

Ajax supportant les navigateurs sans Javascript ("Backward compatible") avec Wicket

par Erik van Oosten Joseph Pachod

Date de publication : 03/05/07

Dernière mise à jour : 03/05/07

Cet article, paru sur **le blog anglophone** d'Erik van Oosten, traite de moyens de rendre un site visible tant via Ajax que sans, le tout aisément et pour tout navigateur.

- I - Avant propos
- II - Introduction
- III - Assembler des composants Wicket (courte introduction à Wicket)
- IV - Pages dynamiques avec Wicket
- V - Conclusion

I - Avant propos

Cet article est une traduction de l'article "Backward compatible AJAX development with Wicket" paru **sur un blog anglophone** et écrit par Erik van Oosten. Cette traduction a été effectuée avec son accord.

II - Introduction

Malgré l'acceptation générale d'Ajax, il y a toujours certaines réticences dans certains secteurs. L'interaction riche fournie par Ajax ne peut être utilisée par des personnes qui, du fait d'un handicap, ne peuvent utiliser de navigateur supportant Javascript ou CSS. Pour un secteur tel le service public, il n'est pas acceptable d'exclure ces personnes, une attitude qui devrait être, je pense, plus répandue.

De nos jours, nombreux sont les développeurs qui ne peuvent ou ne veulent développer des applications Ajax qui ne fonctionnent qu'avec Internet Explorer. Faire une application qui fonctionne aussi sans Javascript est tout simplement hors de portée pour de nombreuses entreprises. Pour faire cela, plusieurs solutions sont connues. Une approche générale est donnée dans l'article **Unobstrusive Javascript**, mais cet article présente une approche différente, plus flexible, basée sur **Wicket**.

Wicket est l'un des nouveaux frameworks "orientés composants". Dans de tels frameworks, le développeur combine des composants jusqu'à avoir une page web. Les composants sont simples à développer et à modifier individuellement, d'où leur intérêt. Avec un framework web "orienté page", le développeur doit généralement créer toute la page en même temps. Struts, par exemple, conseille de recueillir d'abord les informations nécessaires à toute la page avant qu'une JSP n'affiche la page (entière).

III - Assembler des composants Wicket (courte introduction à Wicket)

Un composant Wicket se réalise en écrivant un fragment html (le modèle) ainsi que du code Java correspondant, afin de coupler les deux. La création et la liaison des composants se fait pendant la phase d'initialisation de l'application. A l'exécution, des composants peuvent être ajoutés, changés voir même complètement remplacés.

Voyons un exemple. Voici un modèle html et le code Java associé :

```
<h1 wicket:id="title">_Titre du modèle</h1>
```

```
add(new Label("title", "Le vrai titre"));
```

Ce composant "Label" est couplé avec une balise h1. Ainsi, lors du rendu, le vrai titre sera affiché dans cette balise, comme suit :

```
<h1>Le vrai titre</h1>
```

Composer des éléments est tout aussi simple. Supposons que nous voulions utiliser un titre avec un sous titre en de nombreux endroits sur notre site web. Nous allons créer un composant pour cela, un Panel (panneau) pour être précis :

```
<wicket:panel>
  <h1 wicket:id="title">_Titre du modèle</h1>
  <h2 wicket:id="subtitle">_Sous titre du modèle </h2>
</wicket:panel>
```

```
class TitlePanel extends Panel {
  public TitlePanel(String id, String title, String subtitle) {
    super(id);
    add(new Label("title", title));
    add(new Label("subtitle", subtitle));
  }
}
```

Ce Panel peut maintenant être utilisé (par exemple dans le modèle d'une autre page) comme suit :

```
<span wicket:id="titlepanel"></span>
```

```
add(new TitlePanel(
  "titlepanel", "Un vrai titre", "avec son sous titre"));
```

