JavaScript

M4103C - Programmation Web – client riche

2ème année - S4, cours - 3/4
2021-2022

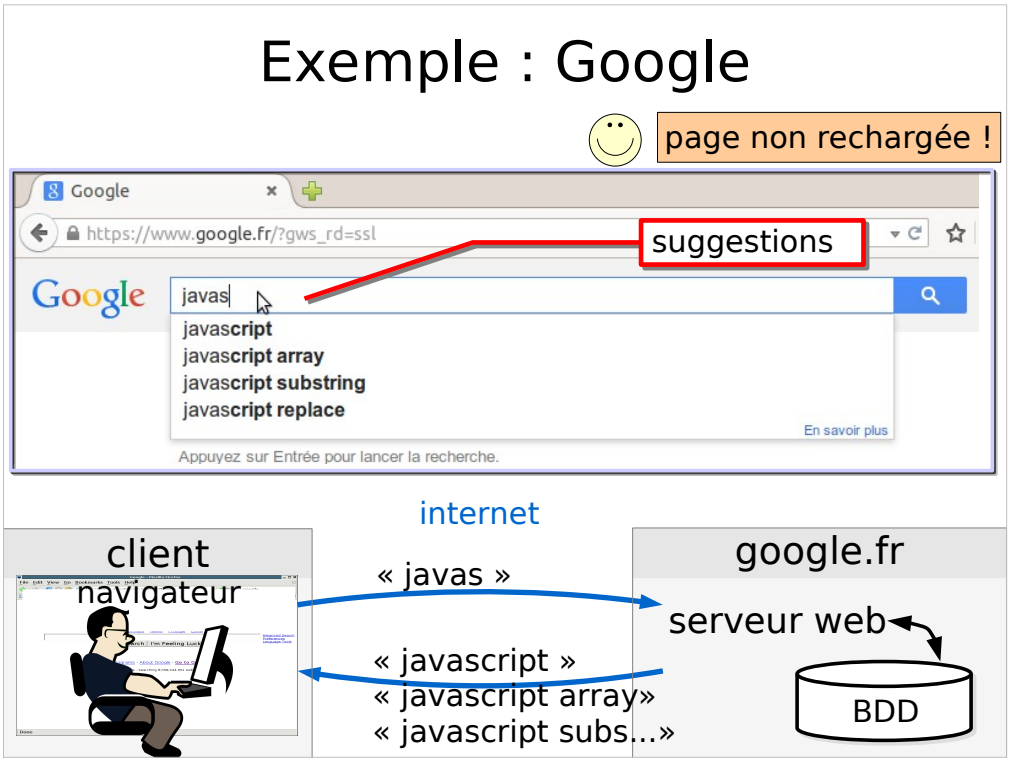
bosc@univ-paris13.fr

Département informatique IUT de Villetaneuse Université Paris-13

Table des matières

1ère partie

Ajax



Quand un utilisateur commence à taper une recherche dans Google, Google lui propose une liste de suggestions.

Il est impossible d'inclure dans la page d'origine toutes les suggestions possibles. Le navigateur va donc demander au serveur la liste de suggestions au fur et à mesure que l'utilisateur écrit. Ceci se fait sans recharger la page.

Dans ce qu'on a vu jusqu'à maintenant, toutes les interactions avec le serveur se faisaient au moment du chargement de la page.

Ici, la liste des suggestions est cherchée à partir du serveur, à partir du JS et après la fin du chargement de la page.
[lp1200]

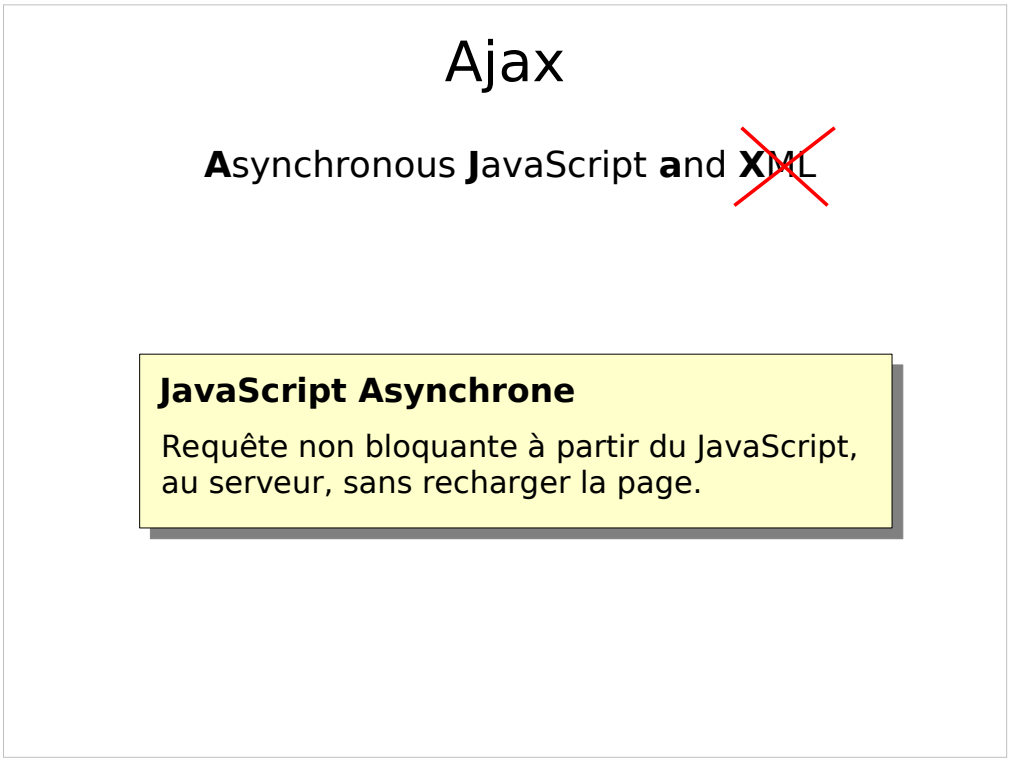
Dans ce deuxième exemple, l'utilisateur appuie sur un bouton « J'aime » dans les commentaires d'une vidéo sur Youtube. Son action est enregistrée sur le serveur, mais la page ne se recharge pas.

Le fonctionnement habituel en HTML du bouton consiste à envoyer une requête POST et afficher une nouvelle page. Le rechargement de la page est désagréable pour l'utilisateur. C'est lent, la vidéo s'interrompt, il perd sa position dans la page...

Ici, la requête sur le serveur se fait sans rechargement de la page.
[lp1199]

Sur la même page Youtube, l'affichage des commentaires se fait après la fin de de l'affichage de la page. A la place des commentaires, un message indique pendant un bref instant « chargement en cours... ».

Par ailleurs, l'utilisateur peut trier les commentaires, sans recharger la page. Une vidéo peut avoir des milliers de commentaires. Ils ne peuvent être chargés qu'à la demande.
[lp1198]

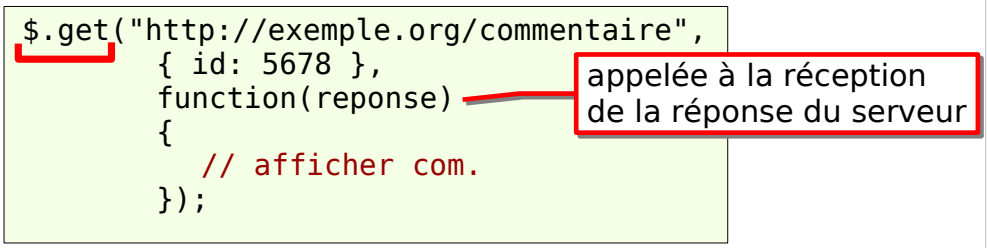


AJAX est une approche dans laquelle on communique avec le serveur, à partir du JavaScript, sans recharger la page.

La requête au serveur peut prendre du temps. Il serait très désagréable que le navigateur soit bloqué pendant cette attente. La requête est donc faite de manière "asynchrone" : on lance la requête en JS et on fourni une fonction qui sera appelée plus tard, lorsque que la requête sera finie.

Le "X" de AJAX vient de "XML". Historiquement la communication entre le serveur et le client se faisait en XML ... aujourd'hui on préfère un autre format appelé JSON.
[lp1197]

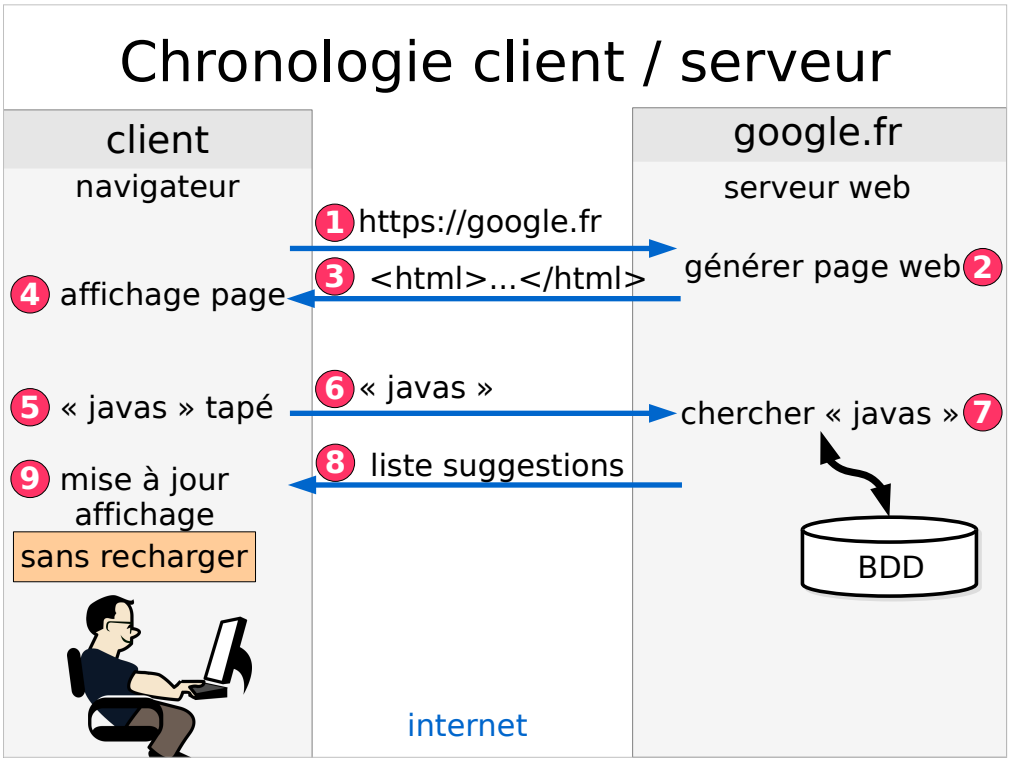
Exemple jQuery .get()



