Les technologiesdu Web (suite)

Interactivité et contenu dynamique

Un langagepour l'interactivité

Interactivité

> Wikipedia : "L'interactivité est une activité nécessitant la coopération de plusieurs êtres ou systèmes, naturels ou artificiels qui agissent en **ajustant leur comportement**."

- **ECMAscript** (ou plus couramment **JavaScript**)
- Développé à l'origine pour Netscape pour ses serveurs
  - Inspiré de Java (objets), mais nouvelle syntaxe plus **simple**
- Désormais **interprété par les navigateurs** usuels
- Véritable langage complet !
  - Il existe des interpréteur JS, voire des shell JS.

CSS 2 propose quelques rudiments de comportements interactifs (notamment avec le sélecteur « :hover » qui réagit au survol de la souris) et cette tendance devrait se poursuivre avec les versions futures, néanmoins les feuilles de styles restent destinées au domaine de la mise en forme et de la présentation. Elles sont donc faites pour adapter un ensemble d'informations au médium utilisé pour les parcourir et le résultat est donc essentiellement statique. De plus, elles ne sont pas faites pour modifier le contenu du document, seulement le présenter.

Pourtant, bien qu'il soit rarement (jamais) indispensable de réagir aux actions de l'utilisateur puisque toute l'information est censée être contenue dans le document (X)HTML, un accès intercatif à l'information est parfois beaucoup plus simple pour l'utilisateur. C'est souvent le cas quand les informations diffusées ne sont pas évidentes à interpréter, par exemple si leur quantité est importante ou si leur technicité n'est a priori pas à la portée de l'ensemble du public visé, ou tout simplement si une mauvaise interprétation des informations peut induire l'utilisateur en erreur. Réagir au comportement de l'utilisateur permet donc de proposer des guides pour sa compréhension. Une utilisation courante de comportements interactifs est d'aider à la saisie de formulaires. En effet, un formulaire en (X)HTML contient l'ensemble des questions posées et des champs destinés aux réponses. Cependant certains champs sont soumis à des règles de saisie (typographie, représentation des dates, etc ...) qui font que toute réponse n'est pas forcément cohérente. Même s'il est possible (et souhaitable) de préciser dans la question la manière dont l'utilisateur doit répondre, il est impossible de s'assurer dans le document (X)HTML que l'utilisateur donnera réellement une réponse cohérente. Proposer une aide au moment de la saisie qui indique les erreurs (voire les corrige) simplifie grandement l'expérience de l'utilisateur, surtout dans le cas de formulaires longs et complexes.

L'intercativité se traduit par la capacité du document à réagir aux actions de l'utilisateur. En pratique, ces actions sont traduites par l'agent utilisateur en évènements auxquels il est possible d'associer des instructions qui composent une réaction. Pour le Web, un langage standardisé permet d'accéder à la liste d'évènement que transmet le navigateur et de décrire les instructions associées. Ce langage, appelé JavaScript, est un langage de programmation à part entière qui est directement interprété par l'agent utilisateur. Ce langage étant un langage de programmation, il est aussi possible d'écrire des programmes destinés à s'exécuter autrement que dans un navigateur, et des interpréteurs de JavaScript indépendant existent.

Principesd'ECMAscript

- **Tout est objet** !
- Toute **fonction** peut-être un constructeur d'objet
- Déclaration dynamique des variables
- Existence d'un **état « undefined »**
- Libération de la mémoire automatique (**ramasse-miettes**)
- Nombreux **objets** et **fonctions** disponibles pour la manipulation de documents Web

JavaScript est un langage orienté objets à prototype. Les objets n'héritent pas de classes, mais sont construits par des fonctions (prototypes).

En JavaScript, tout est objet. Tous les types sont des objets, y compris les fonctions. Le type fonction est même lui-même un type d'objet puisqu'une fonction est un constructeur d'objet. Cela tient au fait que tout peut être déclaré dynamiquement, aussi bien les propriétés que les méthodes d'un objet, donc construire un objet est aussi simple que définir un nouvel objet vide puis de le peupler de méthodes et de propriétés à n'importe quel moment dans le programme. Si toutes les définitions de propriétés et de méthodes sont regroupées dans une seule fonction, alors cette fonction est un constructeur, facile ! L'exemple est fourni sur la diapositive suivante. Le mot clef `this` représente toujours l'objet appelant un méthode.