Les liens entre pages sont réalisés avec le composant "Link" (lien) :

```
<a href="#" wicket:id="detaillink">Détails du livre</a>
```

```
add(new Link("detaillink") {  
    void onClick() {  
        setResponsePage(new DetailPage(bookId));  
    }  
});
```

Le composant Link, lors du rendu, va générer et insérer l'attribut href de la balise <a>. Quand le lien sera cliqué, Wicket va appeler la méthode onClick(). Dans cet exemple, la page donnée en réponse est construite à la volée (les pages étant bien sûr également des composants). Après cela, la page de réponse est envoyée au navigateur. Si la méthode onClick() avait été laissée vide, la page de réponse n'aurait pas changé : la même page aurait été réaffichée.

IV - Pages dynamiques avec Wicket

Le composant Link ne sert pas uniquement à passer d'une page à une autre. Avec Wicket, il est tout aussi facile de remplacer une partie d'une page en remplaçant un composant par un autre. Continuons l'exemple précédent :

```
final BookDetailPanel bookDetailPanel = ...;
add(bookDetailPanel);
add(new Link("detaillink") {
    void onClick() {
        bookDetailPanel.replaceWith(
            new BookDetailPanel(bookId));
    }
});
```

Cliquer sur le lien entraîne un changement sur la page courante. Après cela, cette même page courante est affichée à nouveau, et un autre livre est affiché. Remarquez que très peu de code est requis. Dans de nombreux autres frameworks les données de la page entière doivent être à nouveau collectées.

Le lecteur observateur aura remarqué que, de nos jours, remplacer une partie de page est généralement fait en Ajax. Dans l'exemple toutefois, nous n'avons pas utilisé une ligne de Javascript. Vu que toute la page est envoyée au navigateur à chaque fois, changeons encore un peu l'exemple :

```
final Component bookDetailPanel = ...;
bookDetailPanel.setOutputMarkupId(true);
add(bookDetailPanel);
add(new AjaxFallbackLink("detaillink") {
    void onClick(AjaxRequestTarget target) {
        Component newBookDetailPanel =
            new BookDetailPanel(bookId);
        newBookDetailPanel.setOutputMarkupId(true);
        bookDetailPanel.replaceWith(newBookDetailPanel);
        if (target != null) {
            target.addComponent(newBookDetailPanel);
        }
    }
});
```

Pendant la phase de rendu, le composant "AjaxFallbackLink" génère tant un attribut href qu'un traitement Javascript sur onclick, le tout associé à l'élément html a. De plus, le composant s'assure que les fichiers Javascript Wicket requis soient ajoutés au header HTML.

Ainsi, en fonction du support Javascript au sein du navigateur, ce dernier soit demandera à nouveau l'url courante soit fera une requête ajax. Dans les deux cas, Wicket appellera la méthode onClick.

Dans le premier cas, l'argument de la méthode onClick est nul. Tout fonctionnera ainsi comme dans l'exemple précédent. Quand un appel Ajax est fait, Wicket fera le rendu du composant qui a été ajouté à l'AjaxRequestTarget et enverra le résultat au navigateur. Du côté du navigateur, le code Javascript Wicket regardera les éléments à modifier en utilisant l'attribut id. D'ailleurs, pour être sûr que cet id soit présent, la méthode setOutputMarkupId(true) a été appelée.

Ainsi, avec juste quelques lignes de code, nous avons créé une application Ajax qui fonctionne même dans les navigateurs sans Javascript.

V - Conclusion

Cet article ne montre qu'une petite partie des capacités Ajax de Wicket. Par exemple, il est facile de laisser l'utilisateur savoir qu'un appel Ajax est en cours, d'exécuter du Javascript avant un appel Ajax ou de valider les données d'un formulaire lors de leurs saisies.

Wicket n'est pas seulement un fantastique framework pour construire aisément des applications modernes et maintenables, il est même possible de le faire d'une telle façon que les navigateurs anciens ou spéciaux puissent y accéder.

