Se simplifier AJAX avec Prototype

Merci à **denisC** pour ses conseils et l'aide qu'il m'a apportée pour la publication de cet article et à **Aspic** pour la relecture.

par Aurélien MILLET (ma page sur Developpez.com)

Date de publication : 01/10/2007

Dernière mise à jour :

En un peu plus de deux ans, le terme "AJAX" s'est répandu comme une traînée de poudre dans les bouches des développeurs Web. Interactivité, rapidité, tels sont les mots d'ordre des utilisateurs au sujet des applications. AJAX est là pour y répondre. Et comme un bonheur n'arrive jamais seul, des outils apparaissent pour aider les développeurs. Cet article a pour but de vous présenter les différents moyens que le framework JavaScript Prototype vous offre pour utiliser AJAX.

I - Introduction à AJAX et Prototype

I-A - Communication synchrone

Une application Web est un dialogue client / serveur. Le serveur est en écoute et attend que les clients viennent lui demander un service. Un serveur HTTP répond aux requêtes HTTP. Il passe à différents programmes les paramètres envoyés par le client. Ces programmes retournent du code HTML au serveur qui termine la transaction en faisant suivre ce code au client, dans une réponse HTTP.

Jusqu'à il y a peu (les technologies du Web évoluent très vite), les applications Web étaient synchrones. C'est-à-dire que l'utilisateur effectue une action sur une page Web, son navigateur envoie la requête au serveur, interprète sa réponse et charge la nouvelle page correspondant au code reçu. Pendant tout cet échange, l'utilisateur a lâché sa souris pour attendre sagement de voir apparaître ce qu'il a demandé.

I-B - Communication asynchrone

Aujourd'hui, il est possible de passer outre ce fonctionnement par l'utilisation toute récente de ce qui est désigné par l'acronyme bien connu : AJAX, soit : Asynchronous JavaScript And XML.

Alors, AJAX, qu'est-ce que c'est donc ?! Déjà, ce n'est pas UNE technologie. C'est l'utilisation conjointe de différentes technologies. Ensuite, on ne peut pas réellement dire que c'est une NOUVELLE technologie. Tout dépend du point de vue en fait. Les éléments qui lui servent de base (HTML, DOM, JavaScript, XML et objet XmlHttpRequest) lui sont bien antérieurs. Mais c'est l'utilisation de ces éléments qui est nouvelle.

Le pivot central est l'objet XmlHttpRequest (XHR). C'est le portage par Mozilla de l'objet XmlHttp développé par Microsoft et intégré à Internet Explorer 5 en tant qu'objet ActiveX. Cet objet permet d'envoyer des requêtes HTTP et de récupérer les données retournées par le serveur. Au fur et à mesure de l'avancée du traitement de la requête HTTP côté client puis côté serveur, l'objet passe par différents états. A chaque changement d'état, il déclenche un événement. Il est donc possible de suivre facilement la progression du traitement.

Il s'agit toujours d'échanges HTTP entre un client et un serveur mais cet échange est transparent pour l'utilisateur. Il n'attend plus forcément le chargement d'une nouvelle page en restant inactif. Son navigateur attend une réponse mais l'utilisateur, lui, a toujours la main. Classiquement, AJAX est utilisé pour mettre à jour dynamiquement une partie d'une page Web.

I-C - AJAX : intérêts et inconvénients

Le principal intérêt d'AJAX est d'apporter beaucoup plus d'interactivité aux applications Web. L'utilisateur n'a plus besoin de valider une page entière pour que ses actions soient prises en compte, il peut modifier l'interface à chaque clic. On évite ainsi le problème de la page blanche qui est affichée pendant quelques secondes le temps de charger toutes les données. Cette page blanche peut parfois dérouter l'utilisateur inexpérimenté.

L'autre avantage de réduire le volume des données transférées à chaque appel est que cela réduit le temps nécessaire au serveur pour répondre. On peut donc avoir des réactions quasi instantanées aux actions de l'utilisateur.

Un autre avantage est de décharger le serveur d'une partie du travail. Il collecte les informations demandées et les renvoie au client. A ce dernier de savoir quoi en faire, grâce à du code JavaScript. Son pendant négatif direct est la nécessité de disposer d'un navigateur Web supportant le JavaScript. Sans cela, impossible de naviguer correctement.

Il peut aussi y avoir un problème pour le développeur. Utiliser les technologies AJAX rajoute du code JavaScript et sans cadre précis, ce code finira par être brouillon et difficilement maintenable.

