Programmation client / serveur

Benjamin Canou - Christian Queinnec

Cours 5 du 17/12/2012

Architecture des Applications Réticulaires

Programmation côté client

Rappels rapides sur JavaScript
Manipulation du document et du style via DOM

Architecture des Applications Réticulaires

L'inévitable JavaScript

La mode est aux applications Web :

- Plus seulement des sites Web : le client devient intelligent.
- On veut donc programmer le navigateur :
  - Interagir avec l'utilisateur.
  - S'adapter au matériel (ex. capacités spécifiques aux mobiles).
  - Communiquer avec le serveur.
  - Modifier dynamiquement la page Web.

Donc on doit utiliser JavaScript :

- Le langage de script inclus dans les navigateurs grand public.
- La seule façon de programmer certains navigateurs (iOS, Windows 8, etc.)
- Plus simple pour accéder au document que les greffons.

Et on veut l'utiliser pour :

- Une riche galaxie de bibliothèques dont on veut profiter.
- Ses performances passées de ridicules à plutôt bonnes.

Fonctionnement des navigateurs

Modèle d'exécution : la boucle d'évènements

- Pas de mise à jour de l'affichage pendant la gestion d'évènements.
- Impossible de toucher à la queue d'évènements de l'instant courant.
- Exécution bloquée durant la gestion d'évènements.
- (Don't) try `javascript:setTimeout(function(){while(true){}})`.

Gestion d'évènements :

- Prendre en charge un évènement = lui affecter une fermeture JavaScript
- Évènements de ressources : matériel, requêtes réseau, etc.
- Évènements d'interface : souris, clavier, etc.
- Mécanisme particulier d'inhibition/propagation d'évènements (le bullage).
- Pour simuler un évènement : setTimeout (pour le tour prochain).

Accès à la représentation interne du document dans le navigateur.

- Arbre (sans cycle ni partage).
- Initialement :
  - Nœud HTML ⇝ un nœud DOM = un objet JavaScript.
  - Attribut HTML ⇝ attribut DOM = propriété JavaScript.
- Possibilité de modifier les nœuds, d'en ajouter, retirer, etc.

Comme en XML, les nœuds représentent :

- La structure (nœuds élément div, p, table) ;
- le contenu (nœuds texte) ;
- les méta-données (nœuds élément style, script, meta, etc.)

Les attributs stockent les données annexes associées aux nœuds.

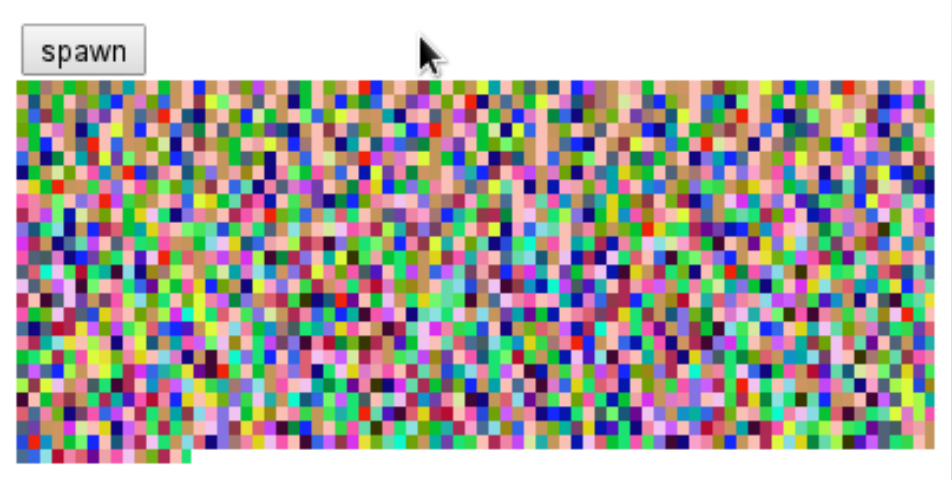
- Les gestionnaires d'évènements deviennent des fonctions JavaScript.
- Les styles deviennent des objets JavaScript complexes.
- Certains attributs subissent un traitement particulier (id, class, src, etc.)