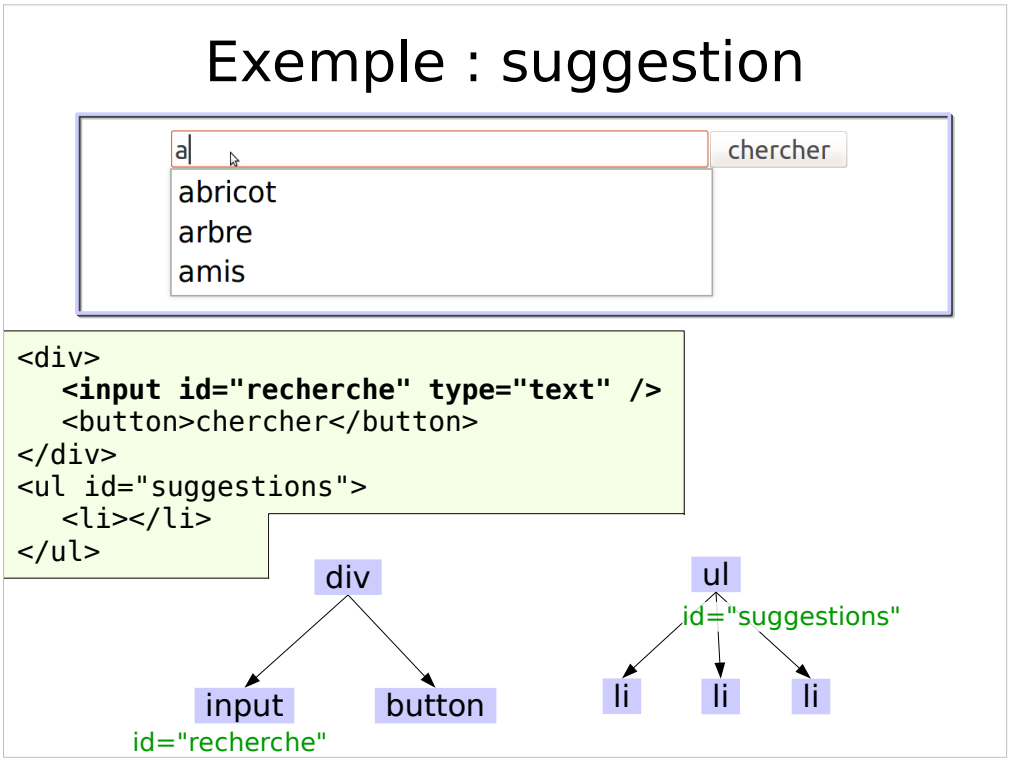
La fonction jQuery .get() permet de faire une requête AJAX en utilisant la méthode "GET".

Remarquez la syntaxe « $.get » : c'est une fonction qui est appellée directement sur l'objet jQuery ($) et pas sur une liste d'éléments jQuery.

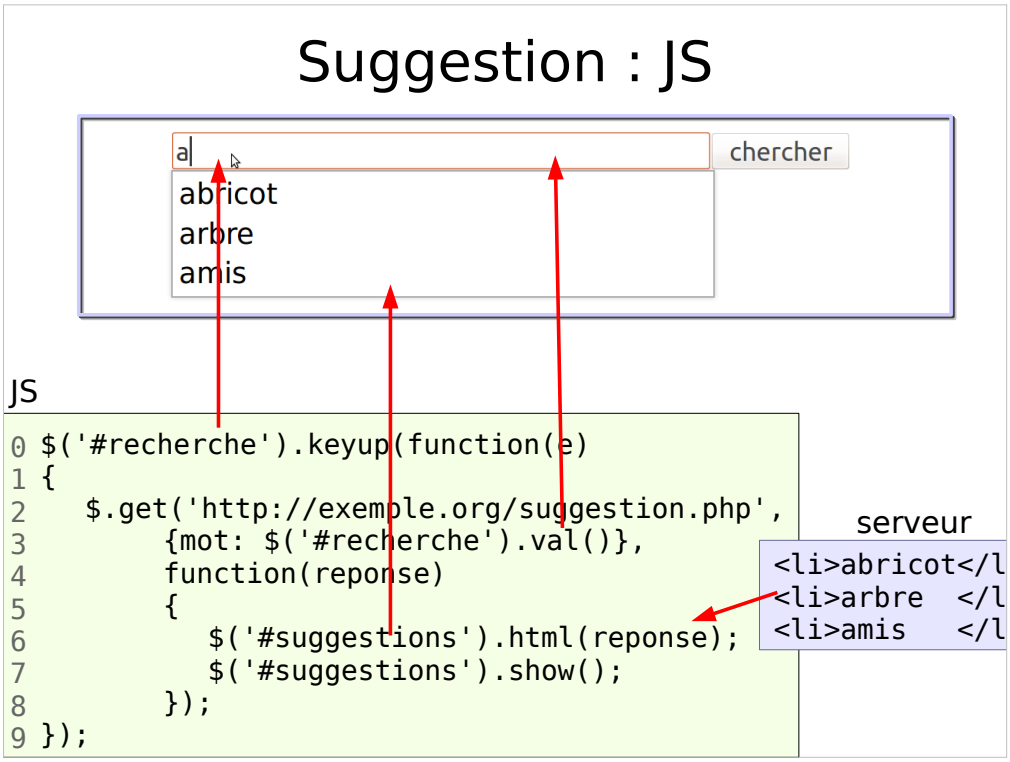
Ici, $.get prend 3 arguments :
1) l'URL vers lequel on fait la requête
2) des données GET à envoyer au serveur
3) la fonction qui sera appelée quand la requête aura réussi
Important: cette fonction (3) n'est pas appelée tout de suite, lors de l'appel de $.get ... mais plus tard, en cas de succès de la requête
[lp1196]



Reprenons l'exemple des suggestions dans la recherche Google. Il est très important de bien comprendre la chronologie et de savoir ce qui se passe sur le client et sur le serveur. En pratique, les échanges peuvent devenir compliqués.
1) Client au serveur « bonjour google.fr, je veux la page https://google.fr »
2) Le serveur de Google exécute un programme qui génère la page HTML
3) Serveur au Client : voici la page au format HTML
4) Le client affiche la page reçue (simplifié, en pratique il y a de nombreuses autres requêtes pour le CSS, le JS, les images...)
5) L'utilisateur commence à taper « javas ». Un programme JavaScript sur le client réagit à l'événement clavier et lance une requête AJAX au serveur.
6) Requête AJAX au serveur
7) Le serveur reçoit la requête et exécute un programme (par exemple PHP) qui va chercher des suggestions commençant par « javas » dans une Base de Données (ex: sql).
8) Le serveur envoie les suggestions au client
9) Le JavaScript sur le client affiche les suggestions
[lp1195]



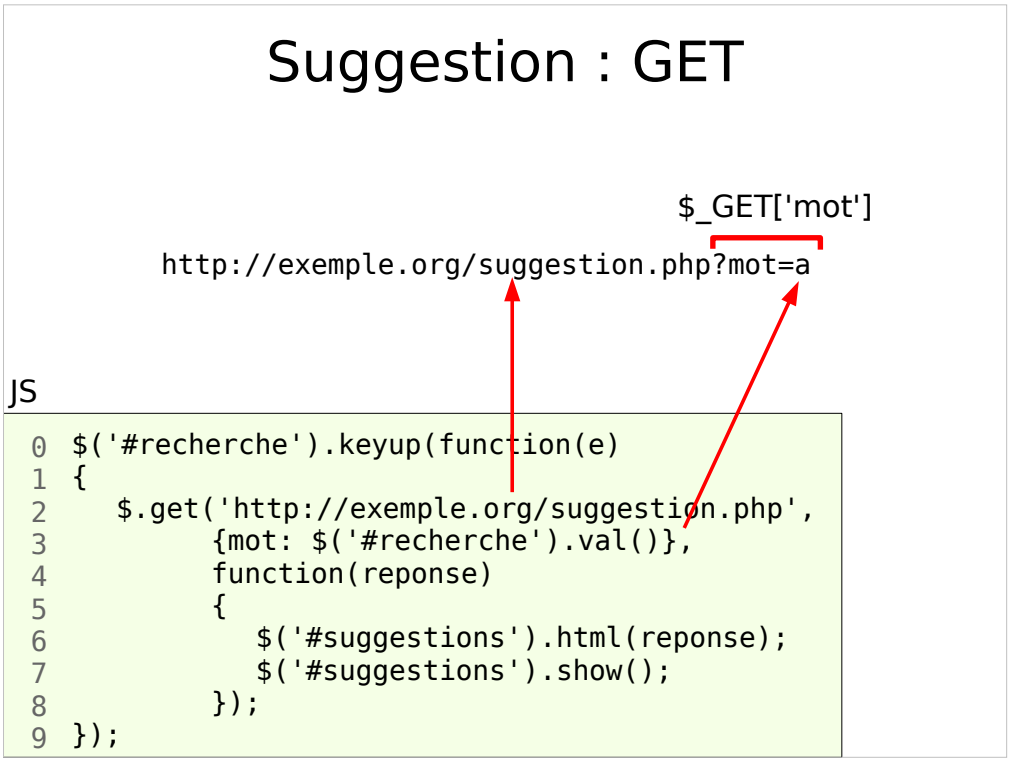
On va voir un exemple similaire aux suggestions Google, très simplifié.
Le HTML est très simple:
- un champs texte <input> pour que l'utilisateur puisse taper sa recherche.
- en dessous, une liste <ul> / pour afficher les suggestions.
[lp1194]



Le début du JS est classique :
$('#recherche') est une liste jQuery contenant uniquement le champs texte. On écoute les événements du clavier dans ce champs texte grâce à « .keyup() ».

