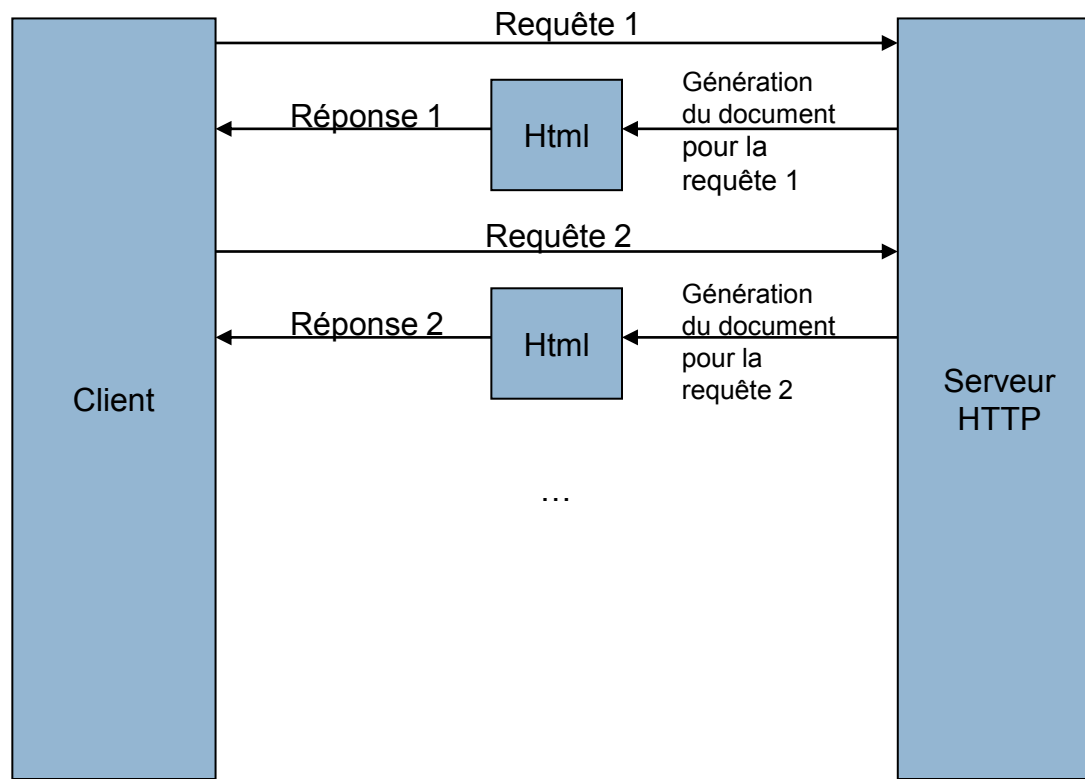


AJAX

Application traditionnelle

- Application WEB traditionnelle :
 - ▣ Le client envoie une requête HTTP
 - ▣ Le serveur renvoie une page



Application traditionnelle

□ Inconvénients :

▣ Consommation inutile de la bande passante :

- Une grande partie du code est commun aux différentes pages. Le navigateur gère un cache mais pas pour des parties d'un fichier HTML

▣ Le chargement d'une nouvelle page n'est pas ergonomique :

- Délai variable pendant lequel la page n'est pas affichée ou seulement partiellement
- A comparer à une application traditionnelle

Application traditionnelle

□ Avantages :

