

Ajax facile avec Ruby on Rails, Prototype, script.aculo.us et les RJS

par [Pierre-Baptiste Naigeon](#) ([Tutos](#), [tests et articles web](#), [Ruby et Rails](#))

Date de publication : 21 Mai 2007

Dernière mise à jour : 10 Octobre 2007

Nous allons voir au cours de cet article la facilité de mise en place qu'offre Rails pour faire de l'ajax, les superbes effets que cela permet, et tout cela sans une ligne de Javascript ou presque, grâce aux librairies **Prototype** et **script.aculo.us**.

Nous verrons également l'intérêt des RJS.

- I - Introduction
- II - Rappels
 - II-A - Génération d'un contrôleur
 - II-B - Fichier de configuration de la base de données
 - II-C - Génération d'un modèle
 - II-D - Utilisation des fichiers de migration
 - II-E - Utilisation des fixtures
 - utilisateurs.yml
 - utilisateurs.csv
 - Déploiement des fixtures
- III - Les RJS (Remote JavaScript)
- IV - Quelques petits effets graphiques
 - IV-A - Effets combinés
 - Fade / Appear / ToggleAppear
 - Highlight
 - SlideUp / SlideDown / ToggleSlide
 - Grow / Shrink
 - BlindUp / BlindDown / ToggleBlind
 - Pulsate
 - Fold
 - Puff
 - Squish
 - SwitchOff
 - DropOut
 - IV-B - Effets 'de base'
 - Insérer / Remplacer / Supprimer un élément
 - Cacher / Afficher un élément / alterner entre les deux
 - Agrandir / Réduire un élément
 - Modifier le style d'un élément
 - Retarder un effet
 - Exécuter un effet au chargement de la page
 - Récapitulatif
- V - Manipulation de formulaires
 - V-A - Ajouter dynamiquement un champ à un formulaire en fonction de la valeur d'un autre
 - V-B - Mettre à jour dynamiquement un champ
 - V-C - Validation dynamique d'un formulaire
 - form_remote_tag
 - submit_to_remote
- VI - Auto-complétion
- VII - Tri d'une liste d'éléments
- VIII - Drag & Drop is easy
- IX - Petit mélange des deux
- X - Allons plus loin
 - X-A - Créer ses propres helpers
 - X-B - Créer ses propres fonctions Javascript
- XI - Conclusion, remerciements et liens complémentaires

I - Introduction

AJAX est une technologie extrêmement intéressante à mettre en place, puisqu'elle permet une interaction forte avec le client. L'idée est de pouvoir mettre à jour des éléments de la page sans avoir à recharger la page complète.

Qui dit AJAX dit Javascript, mais Rails va nous permettre de faire tout cela sans en faire une ligne, ou presque.

Nous commencerons par un petit rappel pour les moins à l'aise d'entre vous, les autres peuvent passer directement au chapitre III (si vous êtes débutant avec Rails, vous pouvez aller consulter le tutorial "[Initiation à Rails](#)", par Yann Marec).

Dans ce cours, nous allons utiliser les différentes bibliothèques Javascript de Rails.

Afin que ces bibliothèques soient déclarées dans toutes nos pages, nous allons définir un fichier "`app/views/layouts/application.rhtml`" comme suit :

```
application.rhtml

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <%= javascript_include_tag :defaults %>
    <title><%= @body_title || "titre par défaut" %></title>
  </head>
  <body<%= @body_extras %>>
    <%= yield %>
  </body>
</html>
```

Ce fichier se charge d'inclure les bibliothèques Javascript dont nous aurons besoin, et nous permettra de stipuler pour chaque vue une valeur pour la balise **title**, et éventuellement des options à passer à la balise **body**.

II - Rappels

II-A - Génération d'un contrôleur

Pour générer un contrôleur, nous taperons la ligne suivante à la racine de notre application Rails : `"ruby script/generate controller nom_du_contrôleur"`.

Par exemple, nous souhaitons créer un contrôleur `"test"`, ce qui nous donnera : `"ruby script/generate controller test"`. Voici ce que Rails vous renverra :

Retour lors de la création d'un contrôleur

```
exists  app/controllers/
exists  app/helpers/
create  app/views/test
exists  test/functional/
create  app/controllers/test_controller.rb
create  test/functional/test_controller_test.rb
create  app/helpers/test_helper.rb
```

Ruby nous a donc créé :

- **create app/controllers/test_controller.rb** : le contrôleur lui-même
- **create app/views/test** : le répertoire dans lequel nous placerons les vues associées au contrôleur
- **create test/functional/test_controller_test.rb** : un fichier dans lequel nous écrirons nos tests fonctionnels
- **create app/helpers/test_helper.rb** : un fichier qui contiendra les helpers (assistants) de notre contrôleur

II-B - Fichier de configuration de la base de données

Le fichier qui sert à configurer votre base de données est le suivant : `"config/database.yml"`.


Il vous permet de configurer quelle base de données utiliser dans les trois modes de ruby : développement, test et production.

Voici un exemple pour le mode développement :

Contenu du fichier 'database.yml' pour le mode développement

```
development:
  adapter: mysql      # Le type de la base
  database: mon_application # Le nom de la base
  username: root      # L'identifiant à utiliser pour se connecter
  password:           # Le mot de passe
  host: localhost     # L'adresse du serveur qui héberge la base
```

Configurez ce fichier en fonction de votre environnement de travail, et passons à la suite.

 **Il faut tout de même créer la base de données à la main.**

II-C - Génération d'un modèle

Pour générer un modèle, nous taperons la ligne suivante à la racine de notre application Rails : `"ruby script/generate model nom_du_modele_au_singulier"`.

