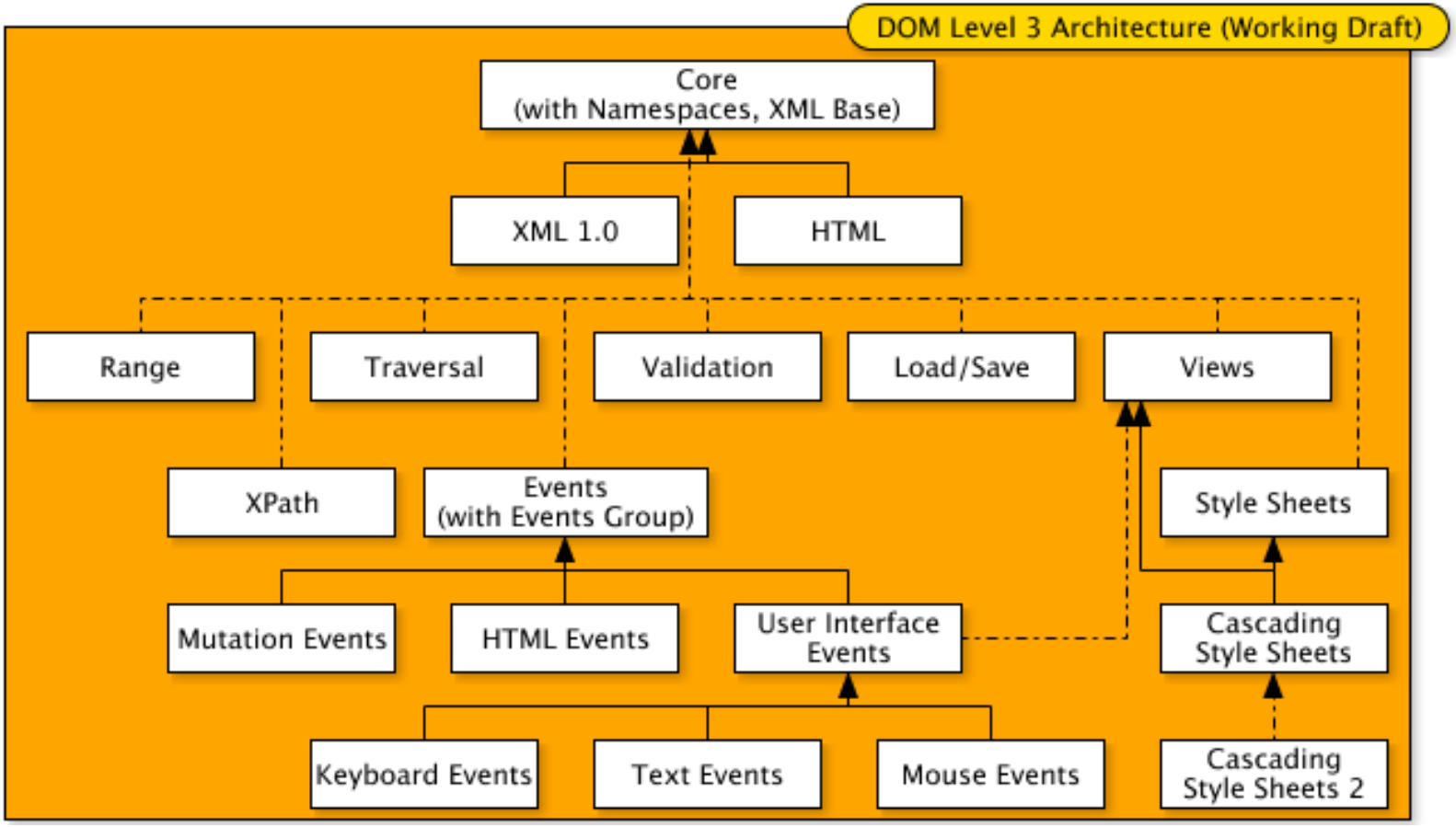
Généralités

- 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

Principes fondamentaux

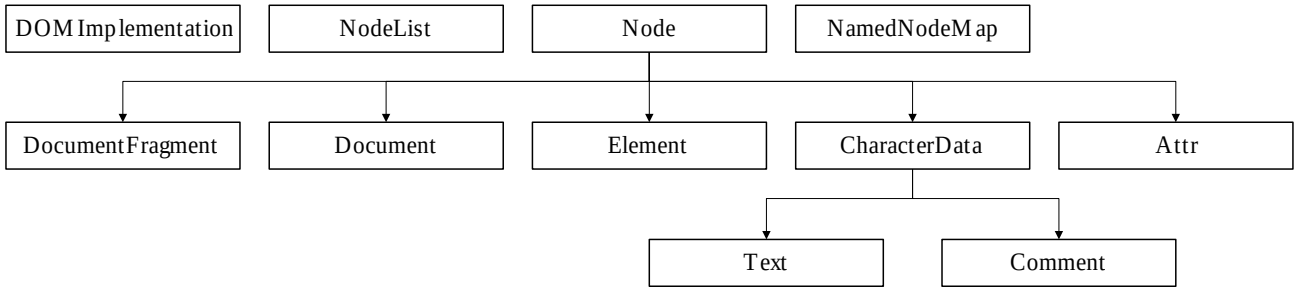
- Représentation arborescente d'un document
  - Tout le document est chargé en mémoire
  - Navigation dans la structure arborescente
  - Représentation des nœuds par des *interfaces*