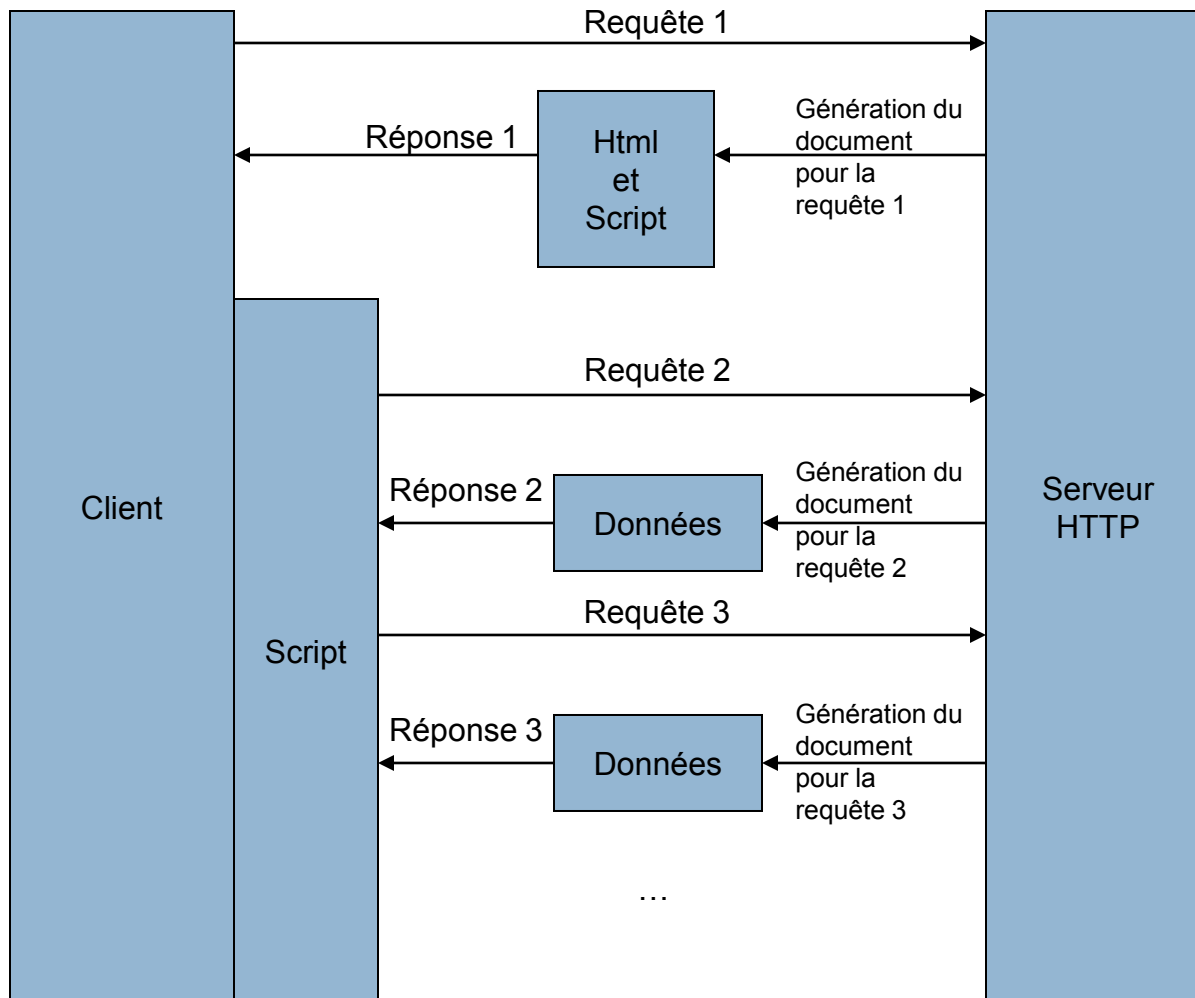
- ▣ Le serveur et le client ont chacun un travail
 - L'application ne doit donc pas être prise en charge entièrement d'un côté ou de l'autre
- ▣ Tout ne peut pas être confié au client :
 - Manque de sécurité/confiance
 - On ne veut pas envoyer la base de données au client
 - Limitations en puissance de calcul

Pourquoi AJAX

- Nécessité de communication entre client et serveur
 - Volonté de ne pas perdre de bande passante