Attention aussi aux abus ! A trop chercher l'interactivité, on perd en accessibilité. AJAX peut également soulever des questions au niveau de la sécurité et du référencement.

I-D - Prototype : de l'utilité d'un framework

Voilà ma définition d'un framework : ensemble de bibliothèques / classes / fonctions et de conventions permettant le développement rapide d'applications. Donne un cadre structuré et impose une rigueur entraînant la production de code fiable et facile à maintenir. Vous pouvez aussi consulter la définition de **framework** dans le dictionnaire de Developpez.com.

Etant un développeur CakePHP convaincu, l'intérêt d'utiliser ce genre de "cadres de travail" me paraît juste évident. Aux débuts du Web, développer ses pages statiques en HTML dans son coin, ça passait. Aujourd'hui on parle dynamique et on parle surtout efficace, rapide, sûr. Non aux développements à l'artisanal au fond du garage pour un code cochon. Dites oui aux frameworks ! Vive le code lisible ! Aimez-vous les uns les autres. Pardon...

Le JavaScript est généralement le parent pauvre du développement Web. Ce n'est évidemment pas une règle générale mais le code est souvent réalisé rapidement, pour répondre aux besoins sans chercher à affiner, tant que ça marche. Et il faut tester sur tous les navigateurs puisqu'aucun ne supporte le même standard. **Prototype** c'est le chevalier en armure rutilante qui vient sur son fier destrier blanc immaculé sauver le pauvre développeur de ces horreurs. Il vous offre les moyens d'écrire du code concis et clair. Se chargeant des problèmes de navigateurs et autres tâches pénibles, il vous laisse vous concentrer sur l'aspect métier, ce que doit faire votre code. Et pour ça, on lui dit un grand merci !

I-E - AJAX sans et avec Prototype

Sans cadre spécial pour utiliser AJAX, il faut disposer d'une fonction spécifique qui va d'abord instancier un objet XmlHttpRequest, de différentes façons suivant le navigateur utilisé. Elle doit ensuite associer une fonction aux changements d'états de l'objet XHR (appelons la xhrStateChange). Elle doit enfin ouvrir une connexion avec le serveur en spécifiant une URL et éventuellement des paramètres.

La main passe ensuite à la fonction que nous avons nommée xhrStateChange. Elle est appelée à chaque changement d'état de l'objet XHR. On peut lui demander d'exécuter du code suivant l'état qui lui est indiqué. La transaction est considérée terminée avec succès lorsque xhr.readyState vaut 4 et lorsque xhr.status vaut 200 (ou plus généralement quand il est inférieur à 400). Quand c'est le cas, il faut ensuite passer la réponse du serveur à une fonction qui va se charger de mettre à jour la page Web avec ces nouvelles informations.

Vous pouvez regarder le code proposé par siddh dans **cet article** pour voir ce qu'il est nécessaire de mettre en oeuvre dans ce type de fonctionnement.

Avec Prototype, plus besoin de vous embêter à savoir si votre script crée correctement l'objet XHR ou si vous attendez bien le bon état. Vous créez un objet, auquel vous indiquez la page à interroger, les éventuels paramètres et la fonction à appeler quand la réponse a été reçue. C'est un exemple simple, il est possible de préciser plus de choses mais nous verrons les détails plus tard. Une ligne peut donc suffire à gérer la requête asynchrone, sans avoir à se préoccuper des problèmes liés à la plateforme d'exécution du JavaScript !

A noter que pour des raisons de sécurité, les requêtes AJAX ne peuvent être faites que vers des URLs de même domaine que la page contenant le JavaScript en cours d'exécution (c'est-à-dire même protocole, même serveur et

même port). Ceci vaut pour AJAX en général, Prototype n'a rien à voir là-dedans. Pour passer outre il faut par exemple passer par un "proxy" en PHP (lit le document demandé et en retourne le contenu) ou demander à l'utilisateur de diminuer les paramètres de sécurité de son navigateur.

II - Envoyer une requête

Prototype permet d'utiliser trois objets pour gérer des requêtes asynchrones :

- Request
  - Ajax.Request(url[, options])
  - Instancie un objet XHR, envoie la requête au serveur et reçoit sa réponse.
- Updater
  - Ajax.Updater(container, url[, options])
  - Effectue les mêmes opérations que Request puis met à jour le contenu d'un élément de la page avec les données reçues.
- PeriodicalUpdater
  - Ajax.PeriodicalUpdater(container, url[, options])
  - Effectue les mêmes opérations que PeriodicalUpdater mais répétées à intervalles réguliers.