Par exemple, nous souhaitons créer un modèle `"utilisateur"`, ce qui nous donnera : `"ruby script/generate model utilisateur"`. Voici ce que Rails vous renverra :

Retour lors de la création d'un modèle

```
exists  app/models/
exists  test/unit/
exists  test/fixtures/
create  app/models/utilisateur.rb
create  test/unit/utilisateur_test.rb
create  test/fixtures/utilisateurs.yml
create  db/migrate
create  db/migrate/001_create_utilisateurs.rb
```

Ruby nous a donc créé :

- **create app/models/utilisateur.rb** : le modèle lui-même
- **create test/unit/utilisateur_test.rb** : un fichier dans lequel nous écrirons nos tests unitaires
- **create test/fixtures/utilisateurs.yml** : un fichier qui nous servira à remplir notre base lors des tests (je l'utiliserai pour remplir la base avec un jeu de données initial)
- **create db/migrate/001_create_utilisateurs.rb** : un fichier de migration, qui va nous servir à créer la table au départ

II-D - Utilisation des fichiers de migration

Les fichiers de migration sont situés dans le répertoire `"db/migrate/"` de votre application.

Ils peuvent servir à créer la structure de vos tables, à la modifier ou à les supprimer.

Les fichiers de migration sont extrêmement puissants, puisqu'ils permettent via une simple ligne de commande de migrer entre les différentes versions de vos bases de données, tout en gardant une traçabilité totale, et ce quel que soit le type de base de données que vous utilisez.

Voici le fichier de migration généré en même temps que le modèle :

fichier de migration '001_create_utilisateurs.rb'

```
class CreateUtilisateurs < ActiveRecord::Migration
  def self.up
    create_table :utilisateurs do |t|
    end
  end

  def self.down
    drop_table :utilisateurs
  end
end
```

Supposons maintenant que nous souhaitons définir un nom et un âge pour un utilisateur.

Modifions le fichier comme suit :

fichier de migration '001_create_utilisateurs.rb'

```
class CreateUtilisateurs < ActiveRecord::Migration
  def self.up
    create_table :utilisateurs do |t|
      t.column "nom", :string, :null => false
      t.column "age", :integer, :null => true
    end
  end

  def self.down
    drop_table :utilisateurs
  end
end
```

Il ne nous reste plus qu'à exécuter ce fichier de migration pour avoir notre table créée à l'aide de la commande suivante : `"rake db:migrate VERSION=1"` (le numéro de version correspond aux trois premiers chiffres de votre fichier de migration).

```
(in mon/path)
== CreateUtilisateurs: migrating =====
-- create_table(:utilisateurs)
   -> 0.1130s
== CreateUtilisateurs: migrated (0.1140s) =====
```

Si vous allez voir la structure de votre table récemment créée, vous vous apercevrez que Rails s'est permis de rajouter un champ `"id"` auto-incrémenté. Ceci est parfaitement normal, Rails a besoin d'un champ nommé `id` auto-incrémenté dans toutes les tables (en respectant les conventions).

Il ne nous reste plus qu'à remplir notre base avec des valeurs par défaut.

II-E - Utilisation des fixtures

Il existe deux types de fichiers pour remplir votre base. Par défaut, rails vous a créé un fichier **yml**. Il est également possible de le renommer en **csv**.

Nous allons voir comment remplir ces deux types de fichier :

utilisateurs.yml

Voici le fichier tel que je l'ai modifié pour remplir ma base avec trois utilisateurs :

utilisateurs.yml

```
utilisateur1: # Un identifiant quelconque pour l'enregistrement en cours
  nom: Riri    # La valeur du champ "nom"
  age: 12      # La valeur du champ "age"

utilisateur2:
  nom: Fifi
  age: 13

utilisateur3:
  nom: Loulou
```

utilisateurs.yml

```
age: 14
```

Ici, je n'ai pas pris la peine de préciser les id. Il peut tout de fois être judicieux de le faire, surtout au cours de tests où vous souhaitez connaître l'id d'un utilisateur précis (cas de jointures par exemple).



Si vous donnez deux fois le même identifiant à deux enregistrements différents, Rails remplacera le premier par le second.

utilisateurs.csv

Voici cette fois un fichier faisant exactement la même chose, mais au format **csv** :

utilisateurs.csv

```
nom, age
Riri, 12
Fifi, 13
Loulou, 14
```

Une fois de plus, je n'ai pas précisé les id.

La première ligne reprend simplement le nom des champs à remplir. Les lignes d'après contiennent les enregistrements, en suivant le "modèle" défini sur la première ligne.

Déploiement des fixtures

Il ne nous reste plus qu'à exécuter ce fichier pour remplir notre table.

Tapez, toujours à la racine de notre application, la ligne de commande suivante : `"rake db:fixtures:load FIXTURES=utilisateurs"`.

Ruby ne vous affichera rien, mais les données sont bien insérées.



Il est possible d'exécuter plusieurs fixtures en même temps. Pour cela, tapez "rake db:fixtures:load FIXTURES=nom_fixture1,nom_fixture2,...".

Cependant, vous ne devez pas avoir d'espaces ni à droite ni à gauche de la virgule, sous peine d'erreur.

III - Les RJS (Remote JavaScript)

Les RJS sont des modèles Javascript destinés à mettre dynamiquement à jour une page.

Ils se placent dans le répertoire "views" de notre application (dans le sous-répertoire du contrôleur correspondant), et prennent simplement l'extension RJS.

Il faut considérer les RJS comme une vue qui met à jour les éléments de la page.