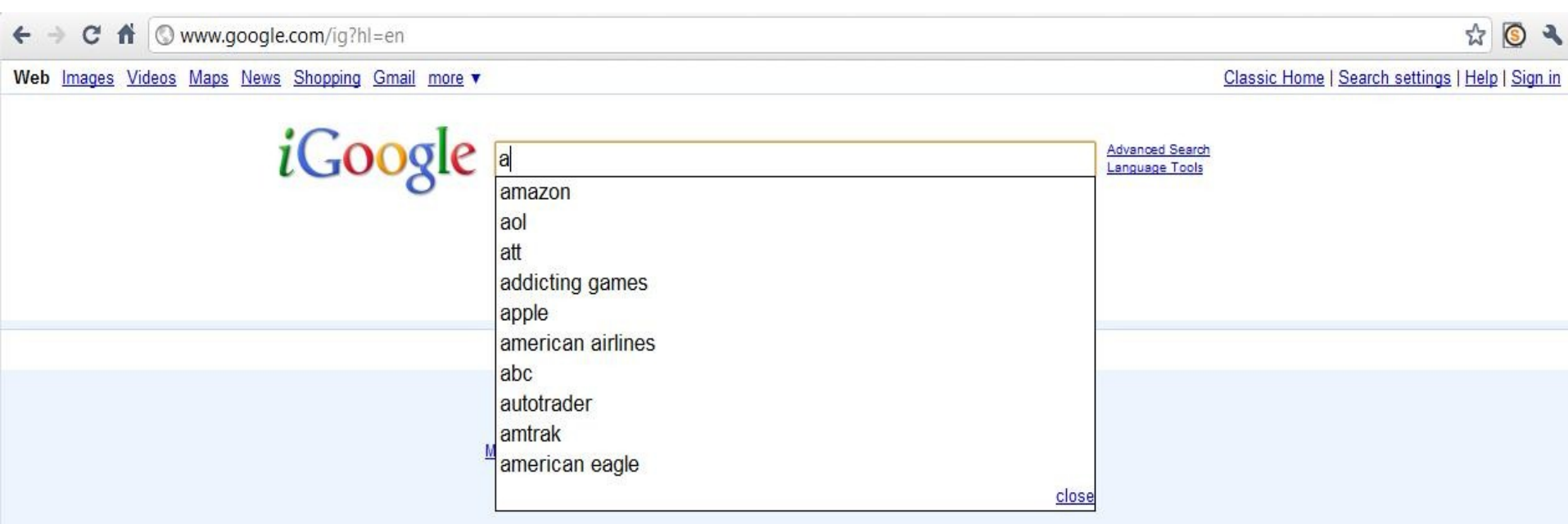
 - Principe de base :
 - ▣ Javascript émet des requêtes vers le serveur
 - ▣ Le serveur répond avec uniquement les informations demandées
 - ▣ Javascript traite les données reçues et modifie la page si nécessaire
 - ▣ Tout se passe sans rechargement de la page
- = AJAX / Asynchronous Javascript and XML

AJAX



Exemple d'utilisation

- Auto complétion d'un moteur de recherche :
 - L'utilisateur commence à taper "xxx"
 - La liste des requêtes possibles s'affiche



Exemple d'utilisation

```
<html><head>
<script type="text/javascript">
function getAnswers(query) {...}

function displayAnswers(ml) {
  var answers = getAnswers(ml);
  var list = "";
  for (i=0; i<answers.length ; i++)
    list += "<li>" + answers[i] + "</li>";
  document.getElementById("answers").innerHTML = list;
}
</script>
</head><body><form>
<input id="search" type="text" value="" onKeyUp="displayAnswers(this.value);">
<ul id="answers">
</ul>
</form></body></html>
```

Exemple d'utilisation

- Moteur de recherche :
 - ▣ L'utilisateur commence à taper "xxx"
 - ▣ Javascript récupère le "xxx"
 - ▣ Puis demande au serveur les recherches fréquentes commençant par "xxx"
 - ▣ Et affiche la réponse du serveur
- Il faut juste écrire la fonction `getAnswers()`
- Le tout prend moins d'une seconde, c'est transparent pour l'utilisateur

Qui utilise Ajax

- ❑ Clients de messagerie : Gmail, Yahoo Mail, HotMail
- ❑ Google, Google Maps
- ❑ Flickr, Picasa
- ❑ Deezer
- ❑ Youtube, Dailymotion
- ❑ Myspace, Facebook

Comment ça marche

- Un exemple sans AJAX direct :
 - ▣ requête faite automatiquement par le navigateur
 - ▣ récupération d'une image à distance

```
<body onLoad="javascript:document.images[0].src =  
'http://...'">
```

```
</a>
```

```
</body>
```

Comment ça marche

- `function getAnswers(query) {...}`
- 1. La fonction `getAnswers()` crée un objet `XMLHttpRequest`
 - ▣ Existe sur tous les navigateurs "modernes".
- 2. Cet objet permet de faire une requête vers la page concernée
 - ▣ `http://www.google.com/...`
 - ▣ Conceptuellement identique à la recherche d'image.
- 3. On traite le résultat et on met à jour la page web