- Un petit exemple en HTML :

```
1 : <body>
2 :   <p style="font-family: 'Comic Sans MS'">
3 :     Hello World
4 :   </p>
5 : </body>
```

- En version DOM :

```
1 : var p = document.createElement ("p");
2 : var t = document.createTextNode ("Hello World");
3 : p.style.fontFamily = "Comic Sans MS" ;
4 : p.appendChild (t);
5 : document.body.appendChild (p);
```



En trois niveaux de complexité :

- Étape 1 : clic $\longrightarrow$ une case de couleur aléatoire
- Étape 2 : clic $\longrightarrow$ un processus générateur de cases
- Étape 3 : clic sur une case $\longrightarrow$ disparition de la couleur et mort du processus

```
1 : <script>
2 :   function spawn () {
3 :     /* chosse random color and delay */
4 :     var timeout = Math.floor (Math.random() * 900) + 100
5 :     var r = Math.floor (Math.random() * 255)
6 :     var g = Math.floor (Math.random() * 255)
7 :     var b = Math.floor (Math.random() * 255)
8 :     /* retrieve marked HTML element */
9 :     var basket = document.getElementById ("basket")
10 :    /* create a color block */
11 :    var li = document.createElement ("span")
12 :    li.innerHTML = "    ␣"
13 :    li.style.backgroundColor = "rgb(" + r + ",␣" + g + ",␣" + b + ")"
14 :    /* insert our new item */
15 :    basket.appendChild (li)
16 :  }
17 : </script>
18 : <body>
19 :   <button onclick="spawn␣()">spawn</button><br/>
20 :   <span id="basket" style="font-size:5px"></span>
21 : </body>
```

```
1 : <script>
2 :   function spawn () {
3 :     var timeout, r, g, b /* ... */
4 :     /* async loop */
5 :     function loop () {
6 :       var basket, li = /* ... with local r, g, b */
7 :       basket.appendChild (li)
8 :       /* delayed recursion */
9 :       setTimeout (loop, timeout)
10 :    }
11 :    loop ()
12 :  }
13 : </script>
```

```
1 : <script>
2 :   function spawn () {
3 :     var timeout, r, g, b = /* ... */
4 :     /* store blocks */
5 :     var all = []
6 :     /* async loop */
7 :     var stop = false
8 :     function loop () {
9 :       if (stop) return;
10 :      var basket, li = /* ... */
11 :      li.onclick = function () {
12 :        /* clearTimeout (loop) */
13 :        stop = true;
14 :        for (i in all) basket.removeChild (all[i])
15 :      }
16 :      all.push (li)
17 :      basket.appendChild (li)
18 :      setTimeout (loop, timeout)
19 :    }
20 :    loop ()
21 :  }
22 : </script>
```

Bibliothèques JavaScript Généralistes

Prototype, Dojo, script.aculo.us, YUI, jQuery, etc.

- Interface uniforme masquant les singularités des navigateurs
- Fonction de confort, navigation dans DOM
- Animations / Transitions
- Support de widgets UI élaborés

La plus répandue : JQuery

- 15ko compressés (après miniaturisation (pour *minified*))
- Recherche, filtrage, modifications dans le DOM
- Mécanismes de quasi héritage (JS est un langage à prototype)
- Encapsulation d'envoi/traitement de requêtes asynchrones
- Bibliothèque d'effets visuels et greffons
- point d'entrée : $()

```
1 : $("div␣>␣p")
2 :    .filter(function(i) ...)
3 :    .each(function(i) this.html('<b>Coucou</b>'));
4 :
5 : $('#id').attr('title', "${this.src}");
6 :
7 : $(document.body).bind("click", function() {alert( $(this).text())});
8 :
9 : $("p.surprise").addClass("ohmy").show("slow");
10 :
11 : $(function () { alert("Fini!"); });
```