Comme on peut le voir ci-dessus, ces trois objets ont tous au moins 2 paramètres en commun :

- url : Adresse de la page à interroger.

- options : Paramètres de la requête et fonctions callback. Ce paramètre est optionnel.

Remarque :

L'émission d'une requête HTTP via AJAX se fait suite à la levée d'un événement JavaScript. Je ne détaillerai pas cette partie mais juste au passage, petit conseil pour ceux qui gèrent encore leurs évènements JavaScript directement dans le code XHTML (ex : <a href="..." onclick="javascript:...) : laissez tomber cette méthode ! Utilisez plutôt les gestionnaires d'évènements de Prototype. Cela séparera définitivement les codes XHTML et JavaScript, c'est plus propre et plus facile à maintenir.

Pour en savoir plus sur ce sujet : **la documentation officielle** ou **la traduction sur developpez.com (v1.4.0)**. Attardez-vous surtout sur la méthode Event.observe().

II-A - Options configurables

L'un des paramètres (optionnel) est donc une liste d'options. Elle est passée sous forme d'un élément au format JSON (assimilable à un tableau associatif). Voici les différentes options accessibles et les valeurs qu'il est possible d'indiquer :

Nom	Valeur par défaut	Description
asynchronous	true	Indique si la requête est synchrone ou non.
contentType	application/x-www-form-urlencoded	En-tête "Content-Type" de la requête. Par défaut, indique que c'est une URL qui est envoyée. A modifier pour envoyer par exemple du XML.
encoding	UTF-8	Encodage du contenu de la requête.
method	POST	Méthode HTTP de la requête. Autre valeur possible : "GET".
parameters	null	Paramètres de la requête, à passer au script appelé. Ils peuvent être

		exprimés principalement sous deux formes : {a: 1, b: 5} ou 'a=1&b=5'
postBody	none	Contenu spécifique pour le corps de la requête. Si une valeur est indiquée, l'argument "parameters" ne sera pas pris en compte.
requestHeaders	X-Requested-With : 'XMLHttpRequest' X-Prototype-Version : version courante de Prototype Accept : 'text/javascript, text/html, application/xml, text/xml, */*' Content-type : construit en fonction de contentType et encoding	En-têtes de la requête. Peut recevoir une valeur sous deux formes : un objet (ses attributs donnent les valeurs des en-têtes) ou un tableau (case paire : nom de l'en-tête ; case impaire : valeur).

II-B - Fonctions callbacks

Les callbacks représentent différents points du cycle d'exécution d'une requête. Il est possible d'associer du code à ces étapes. Pour la liste complète des callbacks : **la documentation officielle** ou **la traduction sur developpez.com (v1.4.0)**. Il faut être prudent en les utilisant car certains sont implémentés différemment suivant les navigateurs.

Les deux callbacks qui sont le plus couramment utilisés sont "onSuccess" et "onFailure". Les deux sont invoqués quand une requête est terminée. onSuccess si le code d'état de la requête est entre 200 et 299, onFailure sinon.

Ils reçoivent deux paramètres :

- L'objet XHR (souvent appelé "transport") ;

- L'évaluation JSON de la réponse si réception d'un en-tête X-JSON, null sinon.

II-C - L'objet Request

- Ajax.Request(url[, options])
- Objet de base pour traiter les requêtes AJAX : instancie un objet XHR, envoie la requête au serveur et reçoit sa réponse. Il vous laisse indiquer quoi demander et quoi faire de la réponse.
- Paramètres : URL de la page à interroger et liste optionnelle des options.

Exemple d'utilisation de l'objet Request

```
new Ajax.Request(
 'http://mon-domaine.com/ma/page.php',
 {
  method: 'get',
  parameters: {nom1: valeur1, nom2: valeur2},
 }
);
```

Voir une page exemple

II-D - L'objet Updater

- Ajax.Updater(container, url[, options])
- Effectue les mêmes opérations que Request puis met à jour le contenu d'un élément de la page avec les données TEXTE reçues (celles contenues dans responseText).
- Paramètres : identifiant de l'élément DOM à mettre à jour, URL de la page à interroger et liste optionnelle des options. En plus des options décrites plus haut, l'objet Updater en propose deux autres :

Nom	Valeur par défaut	Description
evalScripts	false	Le contenu de la réponse peut contenir des balises <script>. Ce paramètre détermine si ces éléments sont évalués ou pas. Si evalScripts est à "true", ces blocs ne seront pas inclus dans la page mais passés à la fonction eval(). Si vous définissez des fonctions dans ces blocs, il ne faut pas utiliser "function foo() { ... }" mais "foo = function() { ... }".
insertion	none	Par défaut, quand un élément DOM est mis à jour, son ancien contenu est effacé. Si ce comportement n'est pas souhaité, il est possible d'indiquer où ajouter, dans l'élément, le texte de la réponse. Les valeurs possibles sont : "Insertion.After", "Insertion.Before", "Insertion.Bottom", "Insertion.Top".