L'objet XMLHttpRequest

- AJAX se base sur XMLHttpRequest
 - ▣ Initialement développé par Microsoft (!), en tant qu'objet ActiveX, pour Internet Explorer 5
 - ▣ Puis repris et implémenté sous Mozilla 1 Safari 1.2, Konqueror 3.4 et Opera 8.
 - ▣ Pas supporté par certains vieux navigateurs.
 - ▣ Proposé en 2006 pour devenir une recommandation du W3C :
 - <http://www.w3.org/TR/XMLHttpRequest/>
 - Draft novembre 2009

L'objet XMLHttpRequest

- Problèmes :
 - ▣ Nécessite un navigateur compatible, autorisant le Javascript et XMLHttpRequest.
 - ▣ Nécessite plus de tests car il existe de grandes différences entre les navigateurs.
 - XMLHttpRequest n'est pas implémenté de la même manière selon les navigateurs (et les versions des navigateurs).
- Solution la plus simple :
 - ▣ Aller chercher le code sur Internet

Création de l'objet XMLHttpRequest

- Pour Internet Explorer (avant IE7) :
 - ▣ `xhr = new ActiveXObject("Microsoft.XMLHTTP");`
- Ou
 - ▣ `xhr = new ActiveXObject("Msxml2.XMLHTTP");`
- Pour les autres navigateurs :
 - ▣ `xhr = new XMLHttpRequest();`

Création de l'objet XMLHttpRequest

```
function getXMLHttpRequest() {  
    if (window.XMLHttpRequest) {  
        return new XMLHttpRequest();  
    } else {  
        if (window.ActiveXObject) {  
            try {  
                return new ActiveXObject("Msxml2.XMLHTTP");  
            } catch (e) {  
                try {  
                    return new ActiveXObject("Microsoft.XMLHTTP");  
                } catch (e) {  
                    return NULL;  
                }  
            }  
        }  
    }  
}
```

Création (bis)

```
function getXMLHttpRequest() {
    if (window.XMLHttpRequest)
        return new XMLHttpRequest();
    if (window.ActiveXObject) {
        var names = [
            "Msxml2.XMLHTTP.6.0",
            "Msxml2.XMLHTTP.3.0",
            "Msxml2.XMLHTTP",
            "Microsoft.XMLHTTP" ];
        for(var i in names) {
            try {
                return new ActiveXObject(names[i]);
            } catch(e) {}
        }
    }
    window.alert("Votre navigateur ne prend pas en charge l'objet XMLHttpRequest.");
    return null;
}
```

Méthodes de l'objet

- `open(method, url, async [, user, password])`
 - ▣ Prépare une requête en indiquant
 - La méthode (GET ou POST), l'URL, le drapeau de synchronisation
 - Et éventuellement le nom d'utilisateur et le mot de passe
- `send (contenu)`
 - ▣ Effectue la requête, éventuellement en envoyant les données
- `setRequestHeader("champ","valeur")`
 - ▣ Assigne une valeur à un champ d'entête HTTP qui sera envoyé lors de la requête

Méthodes de l'objet

- abort()
 - ▣ Abandonne la requête

- getAllResponseHeaders()
 - ▣ Renvoie l'ensemble de l'entête de la réponse sous forme de chaîne de caractères

- getResponseHeader("champEntete")
 - ▣ Renvoie la valeur d'un champ d'entête HTTP

Propriétés de l'objet

- `responseText` / `responseXML`
 - ▣ Réponse sous forme de chaîne de caractères / objet DOM

- `status` / `statusText` :
 - ▣ Code numérique de réponse du serveur HTTP
 - ▣ À tester pour s'assurer que tout est bon
 - 200 : OK
 - 404 : page introuvable
 - ...
 - ▣ message accompagnant le code de réponse :
 - 404 : Not Found...

Exemple simple

```
<html>
<body>
<script language="javascript">
function getXMLHttpRequest() {...}

function ajax() {
    var xhr=getXMLHttpRequest();
    xhr.open("GET", "test.html", false);
    xhr.send(null);
    alert(xhr.responseText);
}
</script>
<a href="javascript:ajax();">Cliquez-moi !</a>
</body>
</html>
```

Synchrone ou asynchrone

□ Requête synchrone :

- ▣ Tout est bloqué en attendant la réponse
 - Pas ergonomique
- ▣ C'est l'approche la plus simple
- ▣ Les réponses arrivent forcément dans l'ordre

□ Requête asynchrone :