Tutoriel JavaScript

Langage de base
Modèle objet à prototypes
DOM & événements

Architecture des Applications Réticulaires

Programmation Client / Serveur

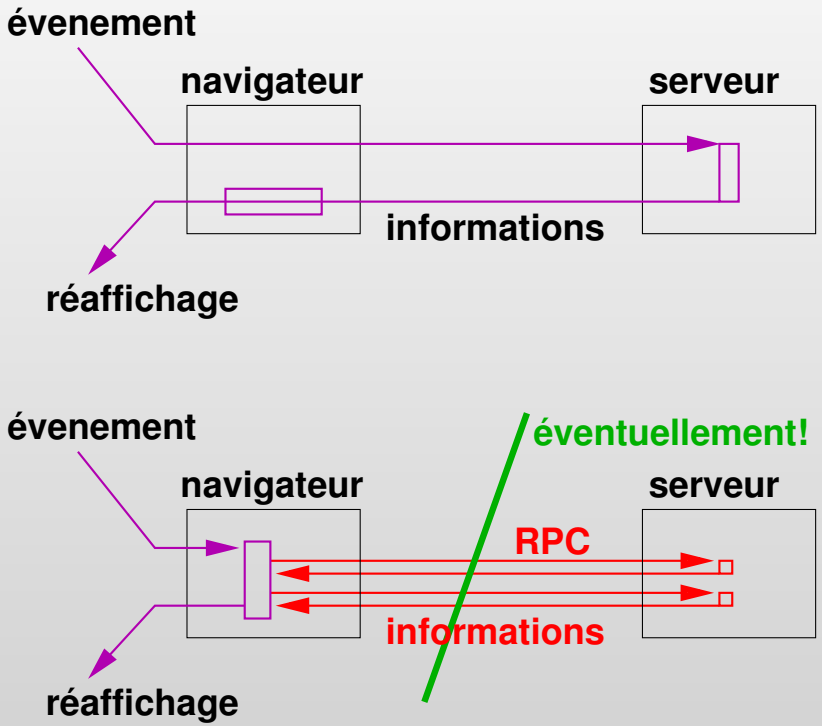
HTTP et AJAX
Formats d'échange
Canaux de communication

Architecture des Applications Réticulaires

Microsoft a introduit l'objet XMLHttpRequest qui permet, depuis le client, d'effectuer une requête HTTP (vers le domaine d'où provient la page).

```
1 : var req = window.ActiveXObject
2 :           ? new ActiveXObject("Microsoft.XMLHTTP")
3 :           : new XMLHttpRequest();
4 : req.setRequestHeader("X-Requested-With", "XMLHttpRequest");
5 : req.setRequestHeader("Accept", "*/*");
6 : req.onreadystatechange = function(isTimeout) {
7 :   if (httpRequest.readyState == 4) { // reponse OK
8 :     if (httpRequest.status == 200) {
9 :       httpRequest.responseText        // -> String
10 :      httpRequest.responseXML         // -> DOM
11 :    } } }
12 : req.open('GET', 'http://www.example.org/some.file',
13 :           isAsync );
14 : req.send();
```

On peut sortir du modèle une requête = une page.



- Requêtes plus petites mais plus nombreuses côté serveur
- Boutons *Previous* et *Next* délicats
- Site non arpentable donc non indexable (cf. mod_proxy)
- Attention à l'ordonnancement (canaux)

Codage des messages

Suivant la tâche :

- XML : parser intégré, bien pour morceaux de documents, attention au xmlns
- JSON : parsing facile, lisible, proche de JavaScript
- Spécialisé (texte, base64, etc.)

JSON : sous-ensemble de définition de données de JavaScript.