Puisque tout peut être déclaré dynamiquement, une variable peut être utilisée sans déclaration et peut changer de type à tout moment. Il est même possible d'accéder à une variable qui n'a jamais été définie, tant qu'elle n'est pas encore été définie elle est de type indéfini (« undefined »). Tout comme il est inutile de déclarer les objets, il est inutile de les libérer, l'interpréteur s'en charge automatiquement quand il n'est plus utilisé (système de ramasse-miettes), ainsi le code JavaScript ne décrit que l'utilisation des variables et permet au développeur de se concentrer sur leur rôle sans avoir à se préoccuper de la manière dont elles doivent être représentées en mémoire pour remplir au mieux ce rôle.

Attention, la libération de la mémoire n'est vraiment pas du ressort du développeur en JavaScript. Mettre la valeur d'un objet à « `undefined` » ne le libère pas. L'objet ne sera libéré que par le ramasse-miettes, et en attendant il demeure dans un état défini, mais avec le type « `undefined` » (ce qui rendra difficile de savoir s'il a réellement été libéré ou non).

Par défaut, les variables ont un spectre global si elles ne sont attribuées à aucun objet. Pour l'écriture de fonctions, il est souvent nécessaire d'utiliser des variables temporaires associées à aucun objet qui n'ont pas de rôle à jouer à l'extérieur de la fonction. Ces variables doivent être précédées de `var` lors de leur déclaration (ou de leur première utilisation).

Comme JavaScript est destiné au Web, les interpréteurs des navigateurs Web disposent de plein de fonctions et d'objets prédéfinis qui décrivent les documents et facilitent leur modification.

Exemple de code

```
function personne () {
  function reset () {
    this.nom = "Martin";
    this.prenom = "Jacques";
  }
  this.reset();
}

philippe= new personne();
philippe.nom = "Risoli";
philippe.prenom = "Philippe";
```

Cet exemple montre comment écrire une fonction pour qu'elle serve de constructeur d'objet. Il y a une différence fondamentale entre cette méthode et celle qui consiste à créer un objet de type générique et de le peupler de propriétés par la suite. En effet, `new` permet une notion d'héritage et créera dans notre objet `personne`, une propriété `prototype` qui indiquera l'objet dont il est hérité. Si l'on peuple après coup un objet générique, son prototype est de type `null` et il n'a de fait pas de parent.

Par contre, une fonction est un objet comme un autre, et elle est déclarée ici en tant que telle pour des raisons de lisibilité du code. En fait, la fonction constructeur est déjà une première instance de l'objet. Elle aurait pu être déclarée :

```
personne = new Object ();
personne.reset = function () {
  this.nom = "Martin";
  this.prenom = "Jacques";
}
personne.reset();
```

Le reste du code de cet exemple n'aurait pas eu à être modifié. Il demeure tout de même une différence entre les deux. En fait, même le mot-clef `function` n'est qu'un raccourci de `new Function ()` qui créé un objet fonction. Pour être strictement équivalent à l'exemple du dessus, il aurait donc fallu déclarer `personne` comme un nouvel objet fonction au lieu d'un nouvel objet générique, cependant comme `personne` ne devrait pas servir de fonction, il n'y a aucune raison de le déclarer aini. Pour être complètement cohérent, c'est le premier exemple qu'il faudrait changer pour qu'il n'hérite pas de l'objet fonction, cependant tant que l'objet fonction n'est pas modifié, il n'y a aucune implication pratique de cette différence.

Pour parfaitement comprendre toutes ces nuances, il faudrait présenter la propriété `prototype` des objets, mais ce n'est pas le but de cette présentation de rentrer dans tant de détail. Ceux qui sont intéressés trouveront des informations ailleurs, ce niveau de compréhension n'est pas utile pour commencer à écrire des scripts.

Manipulations du DOM

- `getElementById("toto")`
- `getElementsByClass("titi")`
- `getElementsByTagName"ul"`
- `createTextNode("Lorem ipsum !")`
- `createElement("img")`
- `createAttribute("alt")`
- `insertBefore(..., ...)` et `appendChild(...)`
- `removeChild(...)`

Les navigateurs proposent un certain nombre d'objets et de fonctions pré-définies pour manipuler le DOM. Certaines permettent de sélectionner des éléments en fonction de critères sur leur type (`getElementsByTagName`) ou leurs propriétés (notamment leur classe et leur identifiant dans le cas de `getElementsByClass` et `getElementById`). Attention, toutes les fonctions données en exemple ici renvoient une liste dynamique des éléments sélectionnés. L'objet liste dynamique se met à jour à chaque modification du DOM, il faut donc être très attentif aux boucles sur type de listes si des itérations peuvent modifier le DOM.