Quand l'utilisateur relâche une touche (keyup), la fonction $.get est appelée.
Sont fournis en argument : L'URL, les données pour le serveur et la fonction à appeler après la requête.
Quand la requête réussi, la fonction (ligne 4) est appelée avec en argument (reponse) le HTML renvoyé par le serveur.
Cet HTML est affiché dans
<ul id="suggestions">
Comme la liste est peut-être caché, il faut l'afficher.

Il manque dans cet exemple le code pour remplir le champs texte quand on clique sur un élément de la liste.
[lp1193]

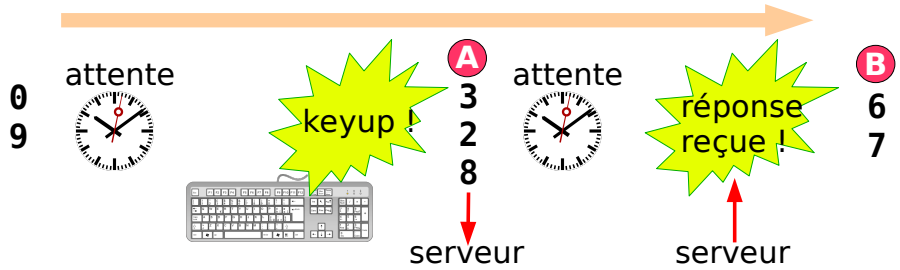


Le 2e argument indique les données fournies au serveur avec la méthode GET.

Dans la méthode GET, les données sont inclues dans l'URL en utilisant le séparateur "?" puis ensuite les séparateurs "&".

Remarque: on aurait pu mettre ces informations directement dans l'URL de l'argument 1... mais il faudrait les encoder correctement (espaces et autres caractères spéciaux). jQuery le gère plus simplement.
[lp1192]

Suggestion : chronologie JS

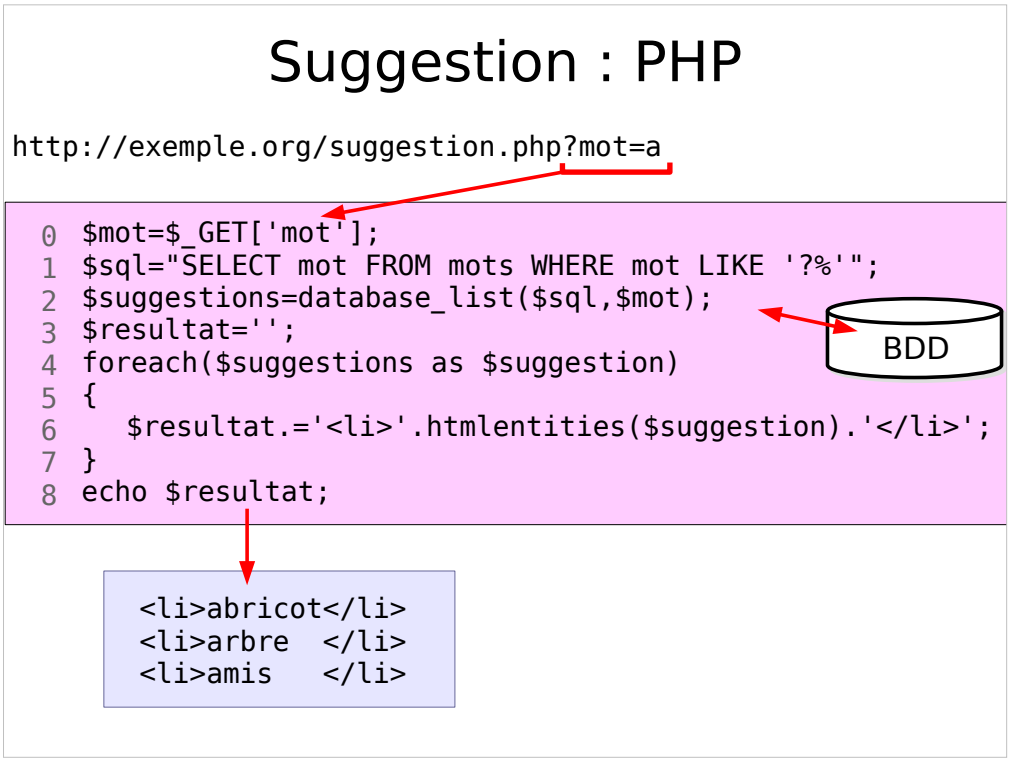


JS

```
0  $('#recherche').keyup(function(e)
1  {                          A
2     $.get('http://exemple.org/suggestion.php',
3           {mot: $('#recherche').val()},
4           function(reponse)
5           {   B
6              $('#suggestions').html(reponse);
7              $('#suggestions').show();
8           });
9  });
```

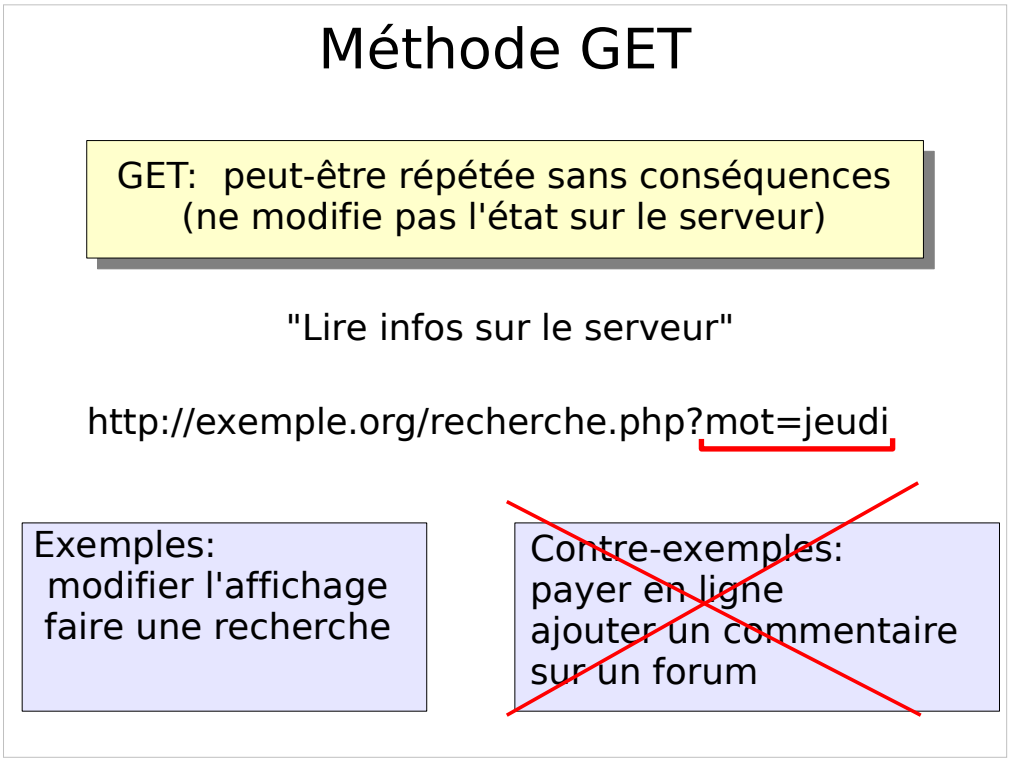
En JS, les fonctions anonymes permettent d'écrire très simplement du code événementiel et différé. Cette simplicité peut nous faire oublier la chronologie compliquée de l'exécution de ce programme.

[lp1191]



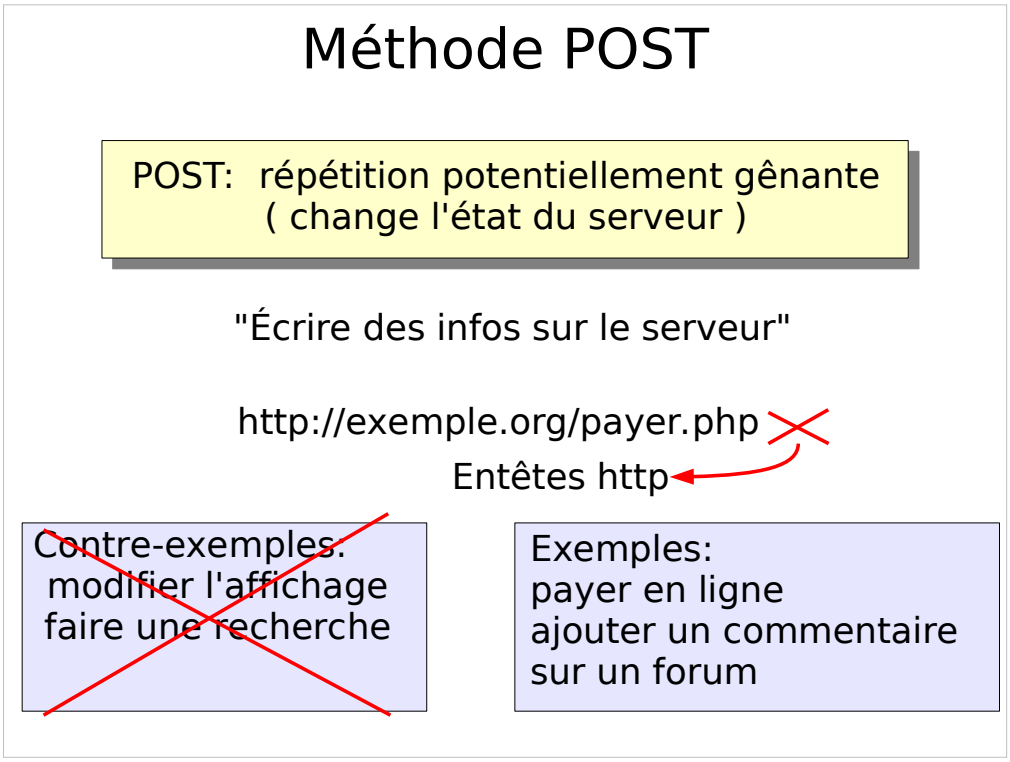
Dans le PHP, les arguments GET sont récupérés dans le tableau global $_GET.
Dans cet exemple, le programme fait une requête SQL à la base de données pour chercher tous les mots qui commencent pas "a".