- ▣ Le navigateur continue à répondre aux événements en attendant la réponse
- ▣ Plusieurs requêtes envoyées en même temps, les réponses peuvent arriver dans le désordre

Javascript Asynchrone

- Le choix entre synchrone et asynchrone se fait dans l'appel à XMLHttpRequest (méthode open) :
 - ▣ true pour asynchrone
 - ▣ false pour synchrone
- Dans le cas d'un appel asynchrone, le résultat est récupéré par une fonction :
 - ▣ `xhr.onreadystatechange = function() { ...};`

Autres propriétés de l'objet

- `readyState` :

- État de l'objet :

- 0 : non initialisé
 - 1 : ouverture = méthode `open()` appelée avec succès
 - 2 : envoyé = méthode `send()` appelée avec succès
 - 3 : en train de recevoir = données en cours de transfert
 - 4 : terminé = données chargées

- `onreadystatechange` :

- Fonction appelée à chaque changement d'état

Pour résumer

- Deux méthodes principales :
 - ▣ open : pour établir une connexion.
 - ▣ send : pour envoyer une requête au serveur.
- Récupération des données :
 - ▣ Champs responseXml ou responseText.
- Créer un nouvel objet XmlHttpRequest, pour chaque fichier à charger.

Un exemple

```
function ajax() {  
    var xhr=getXMLHttpRequest();  
    xhr.onreadystatechange = function() {  
        if(xhr.readyState == 4) {  
            if(xhr.status == 200)  
                alert("Received : " + xhr.responseText);  
            else  
                alert("Error code : " + xhr.status);  
        }  
    };  
    xhr.open("GET", "test2.html", true);  
    xhr.send(null);  
}
```

```
<body>  
<a href="javascript:ajax();">Cliquez-moi !</a>  
</body>
```

HTTP GET ou POST

- GET pour récupérer des données
 - Les informations sont passées dans l'url
 - `www.google.fr/index.php?q=3&req=fdg`
 - Les requêtes GET doivent pouvoir être bookmarkées ou mises en cache
 - Elle ne doivent donc pas provoquer de mises à jour sur le serveur
- POST pour envoyer des données
 - Pour tout ce qui ne correspond pas à un GET

AJAX : X = XML

- Le serveur peut renvoyer des données XML
 - ▣ responseXML contient la réponse dans ce format
 - ▣ La méthode javascript getElementByTagName(nom) d'un objet permet de cibler un élément

```
var docXML= xhr.responseXML;
var items = docXML.getElementsByTagName("donnee");
for (i=0;i<items.length;i++) {
    alert (items.item(i).firstChild.data);
}
```

Attention !

- Les requêtes AJAX asynchrones passent par Internet
 - ▣ Aucune garantie que les paquets arrivent dans l'ordre.
 - ▣ Aucune garantie qu'une requête soit terminée avant qu'une autre ne soit lancée :
 - Les délais peuvent varier énormément à cause de la charge du serveur ou du réseau
- Nécessite de faire très attention à ce qu'on fait :
 - ▣ Cf exemple de recherche google, on ne veut pas afficher pour "a" après celles pour "ab"

Inconvénients

- JavaScript doit être activé
- Les données chargées de façon dynamique ne font pas vraiment partie de la page. Prise en compte par les moteurs de recherche pas claire
- Asynchrone => affichage avec délai, peut poser problème à l'utilisateur
- Le bouton « Page précédente » ne marche pas en général :
 - ▣ on peut recharger une page mais plus difficilement annuler des modifications faites par Javascript

Conclusions sur Ajax

- ❑ Combinaison de langages standards du WEB (Javascript, DOM HTML, XML) grâce à l'objet XMLHttpRequest
- ❑ WEB dynamique « coté client »
- ❑ Utilisé par tous les sites « WEB 2.0 »
- ❑ Un outil à utiliser en attendant le déploiement du HTML5 (prévu pour 2010)

Optimisation

□ Librairies :

- ▣ Utiliser des outils pour combiner plusieurs fichiers Javascript ou CSS afin de créer un seul gros fichier pour diminuer la charge sur le serveur
 - Une seule requête
 - Le fichier peut être mis en cache

□ Eviter les appels multiples :

- ▣ Mieux vaut faire une seule grosse requête au début que plein de petites requêtes.

JSON JAVASCRIPT OBJECT NOTATION

JSON ?