- tableau d'objets entre crochets
- table associative entre accolades et clés suffixées par deux points
- nombres, booléens, chaînes

```
1 : [ { a: 1, foo: [-1e-23, 2.0, true] },
2 :     "et␣meme␣\nUnicode␣\uABCD!" ]
```

Analyse syntaxique :

- Facile mais dangereux : *eval*
- Analyseur syntaxique spécialisé

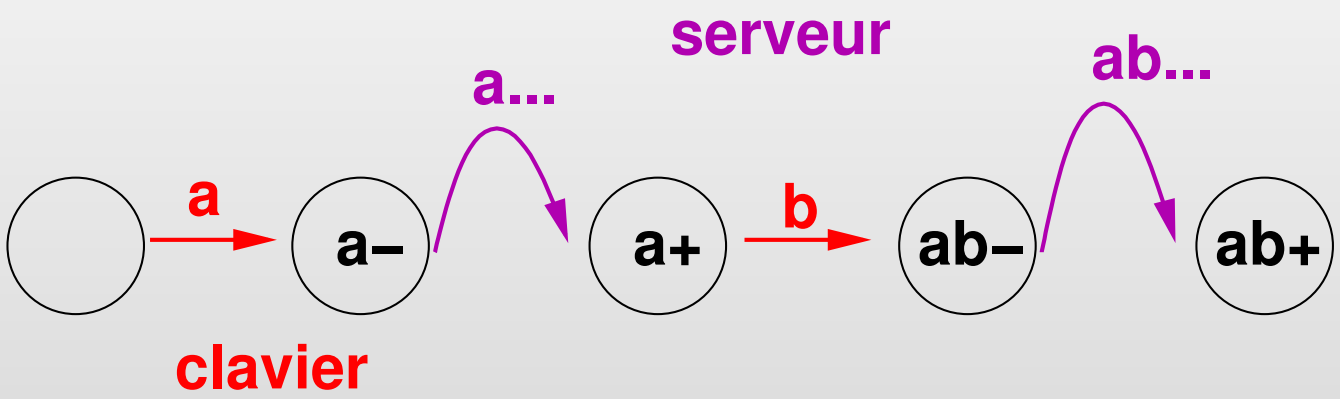
Sécurité

On ne peut envoyer de requêtes qu'au site d'où la page provient.
Comment faire du *mashup* ?

- Greffer dynamiquement dans le DOM un nœud script Une sorte de (très dangereuse) fonction eval, voir aussi JSOD (JavaScript On Demand).
- Établir des (coûteux pour le serveur) relais (proxys) sur le site originel
- Utiliser les iframes, exploitation de l'URL pour communiquer