On construit alors le HTML contenant les suggestions (<li>......) .
Cet HTML est envoyé au client tout simplement avec "echo"...
[lp1190]



Chaque fois qu'on écrit une requête AJAX, comme chaque fois qu'on écrit un formulaire, il faut choisir entre la méthode GET et la méthode POST.
La principale motivation de ce choix est définie par les normes :
les requêtes GET sont "idempotentes", elles peuvent être répétées sans conséquences.

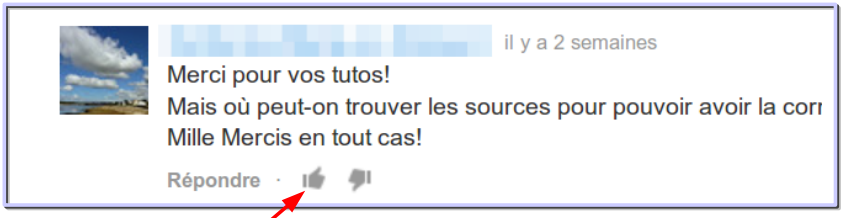
Les données des requêtes GET sont transmises dans l'URL. Ceci a conduit de nombreuses personnes à penser, à tord, que le choix entre GET et POST était une question de sécurité.
[lp1189]



Le requêtes POST sont choisies pour des actions qui ne peuvent pas être répétées sans conséquences.

Les données sont transmises dans les entêtes de la requête et pas dans l'URL. Elles ne sont pas visibles dans l'URL, mais elles ne sont pas pour autant "cryptés". L'argument de sécurité est souvent mal compris.
[lp1188]

Exemple : .post()



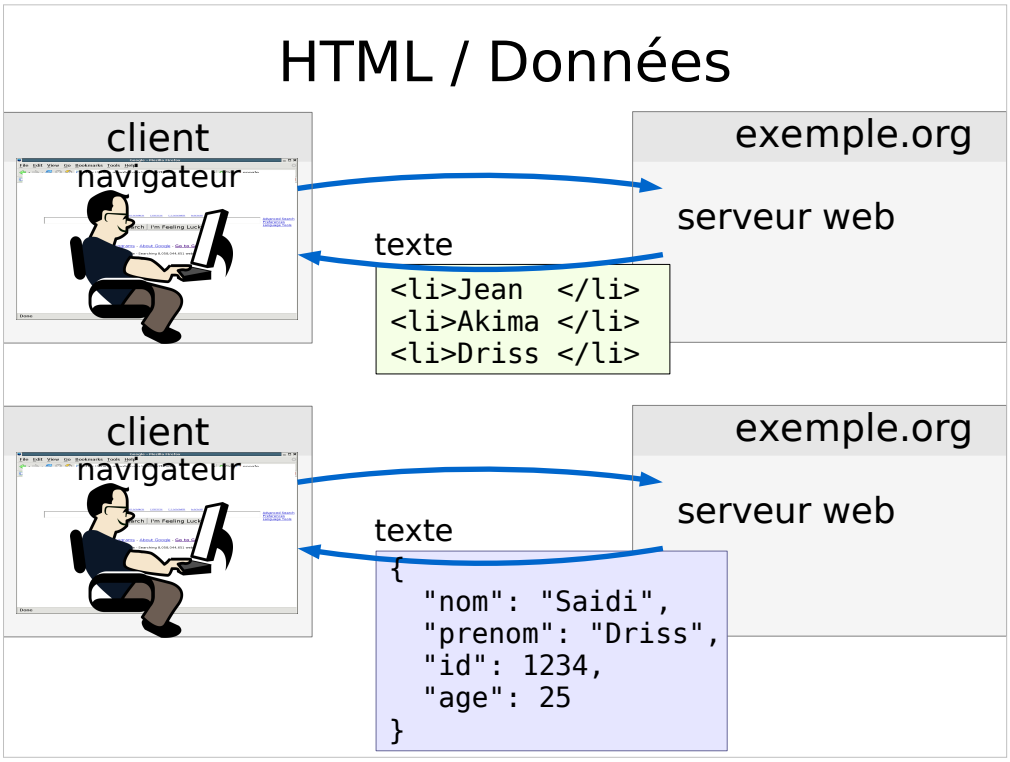
JS

```
0 $('.jaime').click(function(e)
1 {
2    $.post('http://exemple.org/jaime.php',
3           {nbCom: $(this).parent().attr('id')},
4           function(reponse)
5           {
6               ...
7           });
8 });
9
```

Dans cet exemple, l'utilisateur clique sur le bouton "J'aime" en dessous d'un commentaire. Cette action va modifier l'état du serveur (+1 à la valeur j'aime sur ce commentaire dans la base de données). Il faut donc utiliser POST.
$.post() s'utilise de la même manière que $.get()
[lp1187]

2ème partie

JSON



Dans les exemples vus précédemment, le serveur renvoyait un fragment de HTML qui était ensuite affiché.
Bien souvent, on veut chercher des données complexes sur le serveur, et pas juste du HTML.

On pourrait utiliser n'importe quel format pour ces données. Il suffit juste que le programme sur le serveur et le JavaScript soient d'accord.
Le XML a été beaucoup utilisé, mais est lourd à manipuler.
Pour les tâches simples, du texte brut est possible.
Pour les informations plus structurées on utilise souvent, aujourd'hui, un format appelé JSON.
[lp1185]

JSON

JavaScript Object Notation

JSON

Format de fichier texte, utilisant la syntaxe JavaScript pour représenter des données (objets, tableaux ...)

très utilisé !

beaucoup de langages PHP

Le JSON est format texte qui est presqu'identique au format utilisé en JavaScript pour déclarer des Objets ou des Tableaux.

Tous les principaux langages fournissent des méthodes pour encoder et décoder le JSON.
[lp1184]

JSON : exemples

Objet simple

```
{
  "nom": "Saidi",
  "prenom": "Driss",
  "id": 1234,
  "age": 25
}
```

Objet complexe

```
{
   "nom": "Collège Grange Bois",
   "ville":
   {
      "nom": "Savigny-le-Temple",
      "nom-court": "Savigny",
      "code": 77176
   },
   "adresse": "2 av. Victor..."
}
```

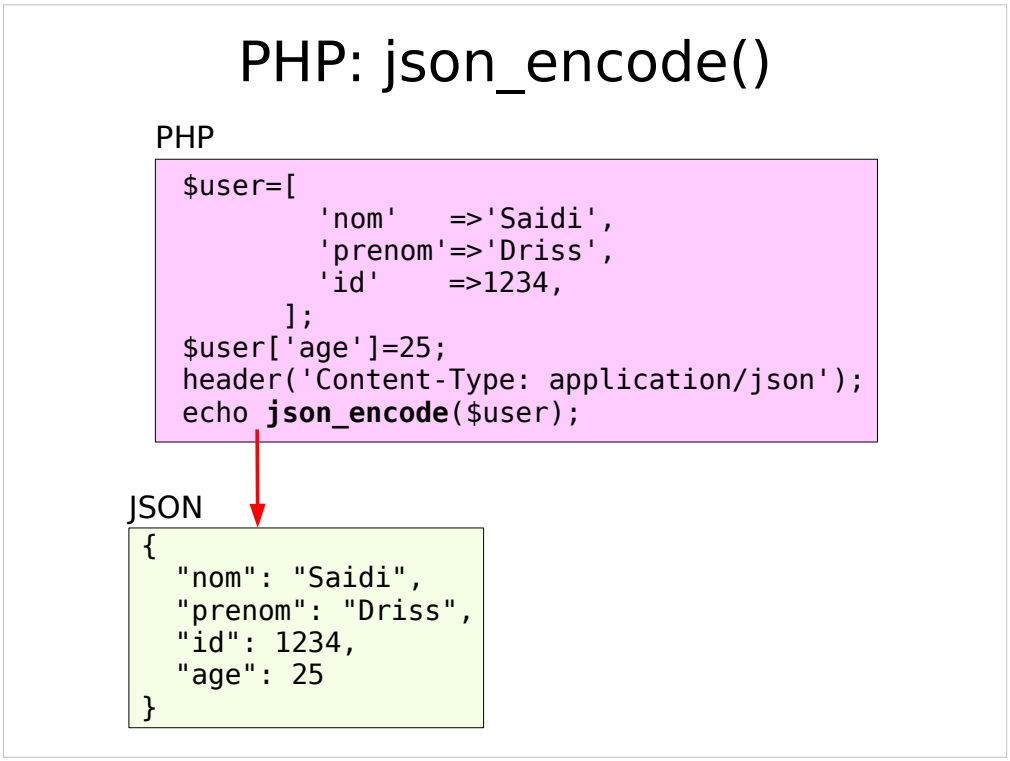
Tableau simple

```
[
   "Fraise",
   "Chocolat",
   "vanille"
]
```