Il est possible de mettre à jour un élément DOM suivant l'état de la réponse :

- Updater('monDiv'...) : c'est toujours l'élément monDiv qui est concerné ;
- Updater({ success: 'monDiv' }...) : l'élément monDiv n'est mis à jour qu'en cas de réussite de la requête ;
- Updater({ success: 'monDiv', failure: 'monAutreDiv' }...) : le texte de la réponse sera affiché dans l'élément monDiv en cas de réussite ou dans l'élément monAutreDiv en cas d'échec.

Exemple d'utilisation de l'objet Updater

```
new Ajax.Updater(
 'monDiv',
 'http://mon-domaine.com/ma/page.php',
 {
  method: 'get',
  parameters: {nom1: valeur1, nom2: valeur2},
  insertion: Insertion.Bottom
 }
);
```

Voir une page exemple

II-E - L'objet PeriodicalUpdater

- Ajax.PeriodicalUpdater(container, url[, options])
- Effectue les mêmes opérations que PeriodicalUpdater mais répétées à intervalles réguliers. Comme Updater, ne prend en compte que le contenu de responseText.
- Paramètres : identifiant de l'élément DOM à mettre à jour, URL de la page à interroger et liste optionnelle des options. PeriodicalUpdater supporte les options de base ainsi que celles apportées par Updater (se référer aux sections dédiées). Il en propose également deux nouvelles :

Nom	Valeur par défaut	Description
frequency	2	Temps en secondes entre deux exécutions de requêtes (c'est en fait une période et non une fréquence). Ne pas mettre un temps trop court au risque d'envoyer des requêtes avant que les précédentes ne se soient terminées.
decay	1	Multiplicateur de la période quand la réponse du serveur ne change pas. A chaque fois que la réponse reçue est la même que la précédente, la valeur courante de "frequency" est multipliée par la valeur de "decay". Les requêtes seront donc envoyées de plus en plus espacées. Quand la réponse reçue diffère de la précédente, "frequency" retrouve sa valeur d'origine. La valeur par défaut de "decay" est 1, ce qui veut dire que "frequency" n'est pas modifiée.

Remarque : il est possible d'arrêter l'exécution d'un objet PeriodicalUpdater avec la méthode stop(). Pour le relancer, utiliser start().

Exemple d'utilisation de l'objet PeriodicalUpdater

```
new Ajax.PeriodicalUpdater(
 'monDiv',
 'http://mon-domaine.com/ma/page.php',
 {
  insertion: Insertion.Bottom,
  frequency: 5
 }
);
```

Voir une page exemple

III - Recevoir et traiter une réponse

Réagir au changement d'état d'un objet XHR se fait en associant du code à un état. Mais il y a deux façons de faire cela :

- Spécifiquement pour une requête donnée ;
- Globalement pour tous les objets XHR qui seront créés, via l'utilisation d'objets "responders".

III-A - Comportements spécifiques

C'est la méthode la plus utilisée : pour une requête donnée, on attend un certain type de réponse qui doit subir un traitement précis. Pour cela, lors de la déclaration de la requête, il faut préciser, dans les paramètres, les états auxquels réagir (les callbacks) et leur associer du code à exécuter.

Exemple : on informe l'utilisateur quand la requête est terminée

```
new Ajax.Request(
 'http://mon-domaine.com/ma/page.php',
 {
  method: 'get',
  parameters: {nom1: valeur1, nom2: valeur2},
  onSuccess: function() { alert('Requête terminée avec succès.') },
  onFailure: function() { alert('Requête échouée.') }
 }
);
```

III-B - Comportements génériques ("responders")

Si pour un même état de tous les objets XHR, un même code doit être exécuté, il est possible de ne pas l'écrire dans les déclarations de toutes les requêtes. Il suffit de l'enregistrer en tant que "responder".

Exemple : alerter à chaque fois qu'une exception est levée

```
Ajax.Responders.register({
 onException: function() { alert('Une exception a été levée !') };
});
```

Il est aussi possible de supprimer ce qui a été enregistré mais pour cela il faut avoir gardé une référence.