- Format d'échange de données.
- Objectifs :
 - ▣ Simple.
 - ▣ Extensible.
 - ▣ Ouvert.
 - ▣ Lisible par un humain.
- Similaire à
 - ▣ la définition des objets Javascript
 - ▣ Les tableaux associatifs javascript

JSON : JavaScript Object Notation

- Les types de base
 - ▣ Nombres entiers, réels ou à virgule flottante
 - ▣ Chaînes de caractères
 - ▣ Booléen true et false
 - ▣ Tableaux [..., ...] ou tableaux associatifs (objets) "clé":valeur : {..., ...}
 - ▣ null

```
{ "Nom": "Guillaume",  
  "Age": 33,  
  "Adresse": { "rue": "104 avenue Kennedy",  
                "cp": "75016", "ville": "Paris" },  
  "notes": [1, 2, 4, 8, 16, 32]  
}
```

JSON et Javascript

- JSON peut être utilisé directement :
 - ▣ Inclusion dans du HTML
 - `<script ...> var data = JSONdata; </script>`
 - ▣ Peut être converti en un objet Javascript
 - `responseData = eval('(' + responseText + ')');`

JSON ou XML ?

- JSON est très utilisé avec AJAX (le X de AJAX est pour XML)
- JSON :
 - ▣ {"nom": "Guillaume", "prenom": "Jean-Loup"}
- XML :
 - ▣ <?xml version='1.0' encoding='UTF-8'?>
 - ▣ <element>
 - ▣ <nom>Guillaume</nom>
 - ▣ <prenom>Jean-Loup</prenom>
 - ▣ </element>

JSON ou XML ?

- Evaluation en JSON :

- ▣ `var name = eval('(' + req.responseText + ')').nom.value;`

- Ou :

- ▣ `eval('(' + req.responseText + ')').xyz.abc.value;`

- ▣ Accès simplifié aux différent niveaux.

JSON ou XML ?

- Evaluation en XML :
 - ▣ `var root = req.responseXML;`
 - ▣ `var name = root.getElementsByTagName('nom');`

- Ou :
 - ▣ `root.getElementsByTagName('xyz')[0].firstChild`

- Un peu plus complexe

JSON ou XML

- Taille des données :
 - ▣ plus petite en JSON (pas de fermeture de tag)
 - ▣ XML se compresse mieux
- Vitesse :
 - ▣ XML se parse mieux
 - ▣ JSON s'évalue avec eval : peu efficace (pour l'instant)
- Choix :
 - ▣ JSON : structures de données
 - ▣ XML : structuration de documents
- A vous de faire votre choix !

AJAX ou AJAJ : J = JSON

- XML pas évident à parser en Javascript
- On utilise plutôt JSON

```
{ "menu": {  
  "id": "file",  
  "value": "File",  
  "popup": {  
    "menuitem": [  
      { "value": "New", "onclick": "CreateNewDoc()" },  
      { "value": "Open", "onclick": "OpenDoc()" },  
      { "value": "Close", "onclick": "CloseDoc()" }  
    ]  
  }  
}}
```

Exemple d'utilisation de JSON

- Coté client :
 - ▣ JSON inclus dans JavaScript.
 - ▣ Le contenu est assigné à une variable et devient un objet.

```
// Création de la connexion :  
var req = new XMLHttpRequest();  
req.open("GET", "fichier.json", true);  
req.onreadystatechange = function() {  
    if (req.readyState == 4) {  
        var doc = eval('(' + req.responseText + ')');  
    }  
}  
req.send(null);
```

Exemple d'utilisation de JSON

- Côté serveur :
 - ▣ Parseurs pour générer du JSON à partir d'objets en Php ou JAVA
- L'échange de données :
 - ▣ Envoi avec AJAX

```
<?php
    $arr = array ('a'=>1, 'b'=>2, 'c'=>3, 'd'=>4, 'e'=>5);
    echo json_encode($arr);
?>
```

Affiche :

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

JAVASCRIPT OPTIMISATION DE CODE



Un navigateur

- Fonctionne mais :
 - ▣ Problèmes de sécurité
 - ▣ Problèmes de performance :
 - Ouvrir 20 onglets sous Firefox pose quelques problèmes
 - ▣ Pas prévu spécifiquement pour Ajax
 - Très gourmand en ressources
 - Modification du DOM et autres
- Autant aider le navigateur en codant proprement

Du code correct avant tout !