Tableau d'objets

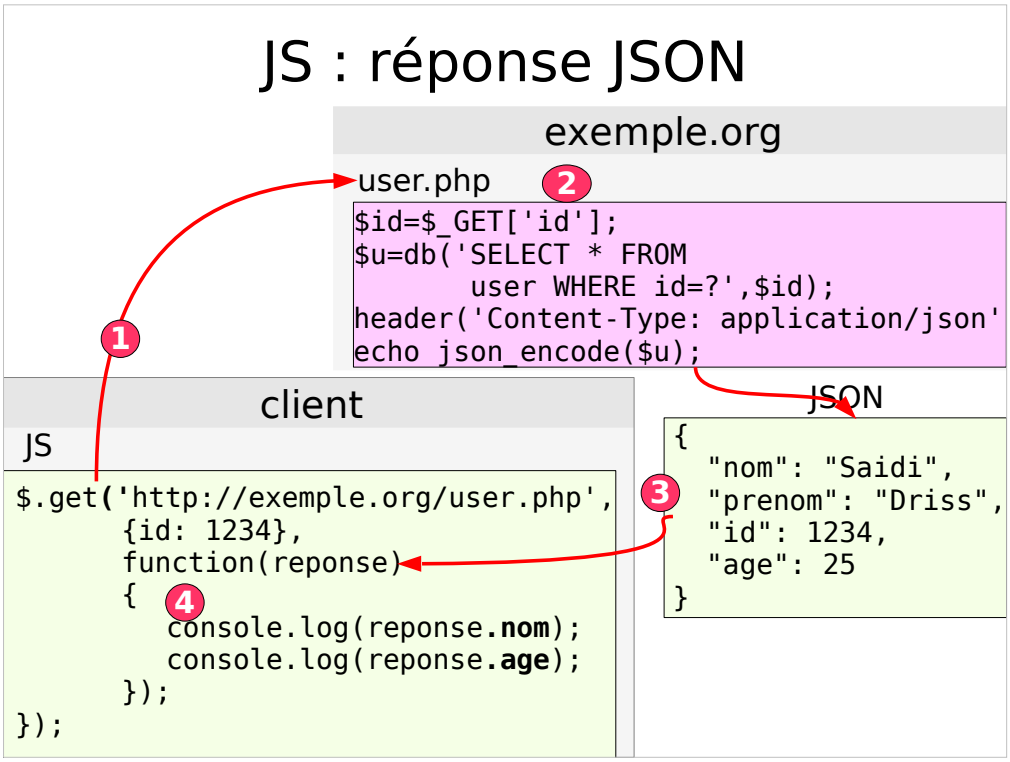
```
[
   { "nom": "Wang",
      "id": 4321   },
   { "nom": "Amara",
      "id": 5612   }
]
```

Voici quelques exemples.
En pratique, les données JSON peuvent être volumineuses et complexes
[lp1183]



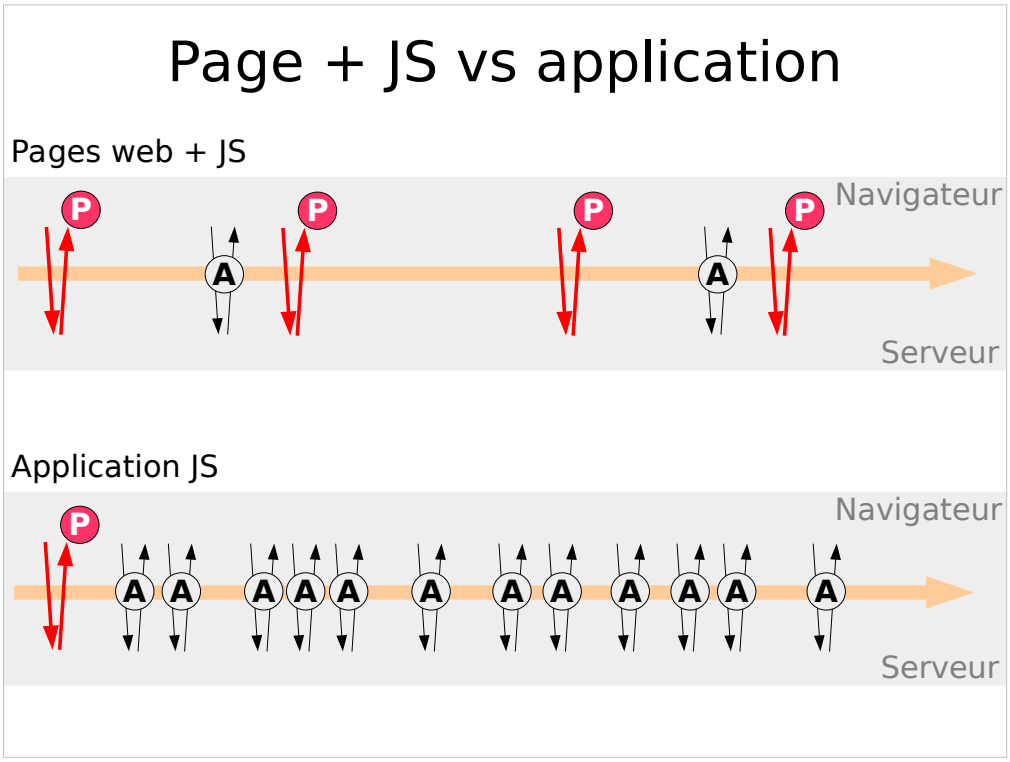
En PHP la fonction json_encode() permet de transformer très simplement un tableau (ou un objet) en JSON. Ce JSON est ensuite envoyé au client, qui pourra le décoder simplement en un objet (ou tableau) JavaScript.

Remarquez la fonction "header()".
Toute réponse HTTP contient des données appelées "entêtes". Attention: il ne s'agit pas des entêtes du HTML, mais des informations, cachées, en dehors de tout code HTML.
Dans ces entêtes HTTP, on doit préciser le type de contenu transmis:
Pour du HTML: Content-type: text/html
Pour du JSON : Content-type: application/json
[lp1182]



Voici un exemple simplifié.
Le client demande des données sur l'utilisateur numéro 1234.
Le PHP cherche ces données dans la BDD, les récupère dans un tableau associatif et transforme ce tableau en JSON.
Le JSON est renvoyé au client.
jQuery décode le JSON et met l'objet décodé en argument (reponse) de la fonction anonyme.

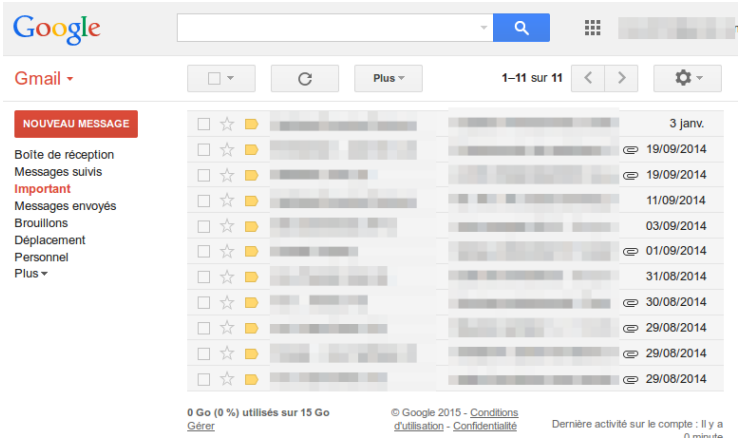
Remarquez que jQuery est capable de distinguer automatiquement une réponse HTML d'une réponse JSON grâce au Content-Type. Si Content-Type est text/html, reponse sera une chaîne de caractères contenant le HTML. Si c'est application/json, reponse est un objet JS.
[lp1181]



Dans l'approche classique, la navigation se fait en cliquant sur des liens ou en appuyant sur des boutons de formulaires. A chaque fois une nouvelle page est visitée. De temps en temps une interaction AJAX peut venir enrichir cette navigation.

À l'autre extrême, dans une application JS, une seule page est chargée et l'utilisateur ne la quitte pas. L'utilisateur interagît avec cette page de manière complexe. Ces interactions sont gérées en JS avec des appels AJAX fréquents.
[lp1180]

Application JS



Frameworks : Angular, React, VueJS ...

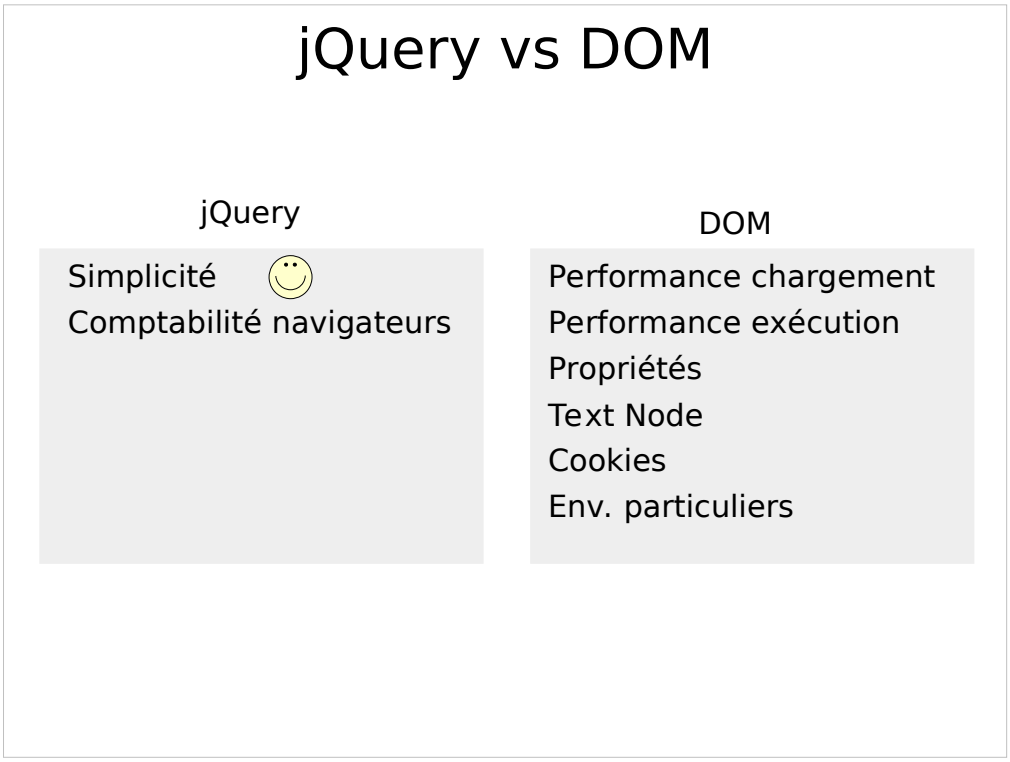
Un exemple d'application JS est gmail. L'utilisateur change de boite mail et ouvre chaque mail sans jamais quitter la page. Toutes ces interactions sont gérées en JS/AJAX.

Les applications JS sont difficiles à écrire. Le développeur doit gérer de nombreuses interactions complexes avec l'utilisateur et synchroniser les données avec le serveur.