Une action du contrôleur et un RJS peuvent (et doivent dans le cas où le RJS est spécifique à une action) porter exactement le même nom. Dans ce cas, ils s'exécuteront l'un après l'autre, d'abord l'action puis le RJS.

Le fait de passer par le contrôleur peut permettre d'affiner les actions qui seront exécutées dans le RJS.

Voyons tout de suite un petit exemple d'utilisation d'un RJS (ne vous attardez pas sur le code, nous y reviendrons à la fin).

Je me suis basé sur le modèle et le contrôleur créés au chapitre II.

test_controller.rb

```
class TestController < ApplicationController
  def index
    @users = Utilisateur.find(:all)
  end
  def make_bigger
    @users = Utilisateur.find(:all, :conditions=>"age=14")
  end
end
```

index.rhtml

```
<% @body_title = "Premier exemple de RJS"%>
<% @users.each do |user| %>
  <span id="element_<%= user.id %>"><%= user.nom %></span>
  <br />
<% end %>
<hr />
<%= link_to_remote "Faire grossir les items (age = 14)",
  :url=>{:action=>"make_bigger"},
  :before => "Element.show('wait_icon')",
  :complete => "Element.hide('wait_icon')"%>
%>
<br />
<%= image_tag "ajax-loader.gif", :id=>"wait_icon", :style=>"display:none" %>
```

Notez simplement la présence de la variable "**@body_title**" définie dans notre vue qui va nous permettre d'affecter à la vue en cours un titre.

 Vous pouvez **télécharger l'image de chargement ici**.

Placez-la dans le répertoire "public/images/" de votre application.


make_bigger.rjs

```
@users.each { |user|
```



```
make_bigger.rjs
  page.call "new Effect.Scale", "element_#{user.id}", 200
}
```

Vous n'avez plus qu'à lancer votre serveur, à cliquer sur le lien et à admirer le résultat.

 **Attention :** un appel à un RJS est un appel serveur. Qui dit appel serveur dit temps de latence, utilisez-les donc en connaissance de cause.

Dans l'état actuel des choses, si la personne a désactivé Javascript, rien ne se produira lors du click sur le lien. Pour comprendre pourquoi, il suffit de regarder le code source généré :

```
<a href="#" onclick="Element.show('wait_icon'); new Ajax.Request('/test/make_bigger',
{asynchronous:true, evalScripts:true, onComplete:function(request){Element.hide('wait_icon')}});
return false;">Faire grossir les items</a>
<a href="/test/make_bigger">coucou</a>
```

Le href ne pointant sur rien, rien ne se passera dans le cas où javascript est désactivé.

Mais il est possible de gérer ce problème.

Remplacez votre "link_to_remote" par le code suivant :

```
<%= link_to_remote "Faire grossir les items (age = 14)",
  {
    :url=>{:action=>"make_bigger"},
    :before => "Element.show('wait_icon')",
    :complete => "Element.hide('wait_icon')",
  },
  :href=>(url_for :action=>"make_bigger")
%>
```

Ici, nous lui précisons quelle valeur mettre dans l'attribut href de notre lien.

Il ne nous reste plus qu'à gérer dans notre contrôleur le cas où la requête est une requête AJAX, ou bien une simple requête HTML.

Modifiez donc votre action "make_bigger" comme suit :

```
def make_bigger
  respond_to do |requete|
    requete.html {redirect_to :action=>"index"}
    requete.js {
      @users = Utilisateur.find(:all, :conditions=>"age=14")
      render :action=>"make_bigger.rjs"
    }
  end
end
```

Nous gérons désormais dans notre contrôleur les différentes actions à effectuer en fonction du type de requête.

Pour en savoir plus sur comment gérer le cas où Javascript est désactivé, regardez du côté du plugin UJS (Unobtrusive JavaScript).

IV - Quelques petits effets graphiques

Nous allons voir dans ce chapitre quelques effets graphiques forts sympathiques, que nous apporte la librairie **script.aculo.us**.

IV-A - Effets combinés

Fade / Appear / ToggleAppear

Disparition par fondu, apparition par fondu, et alternance entre les deux.

Créons un nouveau contrôleur nommé "effets", puis créez le fichier "index.rhtml" comme suit :

Fade / Appear / ToggleAppear

```
<%= link_to_function("Fade") do |page|
  page.visual_effect(:fade, "bout_de_page")
end
%>
<%= link_to_function("Appear") do |page|
  page.visual_effect(:appear, "bout_de_page")
end
%>
<%= link_to_function("ToggleAppear") do |page|
  page.visual_effect(:toggle_appear, "bout_de_page")
end
%>
<br /><br />
<div id="bout_de_page" style="background-color:#1188FF; width:350px; height:150px;">
  <div>
    <%= image_tag "rails.png"%>
    <p>Bonjour à tous, ceci n'est qu'un petit exemple des possibilités de
    <b>script.aculo.us</b></p>
  </div>
</div>
```

Je sais, les styles définis directement dans le code, on a déjà vu mieux, mais pour l'exemple ce sera amplement suffisant ;)

Il ne vous reste plus qu'à cliquer et à admirer.

Bien entendu, pour faire apparaître un élément, il faut déjà l'avoir fait disparaître. Commencez-donc par cliquer sur "Fade" avant de cliquer sur "Appear".

Pour tous les exemples suivants, nous contenterons d'ajouter de nouveaux liens sous le dernier "*link_to_function*" trouvé.

Le div imbriqué dans le premier n'est là que pour que certains effets passent parfaitement.