Exemple : alerter à chaque fois qu'une exception est levée

```
responder = function() { alert('Une exception a été levée !') };
Ajax.Responders.register({
 onException: responder
});

Ajax.Responders.unregister(responder);
```

III-C - Pré-requis aux sections suivantes

Comme déjà mentionné dans la partie sur les callbacks (section 4.2), les fonctions associées au traitement de la réponse reçoivent deux paramètres :

- L'objet XHR (souvent appelé "transport") ;
- L'évaluation json de la réponse si réception d'un en-tête X-JSON, null sinon.

Dans les parties suivantes nous prendrons comme exemple l'instanciation suivante :

```
new Ajax.Request(
 'server.php',
    {
        onSuccess: function(transport, json) {
            document.write(
                "=> transport.responseText : " + transport.responseText
                + "<br />=> transport.responseXML : " + transport.responseXML
                + "<br />=> json : " + json
            );
        }
    }
);
```

III-D - Recevoir du texte

Si aucun en-tête particulier n'est émis avec la réponse, le flux reçu est stocké dans l'attribut responseText de l'objet XHR.

server.php
```
<?php echo 'Ceci est un test d\'envoi de simple texte.' ?>
```

Résultat affiché
```
=> transport.responseText : Ceci est un test d'envoi de simple texte.
=> transport.responseXML : null
=> json : null
```

III-E - Recevoir du XML

Si par contre un en-tête XML est émis en début de flux de réponse, son contenu sera accessible via les attributs responseText et responseXML.

server.php
```
<?php
header("Content-type: text/xml");
echo '<?xml version="1.0" encoding="UTF-8"?>';
?>
<root>
 <data>Donnee 1</data>
 <data>
  <sub1>Donnee 2</sub1>
  <sub2>Donnee 3</sub2>
 </data>
</root>
```

Résultat affiché
```
=> transport.responseText :  Donnee 1  Donnee 2  Donnee 3
=> transport.responseXML : [object XMLDocument]
=> json : null
```

responseText contient sous forme de texte l'ensemble de la réponse (balises et données). Les balises ne sont pas affichées dans notre exemple puisque le navigateur a essayé de les interprêter.

responseXML contient la réponse sous forme de flux XML et peut être manipulé comme tel.

III-F - Recevoir du JSON

Le **JSON** (JavaScript Object Notation) est plus simple, plus lisible et plus léger que le XML. Comme en plus, Prototype en facilite l'évaluation, il est conseillé de l'utiliser quand c'est possible. Si un en-tête X-JSON est émis en début de flux de réponse, Prototype l'analyse automatiquement et stocke le résultat dans l'objet json.

Il est possible de générer du JSON en PHP, soit avec l'implémentation native depuis PHP 5.2 soit en utilisant **cette librairie**.

server.php (avec la fonction native de PHP)

```php
<?php
$datas = array(
    'root' => array(
        'data' => 'Donnee 1',
        'otherData' => array(
            'sub1' => 'Donnee 2',
            'sub2' => 'Donnee 3'
        )
    )
);

header("X-JSON: " . json_encode($datas));
?>
```

server.php (en utilisant de la librairie)

```php
<?php
$datas = array(
    'root' => array(
        'data' => 'Donnee 1',
        'otherData' => array(
            'sub1' => 'Donnee 2',
            'sub2' => 'Donnee 3'
        )
    )
);

require_once('JSON.php');
$json = new Services_JSON();
header("X-JSON: " . $json->encode($datas));
?>
```

Résultat affiché

```
=> transport.responseText :
=> transport.responseXML : null
=> json : [object Object]
```

L'objet json est alors facilement exploitable. Dans notre exemple, pour accéder à la valeur de "sub1", il suffit de passer par "json.root.otherData.sub1".

IV - Conclusion

J'espère que cet article vous aura montré les avantages énormes qu'apporte l'utilisation de Prototype, ne serait-ce que sur la gestion d'AJAX. Sachant que ce n'est qu'une petite partie de ce qu'il propose, je vous recommande très fortement son utilisation pour accélérer vos développements JavaScript et simplifier votre code.

Vous pouvez entre autres lire le code des pages utilisées comme exemples au cours de cet article. J'y utilise différents aspects de Prototype (gestion d'évènements, manipulation d'éléments, méthodes utiles comme $ et $$...).

Liens sur Developpez.com

- **Section AJAX : FAQ, cours, tutoriels...**
- **Traduction de la documentation Prototype v1.4.0**
- **AJAX et l'objet XMLHttpRequest**

Liens sur le site officiel de Prototype

- **Accueil**
- **Introduction à AJAX**
- **Introduction à JSON**
- **API AJAX**