    - Propriétés
    - Méthodes
- Recommandations sous forme de niveaux
  - Niveau 0 : avant…
  - Niveau 1 : octobre 1998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

L'API DOM (Document Object Model)

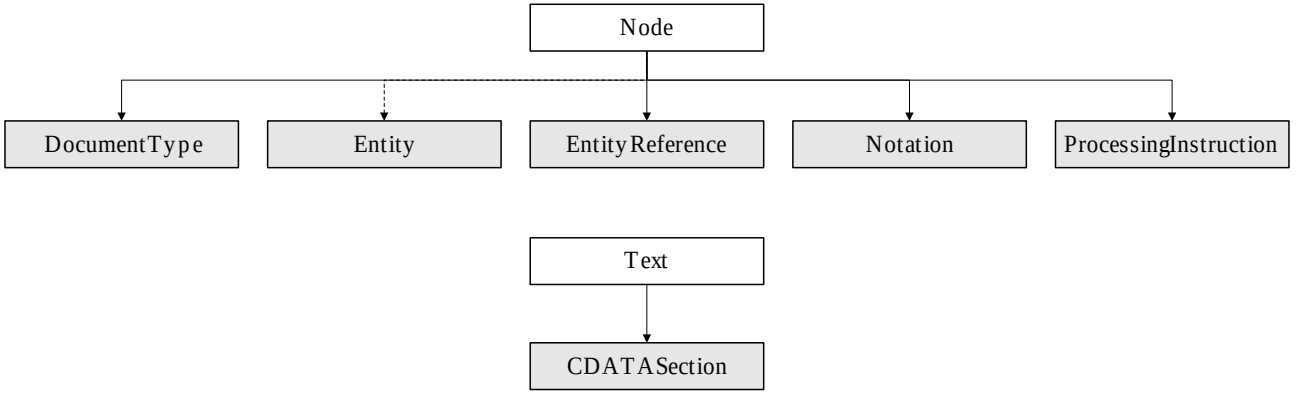


Interfaces

- DOM Core :



- DOM XML :



Interfaces DOM (Core) 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)