Juste à titre d'information, il est également possible d'écrire un "*link_to_function*" comme suit (repreons l'exemple du ToggleAppear) :

```
<%= link_to_function "Hello (local)", update_page { |page|  
  page.visual_effect(:toggle_appear, "bout_de_page")  
}  
%>
```

Ce n'est qu'une question de choix.

Highlight

Mise en valeur visuelle d'un élément.

Highlight

```
<%= link_to_function("Highlight") do |page|  
  page.visual_effect(:highlight, "bout_de_page")  
end  
%>
```

SlideUp / SlideDown / ToggleSlide

Disparition vers le haut, apparition vers le bas, et alternance entre les deux.

SlideUp / SlideDown / ToggleSlide

```
<%= link_to_function("SlideUp") do |page|  
  page.visual_effect(:slide_up, "bout_de_page")  
end  
%>  
<%= link_to_function("SlideDown") do |page|  
  page.visual_effect(:slide_down, "bout_de_page")  
end  
%>  
<%= link_to_function("ToggleSlide") do |page|  
  page.visual_effect(:toggle_slide, "bout_de_page")  
end  
%>
```

Notez simplement ici la séparation des deux mots qui s'est transformée en underscore (valable à chaque fois que vous rencontrerez le cas).

Grow / Shrink

Apparition en zoom, disparition en zoom.

Grow / Shrink

```
<%= link_to_function("Grow") do |page|  
  page.visual_effect(:grow, "bout_de_page")  
end  
%>  
<%= link_to_function("Shrink") do |page|  
  page.visual_effect(:shrink, "bout_de_page")  
end  
%>
```

BlindUp / BlindDown / ToggleBlind

Disparition vers le haut, apparition vers le bas, et alternance entre les deux.

BlindUp / BlindDown / ToggleBlind

```
<%= link_to_function("BlindUp") do |page|
  page.visual_effect(:blind_up, "bout_de_page")
end
%>
<%= link_to_function("BlindDown") do |page|
  page.visual_effect(:blind_down, "bout_de_page")
end
%>
<%= link_to_function("ToggleBlind") do |page|
  page.visual_effect(:toggle_blind, "bout_de_page")
end
%>
```

Pulsate

Faire clignoter un élément.

Pulsate

```
<%= link_to_function("Pulsate") do |page|
  page.visual_effect(:pulsate, "bout_de_page")
end
%>
```

Fold

Disparition vers le haut puis vers la gauche.

Fold

```
<%= link_to_function("Fold") do |page|
  page.visual_effect(:fold, "bout_de_page")
end
%>
```

Puff

Disparition par zoom.

Puff

```
<%= link_to_function("Puff") do |page|
  page.visual_effect(:puff, "bout_de_page")
end
%>
```

Squish

Disparition vers le haut et vers la gauche.

Squish

```
<%= link_to_function("Squish") do |page|
  page.visual_effect(:squish, "bout_de_page")
end
%>
```

SwitchOff

Disparition à la manière de l'extinction des vieux téléviseurs.

SwitchOff

```
<%= link_to_function("SwitchOff") do |page|
  page.visual_effect(:switch_off, "bout_de_page")
end
%>
```

DropOut

Disparition vers le bas.

DropOut

```
<%= link_to_function("DropOut") do |page|
  page.visual_effect(:drop_out, "bout_de_page")
end
%>
```

IV-B - Effets 'de base'

Tout ces effets sont très agréables, mais il est des moments où l'on aimerait se contenter de quelque chose de simple, ou tout simplement qui n'est pas proposé dans ces effets.

Voici comment en réaliser certains :

Insérer / Remplacer / Supprimer un élément

insert_html / update_html / remove

```
<%= link_to_function("Insérer") do |page|
  page.insert_html :bottom, "bout_de_page", "coucou<br />"
end
%>
<%= link_to_function("Remplacer") do |page|
  page.replace_html "bout_de_page", "<p>Le contenu a été remplacé entièrement par du HTML et
une image</p>" + image_tag("rails.png")
end
%>
<%= link_to_function("Supprimer") do |page|
  page.remove "bout_de_page"
end
%>
```

Pour l'insertion, quatre paramètres sont disponibles, et vont définir l'endroit de l'ajout :

- **:top** : ajoute le paramètre dans la balise, tout au début
- **:bottom** : ajoute le paramètre dans la balise, tout à la fin
- **:before** : ajoute le paramètre immédiatement avant la balise
- **:after** : ajoute le paramètre immédiatement après la balise



Pour l'insertion et le remplacement de contenu, il est également possible de passer un partial en paramètre :

```
<%= link_to_function("Remplacer") do |page|
  page.replace_html "bout_de_page", :partial => 'autre_contenu'
end
%>
```

Il vous suffit de créer le partial "`_autre_contenu.rhtml`".

Vous pouvez également passer un objet au partial via le paramètre "**:object**".



Attention : Si vous regardez le code source généré, vous remarquerez que le partial est "embarqué" dans le code JavaScript (puisque aucun appel serveur n'est fait).

Cela peut donc considérablement alourdir le poids de vos pages.

De plus, imaginez que vous faisiez ainsi appel à un partial, qui lui-même à une fonction de ce type pour rappeler le premier partial (par exemple pour changer de formulaire). Et bien vous venez de créer une jolie boucle infinie qui se fera un plaisir de faire exploser votre serveur web. Prudence donc !