D'autres fonctions permettent de créer des nœuds dans le DOM, soit des éléments (`createElement`) soit des nœuds textuels (`createTextNode`). Il est aussi possible de modifier les attributs des éléments, en supprimer ou en ajouter (`createAttribute`, etc ...).

À chaque élément du DOM correspond un type d'objet pré-defini pour pouvoir créer des éléments du DOM de toute pièce depuis les instructions JavaScript ou pour pouvoir copier un élément du DOM à un autre endroit de celui-ci. En effet, à moins de copier un élément du DOM dans un nouvel objet, les opérations d'insertion dans le DOM (`insertBefore` et `appendChild`) déplacent les éléments (et les suppriment donc de leur position initiale). Le DOM étant une arborescence, un élément ne peut pas être à plusieurs endroits en même temps ! `removeChild` et d'autres fonctions permettent aussi de supprimer des éléments du DOM.

Avec toutes ces fonctions (et encore bien d'autres) il est tout à fait possible de modifier le DOM, et donc de modifier/créer/supprimer le contenu d'un document Web. Il est donc possible de créer du contenu à la demande de l'utilisateur (dynamiquement). Il faut toutefois rester cohérent, le principe de séparation des rôles qui veut isoler le contenu d'un document et sa présentation s'applique aussi à l'interactivité. Bien que ce soit possible de tout mélanger, il vaut toujours mieux respecter une organisation cohérente et laisser tout le contenu au document (X)HTML, la présentation au feuilles des style et l'interactivité au scripts JavaScript.

Insertion dans une page Web

- Comme CSS : **inclu** dans le document ou fichier **séparé**
- Élément **<script>**
- **Fichier séparé** lié au document : plus adapté à la séparation des rôles (contenu/mise en forme/interactivité)
  - `<script src="ui/default/slides.js" type="text/javascript"></script>`
- Association d'instructions à un évènement :
  - `element.addEventListener(event, listener, bubbling)`

L'élément <script> permet d'associer du code JavaScript à un document (X)HTML, soit en écrivant le code directement entre la balise ouvrante et la balise fermante de cet élément, soit en liant un fichier externe avec l'attribut « src » (à la manière des images et de l'élément <img>). La solution qui consiste à lier un fichier séparé répond bien à la problématique de séparation des rôles, dans ce cas le plus logique est de l'inclure dans les en-têtes du document (en tant qu'enfant de l'élément <head>) comme une méta-donnée. L'élément <script> peut aussi être inclu directement dans le corps du document (X)HTML avec le contenu. L'intérêt de cette deuxième approche est que le code est s'exécute au moment ou il apparaît dans le documenet pendant que se charge le DOM, ce qui permet d'exécuter des instructions avant que la page ne soit complètement chargée, tout en disposant déjà des éléments du DOM précédents le script. Pour l'ensemble des autres opérations, il vaut mieux associer des instructions à des évènements (l'évènement « load » existe pour exécuter des scripts au chargement du document).

La liste des évènements est très longue et chaque élément peut provoquer ses propres évènements, la fonction `addEventListener` permet d'associer à tout évènement une expression JS (en général un appel à une fonction). À l'heure de cette présentation, `addEventListener` n'est toujours pas implémenté dans IE qui utilise une fois encore son propre fonctionnement (les curieux le chercheront dans cette magnifique source d'information qu'est Internet).

Utilisation complexe en pratique

- Langage standardisé mais **support variable** d'un navigateur à l'autre
- **Difficile** d'écrire du **code portable** pour tous les navigateurs usuels (parce-qu'IE en est un)
- **Tests** et code spécifique (grâce à l'état « **undefined** ») :

```
if (typeOf(document.getElementById) == "function") {
  /* Le navigateur supporte les manips du DOM avec JavaScript */
} else {
  /* Il va falloir trouver une autre méthode */
  /* (probablement spécifique au navigateur) */
}
```

JavaScript a beau être un standard défini dans une norme, tous les navigateurs ne se conforment pas aussi bien à la norme. D'une part, il faut un certain temps pour implémenter un standard et les standards évoluent avec le temps, donc selon leur âge, les navigateurs supportent plus ou moins les standards actuels. D'autre part, certains navigateurs préfèrent imposer leur propre fonctionnement (pratique utile pour entretenir un monopole, ou favoriser le développement de technologies propriétaires). Il ne faut donc pas être surpris de retrouver IE parmi les navigateurs les plus incompatibles avec le standard. Malgré la norme ECMA et le recommandations du W3C, JavaScript n'est donc pas un véritable standard dans les faits, il est donc difficile d'écrire du code JavaScript compatible avec l'ensemble des navigateurs.