- ❑ Ne pas forcément optimiser avant que ça ne marche :
 - ▣ Si ça ne fonctionne pas, on se moque que ça aille vite
 - ▣ Mais y penser avoir d'avoir 10k lignes de code
- ❑ Tester dans des situations réelles :
 - ▣ Réseau, client ou serveur lent (tester en local peut masquer de nombreux problèmes de performance)
 - ▣ Utilisation de Yslow avec Firebug

Optimisation

- Approches complémentaires :
 - ▣ Suivre des cours d'algorithmique
 - ▣ Comprendre que Javascript est un langage interprété :
 - pas de compilateur pour optimiser le code !
 - ▣ Tester différentes façons de faire la même chose

Algorithmique - exemple

- Calcul de Fibonacci en récursif :
- Exemple : calcul de fibonacci(30)
 - ▣ Retourne 832 040
 - ▣ Combien d'appels à la fonction fibonacci ?

```
var fibonacci = function (n) {  
  return n < 2 ? n :  
    fibonacci(n - 1) + fibonacci(n - 2);  
};
```

Algorithmique - exemple

- Nombre d'appels : 2 692 537
- Pourquoi :
 - ▣ on recalcule de nombreuses fois les mêmes chose
 - ▣ autant tout stocker en mémoire et ne pas refaire les calculs

```
function fibonacci(n) {  
  var fibo_tab = [0,1];  
  var fibonacci_aux = function(n) {  
    var result = fibo_tab[n];  
    if (typeof result !== 'number') {  
      result = fibonacci_aux(n-1)+fibonacci_aux(n-2);  
      fibo_tab[n] = result;  
    }  
    return result;  
  };  
  return fibonacci_aux(n);  
}
```

Langage interprété

```
var i;  
for (i = 0; i < divs.length; i += 1) {  
    divs[i].style.color = "black";  
    divs[i].style.border = thickness + "px solid blue";  
    divs[i].style.backgroundColor = "white";  
}
```

□ Que peut-on améliorer ?

Langage interprété

- Les choses "couteuses" :
 - ▣ Calcul de la longueur du tableau
 - ▣ Recherche de `divs[i].style`
 - ▣ Concaténation de chaîne

```
var border = thickness + 'px solid blue',
    nrDivs = divs.length,
    ds, i;
for (i = 0; i < nrDivs; i += 1) {
    ds = divs[i].style;
    ds.color = "black";
    ds.border = border;
    ds.backgroundColor = "white";
}
```

Langage interprété

- La plupart des compilateurs font (par défaut) :
 - ▣ La suppression de sous-expression communes
 - ▣ La suppression des invariants de boucle
 - ▣ ...

- Mais pas Javascript !
 - ▣ Ne pas trafiquer le code à outrance non plus

Travailler sur des chaînes

- Qu'est ce qui est le plus rapide ?
 - ▣ `str = 'test'; str += 'test';`
 - ▣ `str = 'test' + 'test';`

- Répété 1 million de fois :
 - ▣ `str = 'test'; str += 'test';` : 432 ms
 - ▣ `str = 'test' + 'test';` : 70 ms
 - ▣ Résultat pas forcément intuitif...

Optimisation

- Ce qui est couteux :
 - ▣ Modification du DOM (Document Object Model)
 - ▣ InnerHTML en particulier est très couteux
 - Éviter de faire des `obj.innerHTML += chaine`
 - Plutôt construire le nouveau contenu et l'insérer en une fois

Optimiser ou ne pas optimiser ?

- Certains navigateurs ont des fonctions mal implémentées.
 - ▣ Deux versions d'une même navigateur peuvent présenter des caractéristiques très différentes.
 - ▣ Eviter les astuces qui peuvent fonctionner sur un navigateur mais pas sur un autre
- Plutôt penser à long terme.

Optimiser en connaissance

- Mesurer avec précision :
 - ▣ `debut = new Date().valueOf();`
 - ▣ `fonction_a_evaluer ();`
 - ▣ `fin = new Date().valueOf();`
 - ▣ `duree = end_time - start_time;`

- Attention :
 - ▣ Un seul essai ne veut rien dire en général.
 - ▣ Tester avec différents paramètres (complexité)

A faire en TME / toujours

- ❑ Vérifier vos fonctions récursives.
- ❑ Ecrire des fonctions pour faire du profiling de code.
- ❑ Tester plusieurs approches.