Des Frameworks JS, comme AngularJS existent pour faciliter cette tâche. Ces techniques sortent du cadre de ce cours.
[lp1179]

3ème partie

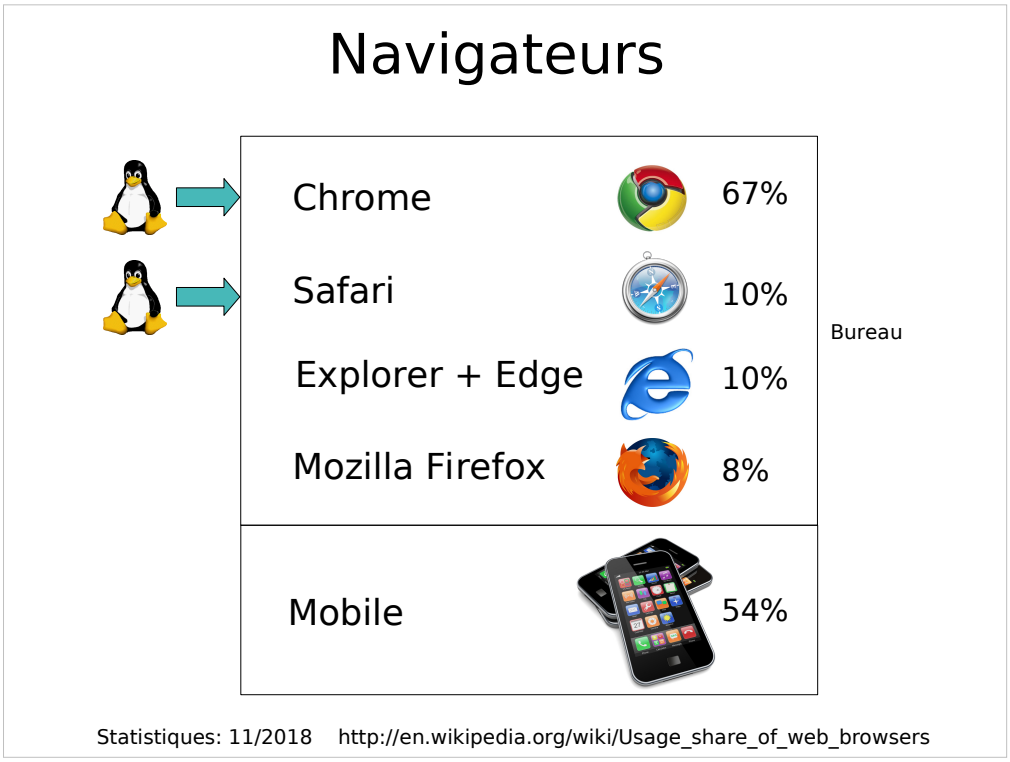
DOM



Jusqu'ici on a écrit nos programmes en utilisant presqu'exclusivement jQuery. jQuery est une surcouche par-dessus le DOM. Le DOM est l'API native permettant d'interagir avec le navigateur. jQuery n'est pas indispensable.

jQuery facilite énormément le travail. Non seulement le code est beaucoup plus simple et court à écrire, mais en plus, jQuery s'occupe des différences entre navigateurs (ie, Firefox, Chrome, ...).
Cependant, jQuery a un prix:
- le téléchargement de jQuery peut prendre quelque temps.
- le traitement / exécution de jQuery peut être long sur des machines peu puissantes (mobile 400ms!).

Même si on utilise jQuery, on a parfois besoin d'accéder à des propriétés ou des fonctions du DOM.
La manipulation de texte nécessite de manipuler les Text Node du DOM.
Dans des environnements où jQuery n'est pas disponible (ex: extension Firefox, page critique performance).
[lp1177]



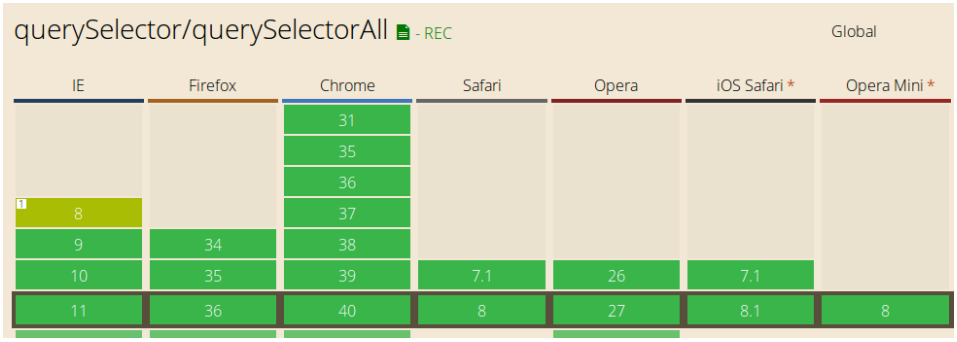
Le développement JS nécessite de vérifier en permanence le bon fonctionnement du code sur les principaux navigateurs du public visé. jQuery masque beaucoup de différences.

Le support des versions anciennes de ie ou certains mobiles peut-être une difficulté majeure qui peut modifier profondément des choix de développement.

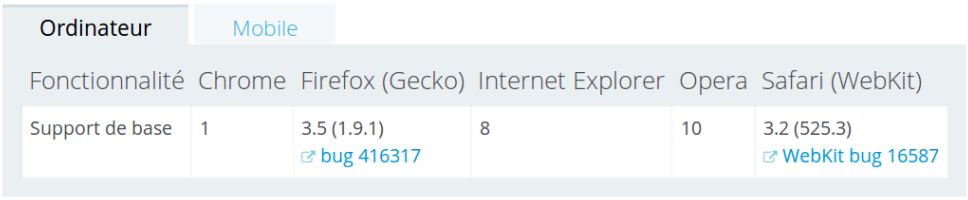
Des différences, moindres, mais significatives existent entre Chrome, Firefox, Safari et aussi entre entre différentes versions de chacun de ces navigateurs.
[lp1176]

Compatibilité

caniuse.com



MDN : developer.mozilla.org

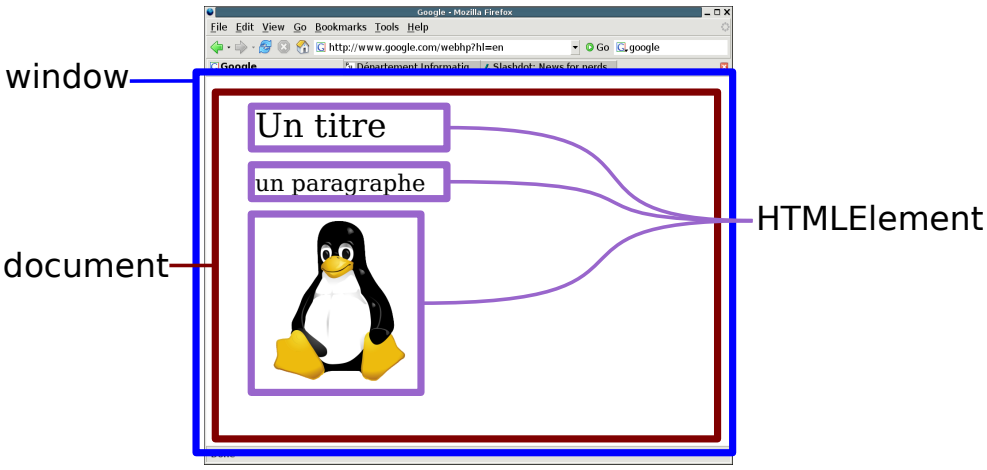


Des sites comme caniuse.com ou MDN fournissent des tables de compatibilité pour chaque technologie et pour chaque navigateur.

Il est indispensable de consulter ces tables quand on utilise une fonctionnalité (JS ou CSS), surtout si la fonctionnalité est récente.

[lp1175]

Principaux objets DOM



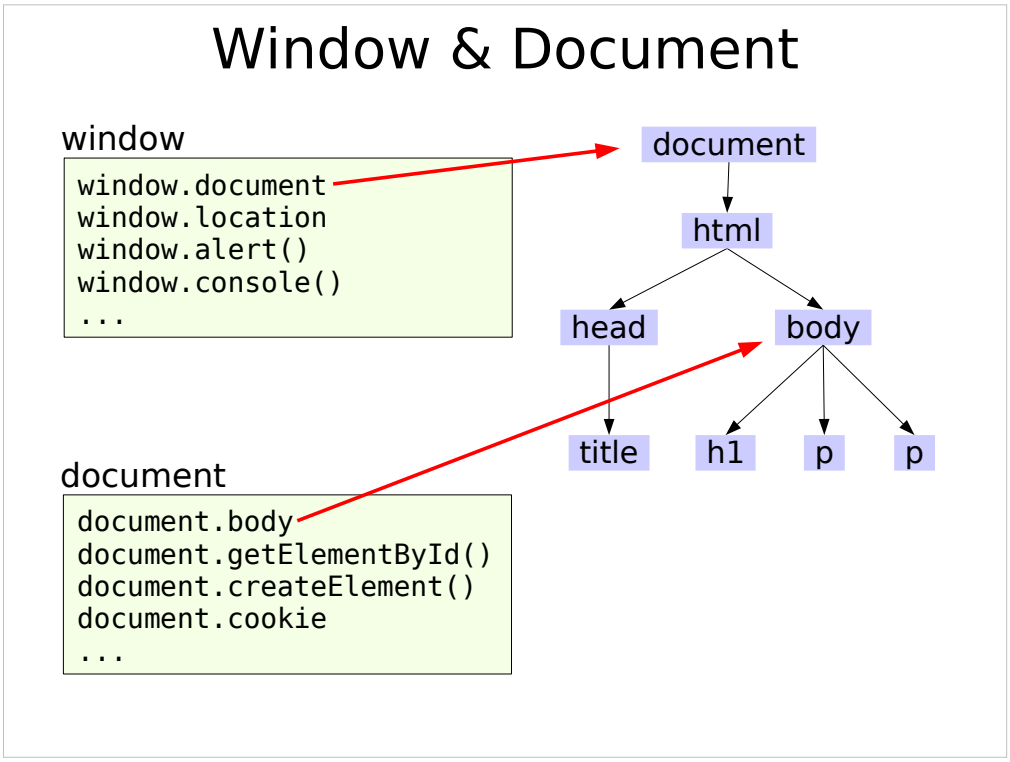
Voici 3 types d'objets DOM importants:

- **window** : la fenêtre d'un document. S'il y a plusieurs onglets, chaque onglet a son window.