Interfaces DOM (Core) 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

Interfaces DOM (Core) les plus utilisées

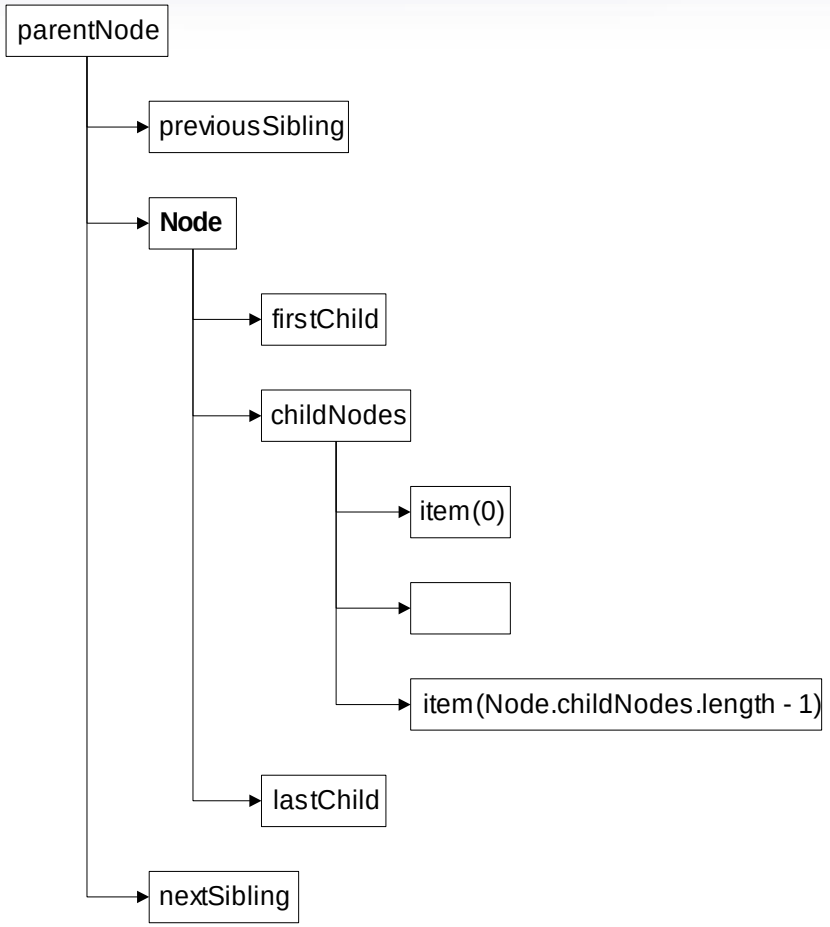
- Element : un élément, au sens HTML ou XML
  - Propriété : tagName
  - Méthodes
    getAttribute(name), setAttribute(name, value),
    hasAttribute(name), getAttributeNode(name),
    setAttributeNode(node),
    removeAttribute(name), removeAttributeNode(node),
- Attr : un attribut…
  - Propriétés : name, value, ownerElement

Interfaces DOM (Core) 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()

Déplacement dans une arborescence DOM

- Interfaces du module Core



Accès aux éléments de la page par le DOM (Core)

- Déplacement dans l'arborescence
  - `Node.firstChild`, `lastChild`, etc.
- Modification de l'arborescence
  - `document.createElement`, `Node.appendChild`
  - `document.removeElement`
- Raccourcis
  - `document.getElementById` → Element
  - `document.getElementsByTagName` → NodeList
- Valeur textuelle d'un élément
  - `Node.value`

DOM HTML (Level 2)

- Recommendation du W3C
  - Niveau 2, version 1 (09/01/2003)
  - http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/
- Interfaces spécifiques liées à la spécification (X)HTML
  - Permettre plus spécifiquement la gestion des contenus d'une page Web
  - …par exemple à l'aide de scripts JavaScript

Propriétés d'interfaces DOM HTML les plus utilisées

- Accès aux contenus par le DOM HTML
  - `Node.innerHTML`
  - `Element.style`
  - `Element.className`
  - `Document.getElementsByClassName` (récemment standardisé)

Conclusion

Utilisation du DOM en JavaScript

- Accès direct aux propriétés et méthodes des objets
- DOM Core relativement standardisé sur les navigateurs récents
  - Exemple : document.getElementById()
- En revanche, DOM HTML plus dépendant du navigateur
  - Exemple : monElement.innerHTML = …;
    n'interprète pas le nouveau code HTML sous IE

Références

- http://www.w3.org/DOM/
- http://www.w3schools.com/dom/

Exercice 3

- Recopiez la page structure.html
- Rajoutez-y un paragraphe avec un id
- Rajoutez-y un script qui modifie le texte de ce paragraphe
- Modifiez ce script pour qu'il rajoute du texte à la fin du paragraphe existant, à chaque fois que vous cliquez dessus

Exercice 4

- Recopiez la page structure.html
- Rajoutez-y un paragraphe
  - Avec un id
  - Avec comme texte la chaîne « `1` »
- Rajoutez-y un script qui
  - récupère le texte de ce paragraphe
  - Le transforme en entier avec la fonction `parseInt`
  - Le multiplie par 2
  - Remplace la valeur du paragraphe par le résultat calculé
- Modifiez la page pour que ce script s'exécute à chaque fois que vous cliquez sur le paragraphe

Exercice 5

- Reprenez la page précédente
- Modifiez le script pour qu'il
  - rajoute un nouveau paragraphe en dessous du précédent
  - Recopie la valeur écrite dans le paragraphe du dessus (avant de la remplacer)

- Modifiez la structure de la page pour afficher les puissances de 2 dans un tableau en JavaScript…