Heureusement, pour vérifier l'existence ou non de l'ensemble des objets et fonctions pré-définis dans un navigateur, il est possible de tester leur état. Les fonctions et objets inexistants étants tous « undefined » il est très facile de vérifier que le code fonctionne sur un navigateur donné au moment de l'exécution. Il est donc courant décrire du code qui teste au moment de son exécution l'environnement dans lequel il s'exécute et vérifie l'existence des fonctions avant de les appeler. Avec cette méthode, il faut réécrire autant de fois une même opération avec du code spécifique qu'il existe de navigateurs différents à supporter.

Comme le support de JavaScript peut être carrément nul, il faut toujours s'assurer que le malfonctionnement d'une fonction ne perturbe pas l'accès au contenu du document.

Contenu dynamique

- Côté serveur : **PHP**
  - PHP: Hypertext Preprocessor : préprocesseur, fabrique le document (X)HTML
  - JavaScript peu utilisé côté serveur.
- Envoi/réception de données : **AJAX**
  - Asynchronous JavaScript And XML : échange de contenu entre le client et le serveur sans recharger le document Application Web
  - De plus en plus utilisé : pousse à l'adoption du DOM et du support de JS dans les navigateurs.
  - Attention à l'accessibilité !
  - URL ?

Bien que JavaScript ait initialement été développé pour être un langage interprété côté serveur, PHP est aujourd'hui largement plus utilisé et JavaScript est devenu le langage de référence pour les scripts côté client grâce à son support dans les principaux navigateurs.

L'exécution de script côté serveur permet de fabriquer les documents Web et l'ensemble des fichiers (comme les CSS ou même les scripts) qui sont servis aux clients. Ils peuvent ainsi être fabriqués pour répondre à des demandes précises sans que tous les fichiers (X)HTML répondant à toutes les demandes possibles et imaginables n'aient besoin d'avoir été pré-écrits. En général, le contenu qui alimente les documents Web ainsi produits provient d'une base de donnée. PHP permet aisément de s'interfacer avec différentes bases de données. Ce système fournit du contenu à la demande, mais une fois le document formé et servi au client son contenu est figé jusqu'à la nouvelle requête du client (nouvelle URL à visiter).

Pour fournir du contenu dynamique en réponse à des actions de l'utilisateur (interactivement), il faut ajouter au document du JavaScript exécuté par le client qui effectue des requêtes au serveur pour compléter/modifier le contenu d'un document. Fournir du contenu à la demande et de manière interactive se fait de plus en plus sur le Web actuellement (et prend le nom de Web 2.0) et permet notamment de développer de véritables applications qui tournent dans un navigateur internet (traitement de texte, éditeur d'image, etc ...).

Comme ces nouvelles applications utilisent au maximum les possibilités de JavaScript et du DOM, elles ont tendance à pousser les développeurs de navigateurs à se conformer aux standards pour rester ou devenir compatibles avec elles. Le Web 2.0 est aujourd'hui un enjeu marketting très important puisqu'il perment aux prestataires de services qui proposent les applications en ligne d'héberger toutes les données manipulées par ces applications sur leur serveur, et donc d'y avoir accès à des fins commerciales.

Le Web 2.0 se développe, mais qu'en est-il du vieillissant Web 1.0 ? Quand quelques firmes disposeront de toutes les informations sur la vie privée de leurs utilisateurs, le Web sera-t-il pour autant plus riche ? Les applications Web remettent en cause le fonctionnement même du web en tournant le dos à nombre d'agents utilisateurs puisqu'elles utilisent des standards modernes sans diffuser de contenu à qui ne les supporte pas. En effet, le contenu étant envoyé au goutte à goutte à travers des requêtes JavaScript interactives, le document Web de base ne contient qu'une portion réduite de l'information disponible sur le serveur. Ainsi, le Web qui regroupait au départ un ensemble de technologies pour favoriser le partage d'information se retrouve l'acteur de la concentration des données (mêmes personnelles) chez quelques hébergeurs tout en perdant son objectif de toujours diffuser l'information quel que soient les capacités du client.

Si le document se remplit d'information interactivement, son URL n'a plus aucun sens. L'URL n'identifie plus une ressource mais uniquement l'application qui permet d'atteindre la ressource. La toile tissée par l'ensemble des liens d'une ressource à une autre qui constitue le Web s'arrête dès qu'un lien pointe sur une application Web 2.0. Dans ces conditions le Web 2.0 est bien une réalité que permettent les technologies du Web, mais est loin d'être une nouvelle version du Web (que l'on peut rétrospectivement appeler 1.0), elle est une autre voie qui avait volontairement été évitée dès le commencement du Web.