- **document** : à l'intérieur du window, contient l'arbre DOM issu du HTML.

- **HTMLElement** : la plupart des noeuds de l'arbre qu'on manipulera sont de type HTMLElement

[lp1174]



L'objet "**window**" contient de nombreuses propriétés et méthodes. L'une d'entre-elles s'appelle "document" et donne accès à l'arbre DOM.

document est la racine de l'arbre DOM. Il ne figure pas dans le code HTML. Dans le DOM, il se trouve au-dessus de l'élément <html>.

document fourni aussi de nombreuses propriétés et méthodes. Certaines permettent d'accéder aux éléments de l'arbre DOM.
[lp1173]

document.getElementById()

DOM

d = un élément DOM

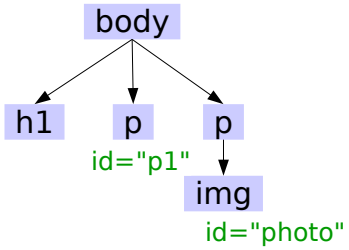
```
var d=document.getElementById('photo');
```

jQuery

j=« liste » d'un seul élément jQuery

```
var j=$('#photo');
```

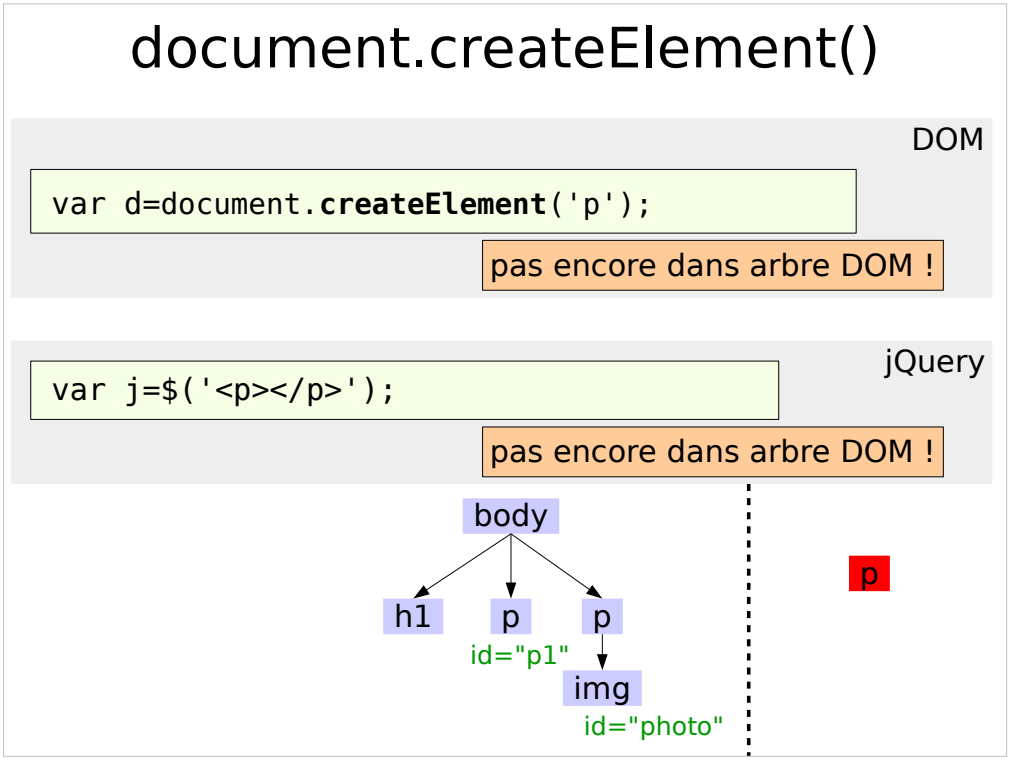
```
<body>
   <h1>Ceci est un titre</h1>
   <p id="p1">Un paragraphe</p>
   <p>2e paragraphe
      <img id="photo" src="ex.png"...
   </p>
</body>
```



La fonction getElementById() permet de chercher l'élément ayant un id fourni en argument. Par définition cet élément est unique. Ici "d" est un objet DOM.

L'équivalent en jQuery est $('#photo'). Remarquez que le type est différent. "j" est une **liste** jQuery (avec un seul élément), alors que "d" est un objet DOM.

On verra plus tard que l'on peut passer d'un type à l'autre.
[lp1172]



La fonction .createElement() permet de créer un élément.

Il est important de bien garder à l'esprit que cet élément n'est pas encore inséré dans l'arbre DOM. Pour l'instant il existe tout seul, sans parents, ni enfants.

jQuery permet d'obtenir un résultat similaire avec $(' '). jQuery est beaucoup plus puissant: on peut spécifier du code HTML complexe et donc créer directement de nombreux éléments.

Une fois de plus, "j" est une **liste** jQuery (avec un seul élément), alors que "d" est un objet DOM.
[lp1171]

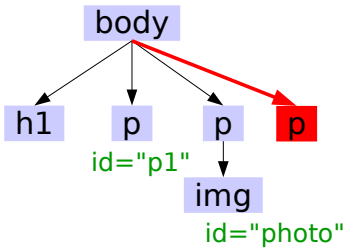
node.appendChild()

DOM

```
var d=document.createElement('p');
document.body.appendChild(d);
```

jQuery

```
var j=$('<p></p>');
$('body').append(j);
```



Une fois qu'un élément est crée, on peut l'insérer dans l'arbre DOM.

La fonction DOM appendChild() permet de le faire. Remarquez que dans cet exemple on utilise document.body qui permet d'accéder directement à l'élément <body>.

En jQuery, on obtient un résultat similaire avec .append()
[lp1170]

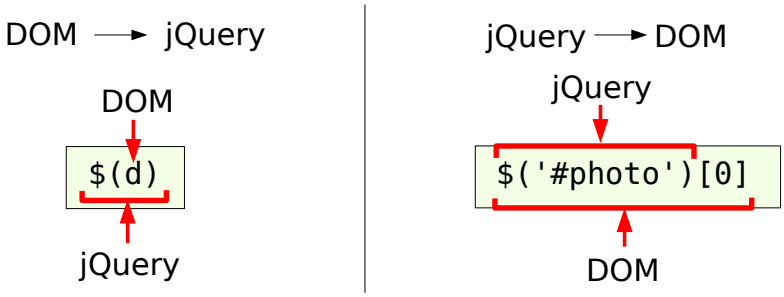
DOM & jQuery

j=« liste » d'un seul élément jQuery

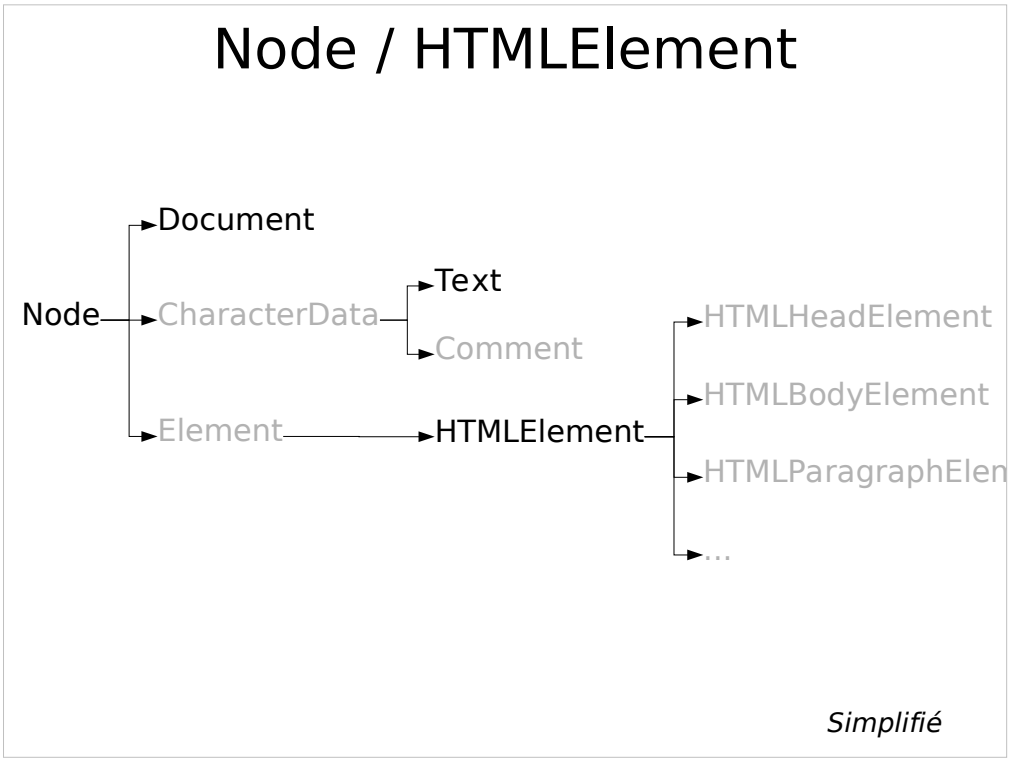
```
var j=$('#photo');
```

d = un élément DOM

```
var d=document.getElementById('photo');
```



On peut facilement passer d'une liste jQuery contenant un seul élément à l'objet DOM.
[lp1169]



Le DOM est une norme indépendante du langage de programmation qui défini une hiérarchie de classes (la notion de classe n'existe pas telle quelle en JS).