Cacher / Afficher un élément / alterner entre les deux

show / hide / toggle

```
<%= link_to_function("Show") do |page|
  page.show "bout_de_page"
end
%>
<%= link_to_function("Hide") do |page|
  page.hide "bout_de_page"
end
%>
<%= link_to_function("Toggle") do |page|
  page.toggle "bout_de_page"
end
%>
```

Agrandir / Réduire un élément

Scale

```
<%= link_to_function("Scale 200%") do |page|
```

Scale

```
page.call "new Effect.Scale", "bout_de_page", 200
end
%>
<%= link_to_function("Scale 50%") do |page|
  page.call "new Effect.Scale", "bout_de_page", 50
end
%>
```

Modifier le style d'un élément

Morph

```
<%= link_to_function("Morph bg-color") do |page|
  page.call "new Effect.Morph", "bout_de_page", :style=>'background-color:#CC55DD;'
end
%>
```

Retarder un effet

Scale

```
<%= link_to_function("Highlight avec deux secondes de décalage") do |page|
  page.delay(2.0) do
    page.visual_effect(:highlight, "bout_de_page")
  end
end
%>
```

L'effet de highlight se produira deux secondes après le clic sur le lien.


Pour effectuer des retards en cascade, il suffit d'imbriquer les *page.delay*.

Exécuter un effet au chargement de la page

Pour exécuter un effet au chargement de la page, il faut utiliser *update_page_tag* dans votre vue :

Scale

```
<%= update_page_tag do |page|
  page.visual_effect(:highlight, "bout_de_page")
end
%>
```

 *Il faut impérativement que le `update_page_tag` soit placé **après** l'élément auquel s'applique l'effet, sous peine de générer une erreur javascript.*

*L'idéal est donc de le placer en dernier, immédiatement avant la fermeture de la balise **BODY**.*

Récapitulatif

Dans tout ce chapitre, nous n'avons pas fait une ligne d'AJAX, l'intérêt étant inexistant en l'état. En effet, qui dit appel AJAX dit appel serveur, totalement inutile pour simplement changer l'aspect d'un élément.

Mais notez que ces effets pourront être utilisés exactement de la même façon depuis un RJS (page.XXX).

Tous ces effets vous serviront lorsque nous attaquerons AJAX, pour agrémenter de jolis effets vos transitions.

Maintenant que vous avez tout ces beaux effets dans vos bagages, vous devriez être capable d'imaginer les combinaisons les plus folles.

Voyons maintenant comment modifier dynamiquement des formulaires.

V - Manipulation de formulaires

V-A - Ajouter dynamiquement un champ à un formulaire en fonction de la valeur d'un autre

Créons un nouveau contrôleur "**my_form**". Remplissons le fichier "*index.rhtml*" comme suit :

index.rhtml

```
<%= form_tag :action=>'index' %>
<label>Sexe :</label>
<%= text_field('profile', 'sexe') %>
<div id="celibataire"></div>
<%= observe_field 'profile_sexe',
      :url=>{:action=>'check_ajout_form'},
      :update=>"celibataire",
      :with => "'sexe=' + escape(value)"
%>
<%= end_form_tag %>
```

Dans la première partie de ce code, nous nous contentons de définir un formulaire, et d'insérer un **div** qui fera office de receveur pour notre appel AJAX.

Le point important ici est le "**observe_field**", qui va nous créer un observateur, chargé de réagir quand un évènement se produira sur notre champ "*profile_sexe*" (ici, la perte de focus).

Nous lui passons en premier paramètre l'id du champ à observer.

Puis, nous définissons l'action à laquelle se référer lors d'un évènement, ainsi que l'id de l'élément à modifier au retour de l'appel AJAX.

Enfin, nous lui passons en paramètre la valeur du champ observé.

Dans notre contrôleur, définissons l'action "*check_ajout_form*" :

check_ajout_form

```
def check_ajout_form
  user_sexe = params[:sexe]
  if user_sexe == "femme"
    render :partial => "celibataire_form"
  else
    render :text => "Les hommes ne m'interessent pas !"
  end
end
```

Enfin, définissons le partiel "*_celibataire_form.rhtml*" :

_celibataire_form.rhtml

```
<label>Etes-vous célibataire ?</label>
<%= text_field('profile', 'celibataire') %>
```

Testez à présent votre contrôleur.

Si vous saisissez "femme" dans votre champ "sexe", un nouveau champ apparait.

Sinon, un 'message d'erreur' vous informe ;)

Le point important ici est le "*render :partial*". Comme l'élément à mettre à jour a déjà été défini dans la vue, nous pouvons donc nous comporter comme si de rien n'était.

V-B - Mettre à jour dynamiquement un champ

Pour cet exemple, nous allons créer deux listes déroulantes, la première se mettant à jour en fonction de la seconde.

Créons deux modèles : *region* et *departement* :

002_create_regions.rb

```
class CreateRegions < ActiveRecord::Migration
  def self.up
    create_table :regions do |t|
      t.column :nom, :string
    end
  end

  def self.down
    drop_table :regions
  end
end
```

003_create_departements.rb

```
class CreateDepartements < ActiveRecord::Migration
  def self.up
    create_table :departements do |t|
      t.column :region_id, :integer
      t.column :num_dept, :string
      t.column :nom, :string
    end
  end

  def self.down
    drop_table :departements
  end
end
```

Il ne nous reste plus qu'à exécuter notre migration.