Exemple: complétion automatique

Graphe d'état: complétion automatique

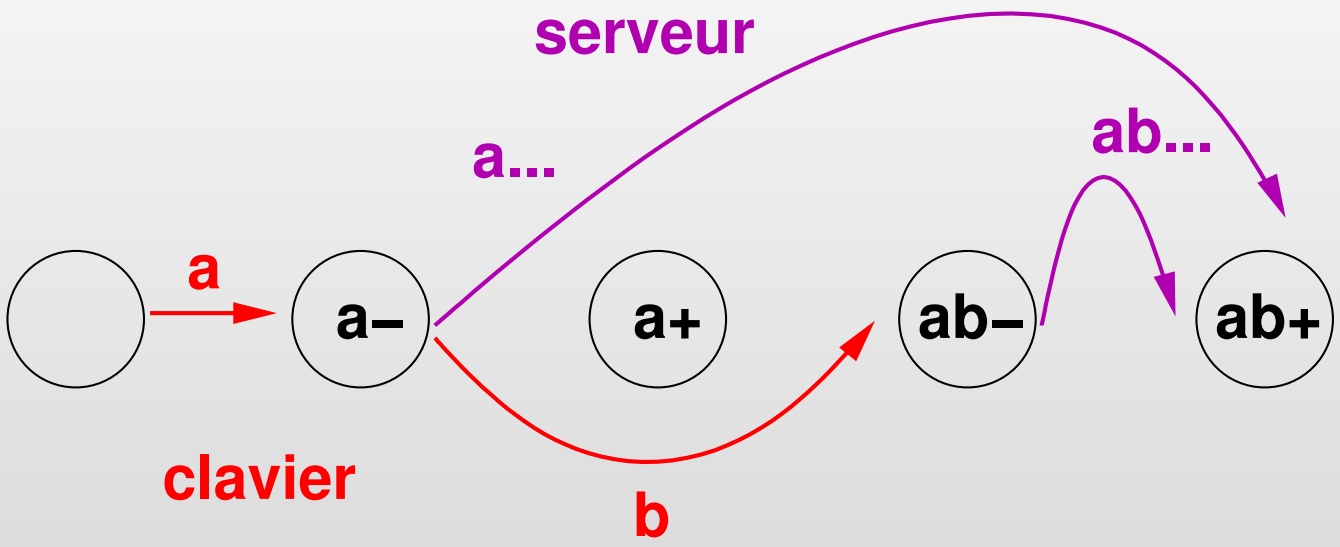


- Chaque lettre saisie envoie une requête au serveur.
- La réponse provoque un affichage des mots ayant ce préfixe.
- La réaction est à la vitesse du réseau (100 à 300 millisecondes).

Nota: *x-* est l'état où le clavier a reçu x et où la requête demandant les mots préfixés par x a été émise. *x+* représente l'état où la réponse à cette requête a été reçue.

Ordonnancement

Non instantanéité des transitions! On peut aussi observer:



Il faut donc :

- ordonner les requêtes et leurs réponses!
- Ordonnancer les fonctions de rappel (onreadystatechange)
- Considérer les codes 200, 304, 307, etc.

En fait, on veut des canaux !

- Pseudo canaux HTTP : Comet, HTTP Push
- Et si possible : WebSockets

Intégration au code serveur

Formes de couplage
Exemples

Architecture des Applications Réticulaires

Formes de couplage

Couplage fort :

- Toutes sortes de RPC (interface commune)
- Solutions mono-langage (HOP, Ocsigen, etc.)

Responsabilité au cadriciel serveur :

- Composants (ex. on ajoute une pré-validation aux champs)
- Génération de code par le moteur MVC

Responsabilité au code client :

- Modèle : squelette initial + construction incrémentale de la page
- Interface à un serveur REST
- Mashup côté client

```
 1 : use CGI;
 2 : use CGI::Ajax;
 3 :
 4 : my $cgi = new CGI;
 5 : my $pjx = new CGI::Ajax();
 6 : $pjx->register( 'JSFUNC' => \&perl_func );
 7 :
 8 : # Dans la page HTML, se trouvent des balises div avec les 3 id
 9 : #   onClick="JSFUNC(['source1','source2'],
10 : #                   ['dest1', jsfunc2]);"
11 : # et la fonction jsfunc2 (a la jsod)
12 :
13 : print $pjx->build_html( $cgi, \&Show_HTML);
14 :
15 : sub perl_func {
16 :   my ($input1, $input2) = @_;
17 :   return( "Ah␣$input2?", "Oh␣$input1!" );
18 : }
```

```
1 : sub Show_HTML {
2 :   my $html = <<EOHTML;
3 :   <HTML><HEAD><script>
4 : function jsfunc2() { ... }
5 :   </script></HEAD>
6 :   <BODY>
7 :     Enter something:
8 :       <input type="text" name="val1" id="s1"
9 :        onkeyup="JSFUNC(['s1','s1'], ['dest1',jsfunc2]);">
10 :    <br>
11 :    <div id="dest1"></div>
12 :  </BODY>
13 :  </HTML>
14 : EOHTML
15 :  return $html;
16 : }
```

Composants MVC

Que peut-on faire sans changer les composants existants ?

- Validation (partielle bien sûr!)
- Tableaux interactifs (tris locaux, défilement AJAX)
- Onglets sans recharger la page
- Animations (glisser-deposer, menu accordéon, etc.)

En fait, beaucoup de choses.

Conclusion

Beaucoup de nouvelles possibilités
Mais aussi beaucoup de nouvelles problématiques