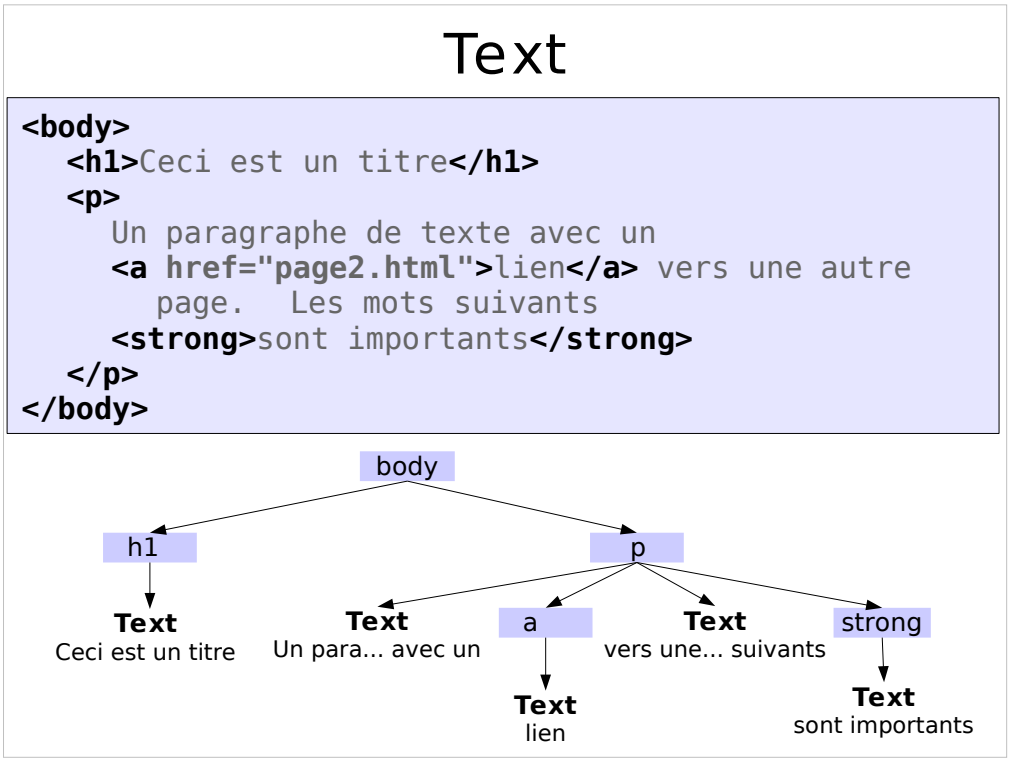
La plupart du temps on manipulera des objets DOM de type HTMLElement.

Node est l'interface définissant les opérations d'arbre (parent, enfants, frères).

Text représente du texte.

Un HTMLElement "est un" Node.
Un document "est un" Node.
Un Text "est un" Node.

Ces trois types peuvent donc être insérés dans un arbre.
[lp1168]



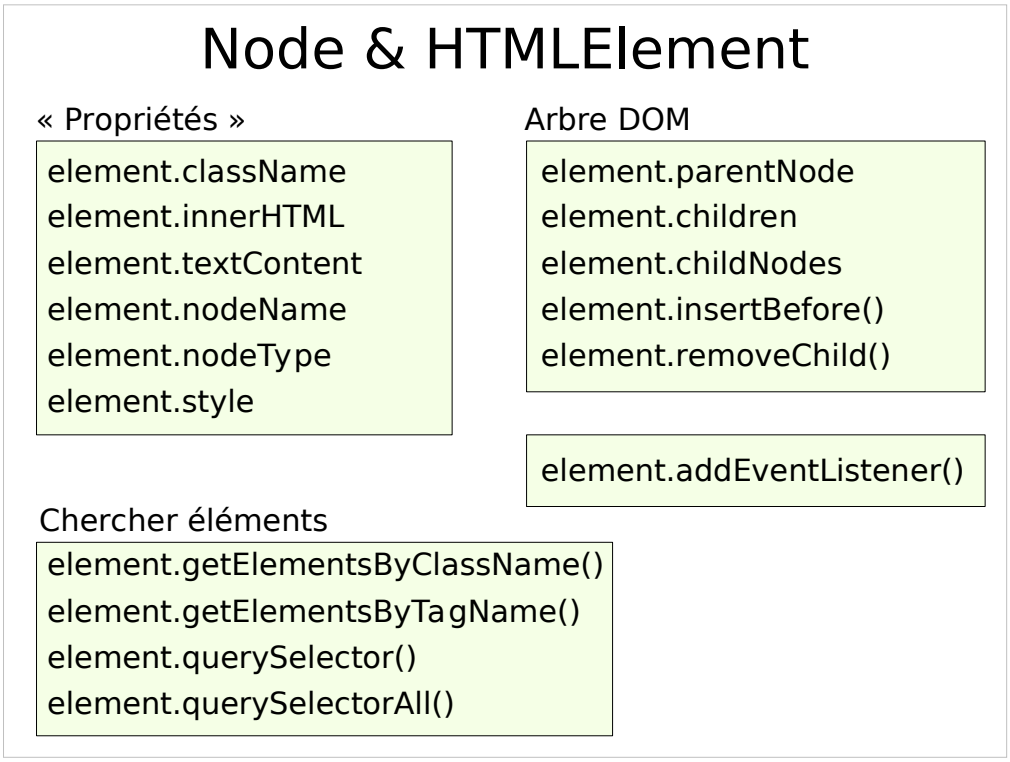
Dans les schémas d'arbre DOM on omet souvent de montrer le texte.

En réalité, dans l'arbre DOM, le texte est contenu dans des Text Node.

On peut manipuler les Text Node comme les autres Node.

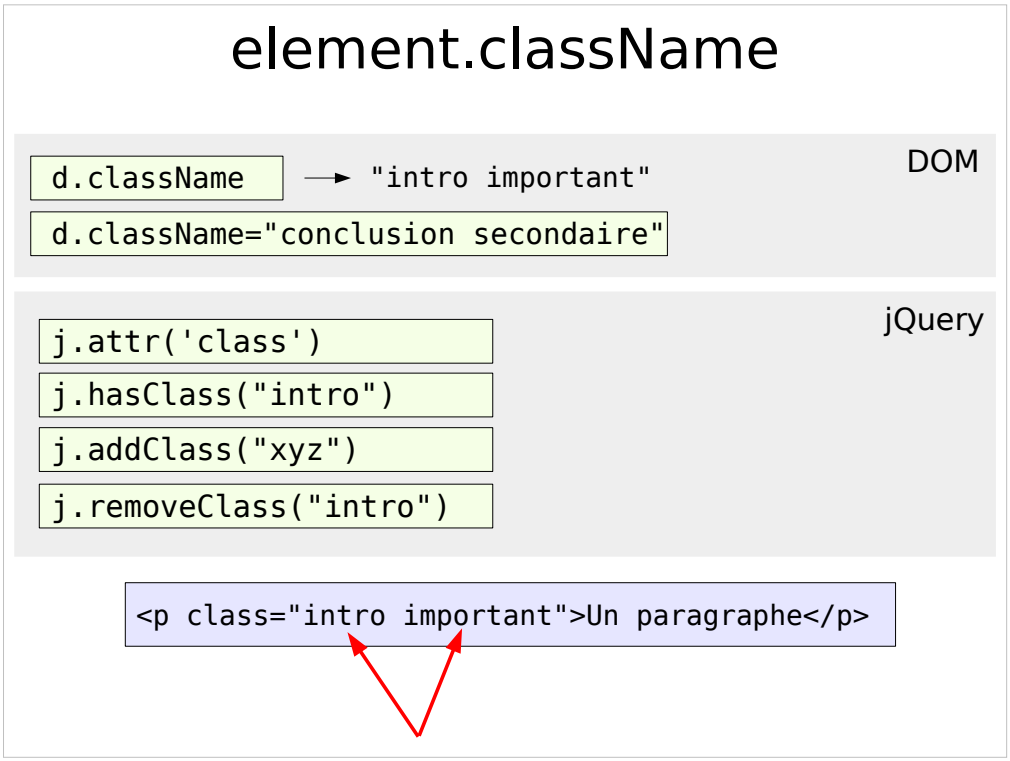
jQuery ne fourni qu'un accès très limité aux Text Node.

Quand on veut manipuler du texte (par exemple couper le texte d'un paragraphe en deux paragraphes), on utilise le DOM plutôt que jQuery.
[lp1167]



Node et HTMLElement fournissent de très nombreuse propriétés et méthodes.

[lp1166]



En HTML, l'attribut class peut contenir plusieurs classes séparées par des espaces.

Pour accéder à la classe d'un élément DOM on utilise className (en effet, "class" est un nom réservé du JS).

className donne la classe en entier. Ce n'est pas très pratique pour manipuler des classes multiples.

jQuery fourni des fonctions beaucoup plus pratiques, pour vérifier, ajouter et retirer des classes.
[lp1165]

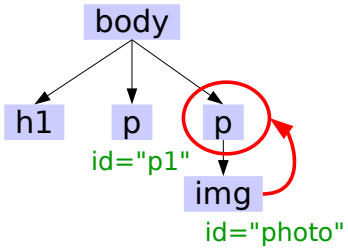
node.parentNode

DOM

```
var d=document.getElementById('photo').parentNode;
```

jQuery

```
var j=$('#photo').parent();
```



Le parcours de l'arbre DOM est une opération de base, très fréquente.

Ici on veut remonter au parent d'un élément.
[lp1164]

node.children

DOM

```
var liste=document.body.children;
for(var i=0;i<liste.length;i++)
{
   // ... liste[i] ...
}
```

jQuery

```
var liste=$('body').children();
liste.each(function()
{
   // ... this ...
});
```

Les fils directs (fils immédiats) sont donnés par .children ou .childNodes ".children" omet les Text Nodes.

Ces deux propriétés donnent une liste, qui peut-être manipulée comme un tableau (liste.length liste[n] ...)

[lp1163]

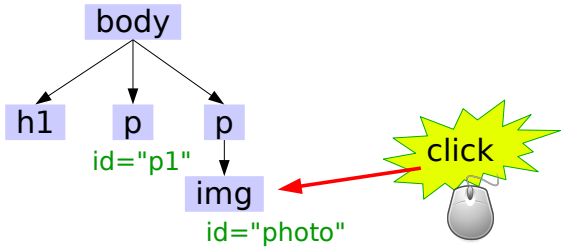
element.addEventListener()

DOM

```
var d=document.getElementById('photo');
d.addEventListener('click',function(){ ... });
```

jQuery

```
$('#photo').click(function(){ ... });
```



La fonction addEventListener permet d'ajouter un gestionnaire d'événements à un élément. (Rappel: le gestionnaire d'événements est une fonction qui est appelée quand un événement survient.)

addEventListener() ajoute un gestionnaire à un seul élément. Il faut donc appeler addEventListener() sur chaque élément visé.

L'approche jQuery est beaucoup plus pratique.
[lp1162]

Marcel.Bosc@iutv.univ-paris13.fr