Remplissons maintenant nos deux tables à l'aide des deux fichiers de fixtures suivant :

regions.yml

```
alsace:
  id: 1
  nom: Alsace
aquitaine:
  id: 2
  nom: Aquitaine
auvergne:
  id: 3
  nom: Auvergne
bassenormandie:
  id: 4
  nom: Basse-Normandie
bourgogne:
```

regions.yml

```
  id: 5
  nom: Bourgogne
bretagne:
  id: 6
  nom: Bretagne
centre:
  id: 7
  nom: Centre
champagneardenne:
  id: 8
  nom: Champagne-Ardenne
corse:
  id: 9
  nom: Corse
dom:
  id: 10
  nom: Départements d'Outre-Mer
franchecombe:
  id: 11
  nom: Franche-Comté
hautenormandie:
  id: 12
  nom: Haute-Normandie
idf:
  id: 13
  nom: Ile-de-France
languedocroussillon:
  id: 14
  nom: Languedoc-Roussillon
limousin:
  id: 15
  nom: Limousin
lorraine:
  id: 16
  nom: Lorraine
midipyrenees:
  id: 17
  nom: Midi-Pyrénées
nordpasdecalais:
  id: 18
  nom: Nord-Pas-de-Calais
paysdelaloire:
  id: 19
  nom: Pays de la Loire
picardie:
  id: 20
  nom: Picardie
poitoucharentes:
  id: 21
  nom: Poitou-Charentes
paca:
  id: 22
  nom: Provence-Alpes-Côte-d'Azur
rhonealpes:
  id: 23
  nom: Rhône-Alpes
tom:
  id: 24
  nom: Territoires d'Outre-Mer
```

departements.csv

```
region_id, num_dept, nom
1, 67, Bas-Rhin
1, 68, Haut-Rhin
2, 24, Dordogne
```

departements.csv

```
2, 33, Gironde
2, 40, Landes
2, 47, Lot-et-Garonne
2, 64, Pyrénées-Atlantiques
3, 03, Allier
3, 15, Cantal
3, 43, Haute-Loire
3, 63, Puy-de-Dôme
4, 14, Calvados
4, 50, Manche
4, 61, Orne
5, 21, Côte-d'Or
5, 58, Nièvre
5, 71, Saône-et-Loire
5, 89, Yonne
6, 22, Côtes-d'Armor
6, 29, Finistère
6, 35, Ille-et-Vilaine
6, 56, Morbihan
7, 18, Cher
7, 28, Eure-et-Loir
7, 36, Indre
7, 37, Indre-et-Loire
7, 41, Loir-et-Cher
7, 45, Loiret
8, 08, Ardennes
8, 10, Aube
8, 51, Marne
8, 52, Haute-Marne
9, 2A, Corse-du-Sud
9, 2B, Haute-Corse
10, 971, Guadeloupe
10, 972, Martinique
10, 973, Guyane
10, 974, La Réunion
10, 975, Saint-Pierre-et-Miquelon
10, 976, Mayotte
11, 25, Doubs
11, 39, Jura
11, 70, Haute-Saône
11, 90, Territoire de Belfort
12, 27, Eure
12, 76, Seine-Maritime
13, 75, Paris
13, 77, Seine-et-Marne
13, 78, Yvelines
13, 91, Essonne
13, 92, Hauts-de-Seine
13, 93, Seine-Saint-Denis
13, 94, Val-de-Marne
13, 95, Val-d'Oise
14, 11, Aude
14, 30, Gard
14, 34, Hérault
14, 48, Lozère
14, 66, Pyrénées-Orientales
15, 19, Corrèze
15, 23, Creuse
15, 87, Haute-Vienne
16, 54, Meurthe-et-Moselle
16, 55, Meuse
16, 57, Moselle
16, 88, Vosges
17, 09, Ariège
17, 12, Aveyron
17, 31, Haute-Garonne
17, 32, Gers
```

departements.csv

```
17, 46, Lot
17, 65, Hautes-Pyrénées
17, 81, Tarn
17, 82, Tarn-et-Garonne
18, 59, Nord
18, 62, Pas-de-Calais
19, 44, Loire-Atlantique
19, 49, Maine-et-Loire
19, 53, Mayenne
19, 72, Sarthe
19, 85, Vendée
20, 02, Aisne
20, 60, Oise
20, 80, Somme
21, 16, Charente
21, 17, Charente-Maritime
21, 79, Deux-Sèvres
21, 86, Vienne
22, 04, Alpes-de-Haute-Provence
22, 05, Hautes-Alpes
22, 06, Alpes-Maritimes
22, 13, Bouches-du-Rhône
22, 83, Var
22, 84, Vaucluse
23, 01, Ain
23, 07, Ardèche
23, 26, Drôme
23, 38, Isère
23, 42, Loire
23, 69, Rhône
23, 73, Savoie
23, 74, Haute-Savoie
24, 984, Terres Australes et Antarctiques
24, 986, Wallis et Futuna
24, 987, Polynésie Française
24, 988, Nouvelle-Calédonie
```

Exécutons maintenant ces deux fichiers, et allez vérifier que vos tables sont bien remplies.

Notez que j'ai pris la peine de préciser l'id dans *"regions.yml"*, afin d'être sûr de ma cohérence avec *"departements.csv"*.

Créons maintenant un contrôleur *"maj_liste"*.

Il ne nous reste plus qu'à éditer les fichiers suivants :

maj_liste_controller.rb

```
class MajListeController < ApplicationController
  def index
    @liste_region = Region.find :all
    @liste_dept = Departement.find :all, :order=>"nom asc"
  end

  def update
    choix_region = params['id_region']
    @liste_dept = Departement.find_all_by_region_id(choix_region, :order => "nom asc")
  end
end
```

index.rhtml

index.rhtml

```
<%= form_tag :action=>"index" %>
<label>Région :</label>
<%= select("profile", "region", @liste_region.collect {|elt| [ elt.nom, elt.id ] }) %>
<%= image_tag "ajax-loader.gif", :id=>"wait_icon", :style=>"display:none" %>
<br />
<label>Département :</label>
<%= select("profile", "departement", @liste_dept.collect {|elt| [ elt.nom, elt.id ] }) %>
<br />
<%= submit_tag "valider"%>
<%= end_form_tag%>

<div id="result_validation"></div>

<%= observe_field 'profile_region',
  :url=>{:action=>'update'},
  :before => "Element.toggle('wait_icon')",
  :complete => "Element.toggle('wait_icon')",
  :with => "'id_region=' + escape(value)"
%>
```

update.rjs

```
# On commence par supprimer le contenu de "profile_departement"
page.replace_html "profile_departement", ""

# Puis on le remplit avec les nouvelles valeurs
@liste_dept.each { |dept|
  page.insert_html :bottom,
    "profile_departement",
    content_tag('option', dept.nom, :value => dept.id )
}
```

Il ne vous reste plus qu'à tester votre contrôleur. Le fait de changer la région dans la liste met automatiquement à jour la liste déroulante contenant les départements.

Cette fois encore, notez l'utilisation du "*observe_field*" qui nous permet de détecter un changement de valeur dans la liste des départements et de mettre à jour nos départements.

Certaines des choses présentes dans le fichier "*index.rhtml*" ne nous ont pas encore servi, mais comme elles vont nous servir dans le prochain chapitre, j'ai préféré les intégrer dès à présent.

Notez dans le "**observe_field**" l'utilisation des paramètres "*:before*" et "*:complete*", qui permettent d'effectuer des actions à différents moments.

Ce ne sont pas les deux seuls, voici donc une liste complète :

- **:before** : juste avant l'appel AJAX
- **:after** : juste après le déclenchement de l'appel AJAX
- **:loading** : pendant qu'XHR reçoit les données
- **:loaded** : quand XHR a fini de recevoir les données
- **:interactive** : pendant qu'XHR traite les données
- **:success** : les données sont finies de traiter, et il n'y a pas d'erreur
- **:failure** : les données sont finies de traiter, et il y a une erreur
- **:complete** : les données ont été renvoyées au navigateur, et **:success** ou **:failure** ont été appelés

Il est agréable de mettre à jour la liste dynamiquement, mais si en plus elle pouvait se mettre à jour automatiquement au chargement de la page, ce serait le paradis.

C'est là que va intervenir la variable "**@body_extras**" que nous avons défini dans l'introduction de ce cours.

Rajoutez la ligne suivante tout en haut de votre fichier "*index.rhtml*" :

```
<% @body_extras = ' onload=' + remote_function(:url=>{:action=>"update"}, :with=>"'id_region=' +  
$('profile_region').value" ) + ' ' %>
```

Cela va nous permettre au chargement de la page (**onload**) d'effectuer un appel AJAX à une action (**:url**), en lui précisant les paramètres que nous souhaitons lui transmettre (**:with**).

Au sein du **:with**, la chose essentielle à retenir est le "*\$('profile_region').value*", qui va nous permettre de récupérer la valeur de l'élément ayant pour id "*profile_region*".

V-C - Validation dynamique d'un formulaire

Il existe deux méthodes aux effets quasiment identiques pour y arriver :

form_remote_tag

Dans votre page "*index.rhtml*", modifiez le tag d'ouverture de votre formulaire comme suit :

```
<%= form_remote_tag :url=>{:action=>"valide_form"} %>
```

Ensuite, ajoutez cette action à votre contrôleur :

```
def valide_form  
  render(:update) { |page|  
    page.replace_html "result_validation", "Le formulaire a été soumis avec les paramètres  
suivants :<br />#{params.inspect}"  
  }  
end
```

Il ne vous reste plus qu'à recharger votre page, et à valider le formulaire pour découvrir le résultat.

La balise **form_remote_tag** vous permet de faire un appel AJAX en passant à l'action de destination l'ensemble des valeurs du formulaire sur la validation de celui-ci.

Notez également que dans ce cas précis, je n'ai pas utilisé de RJS, je me suis contenté de définir ma mise à jour dans mon action.

Ces deux méthodes sont techniquement équivalentes, mais dans un souci de lisibilité du contrôleur, et de respect du MVC, il est souvent préférable de définir un RJS chargé de la mise à jour, surtout s'il y a plus d'une action à effectuer (de plus, il est possible de définir des helpers RJS qui nous simplifient encore plus la vie).

submit_to_remote

Rien de très compliqué cette fois encore, nous allons simplement modifier deux petites choses dans notre fichier `index.rhtml`.

Remplacez la balise **form_remote_tag** par le code suivant :

```
<%= form_tag %>
```

Et remplacez également la balise **submit_tag** par le code suivant :

```
<%= submit_to_remote "id_bouton", "valider", :url=>{:action=>"valide_form"}%>
```

Encore une fois, rafraîchissez votre navigateur, et validez le formulaire pour apprécier le résultat.

VI - Auto-complétion

Comme depuis le début de ce cours, rien de très difficile dans l'auto-complétion.

Créons un nouveau contrôleur nommé "*autocomp*" (original non ?).

Éditons maintenant notre contrôleur et la vue associée comme suit :

autocomp_controller.rb

```
class AutocompController < ApplicationController
  auto_complete_for :departement, :nom
end
```

index.rhtml

```
<p><b>Auto-complete AJAX en se basant sur un modèle</b></p>
<label>Saisissez un nom de département :</label>
<%= text_field_with_auto_complete :departement, :nom %>
<hr>
```

Pour cet exemple, nous nous basons sur notre modèle "Département", et son champ "nom".

En procédant ainsi, ces simples lignes suffisent, à partir du moment où les helpers "**auto_complete_for**" et "**text_field_with_auto_complete**" prennent en paramètre respectivement le nom du modèle et le nom du champ.

Par défaut, "**auto_complete_for**" va limiter le nombre de résultats à dix. Vous pouvez modifier ce comportement en rajoutant un troisième argument, "**:limit**".

Vous pouvez également ajouter un autre paramètre, "**:order**" qui va vous permettre d'organiser vos résultats.

L'intégration de ces paramètres dans notre contrôleur pourra donner par exemple :

```
auto_complete_for :departement, :nom, :limit=>15, :order=>'region_id desc'
```

Vous pouvez également choisir d'appeler une action personnalisée plutôt que d'attaquer directement le modèle.

Voyons de suite un exemple :

A ajouter à autocomp_controller.rb

```
def auto_complete_for_liste_departements
  @liste_dept = Departement.find(:all, :conditions=>"nom LIKE
  '%#{params[:liste][:departements]}%'" )
  render :partial => 'complete_ajax_own_function'
end
```

A ajouter à index.rhtml

```
<p><b>Auto-complete AJAX en appelant sa propre fonction</b></p>
<label>Saisissez un nom de département :</label>
<%= text_field_with_auto_complete :liste, :departements %>
<hr>
```

Créons également le partial associé :

_complete_ajax_own_function.rhtml

```
<ul>
  <% @liste_dept.each do |dep| %>
    <li><%=h dep.nom %></li>
  <% end %>
</ul>
```

Cette fois-ci, les paramètres de "*text_field_with_auto_complete*" ne correspondent à aucun modèle. Il nous faut donc définir une action dans le contrôleur nommée "**auto_complete_for_[param1]_[param2]**" (sans les crochets).

Du coup, c'est à nous de définir notre mode de recherche, et de gérer la limite ainsi que l'ordre, mais cela peut nous permettre d'affiner nos besoins.

VII - Tri d'une liste d'éléments

Créons un nouveau contrôleur nommé **"list"**, et éditons les fichiers comme suit :

list_controller.rb

```
class ListController < ApplicationController
  def index
    @liste_dept = Departement.find(:all, :limit=>10)
  end

  def voir_tri
    render(:update) { |page|
      page.replace_html "informations", params[:liste_departements].inspect
    }
  end
end
```

index.rhtml


```
<ul id="liste_departements">
  <% @liste_dept.each do |dpt| -%>
    <li id="item_<%= dpt.id %>"><%= dpt.nom %></li>
  <% end -%>
</ul>

<div id="informations"></div>

<%= sortable_element 'liste_departements', :url=>{ :action => "voir_tri" } %>
```

Le fait de cliquer sur un élément et de le déplacer dans la liste change sa position, ainsi que la valeur renvoyée à l'action **"voir_tri"**.

Vous pouvez désormais traiter le tableau d'éléments reçus comme bon vous semble, l'enregistrer dans une base de données par exemple (éventuellement avec **"act_as_sortable"**).

 **Les éléments de votre liste doivent obligatoirement avoir pour id "[une_chaine]_identifiant_récupéré]" (sans les crochets).**

Tout ceci est fort joli, mais placer le **"sortable_element"** dans la vue ne permet pas de rajouter un élément dynamiquement et de le trier immédiatement.

Voyons de suite un exemple en rajoutant l'insertion d'un élément à notre vue :

A ajouter à index.rhtml

```
<%= link_to_function("Ajouter un élément") do |page|
  page.insert_html :bottom, "liste_departements", content_tag('li', "Nouvel élément", :id
=>"item_42" )
end
%>
```

Lors d'un click sur le lien, un nouvel élément est bien ajouté, mais il est impossible de déplacer ce nouvel élément.

Pour que ce nouvel élément soit également classable, il faut re-préciser la liste à trier en rajoutant ce code à notre **"link_to_function"**, juste sous le **"page.insert_html"** :

```
page.sortable 'liste_departements', :url=>{ :action => "voir_tri" }
```

Vous avez maintenant une liste avec un élément de plus, entièrement triable.

Dernier petit problème à soulever, comment récupérer la valeur de la liste à un moment donné, sans avoir besoin de la classer ?

Voici un lien qui va vous permettre de l'obtenir :

A ajouter à index.rhtml

```
<%= link_to_remote "Voir le classement de la liste",  
  :url=>{:action=>"voir_classement_liste"},  
  :with=>"val_liste=" + escape(Sortable.serialize('liste_departements'))"  
%>
```

A ajouter à list_controller.rb

```
def voir_classement_liste  
  render(:update) { |page|  
    page.replace_html "informations", "Les valeurs de la liste sont : #{params['val_liste']}"  
  }  
end
```

Vous pouvez maintenant à tout instant connaître l'état de votre liste, bien que le format dans lequel les données sont renvoyées ne soit pas le même que lors d'un changement de la liste. A vous de le gérer ;)

Si vous souhaitez une présentation plus "sexy" des éléments de la liste, vous pouvez également employer la méthode que nous utiliserons dans le prochain chapitre.

VIII - Drag & Drop is easy

Nous allons à présent créer plein de petites boîtes, et faire passer de l'une à l'autre plein de petits éléments, le tout sans trop nous fouler.

Pour l'exemple, nous allons créer trois "boîtes". La première pourra "donner" des éléments à la seconde, la seconde à la troisième, et la troisième à la première.

Afin que cet exemple soit le plus visuel possible, nous allons créer une feuille de style "**style.css**" à placer dans le répertoire `/public/stylesheets` de notre application :

style.css

```
#box1 {
  width:200px;
  height:200px;
  background-color:#1188FF;
  float:left;
  margin-left:10px;
}
#box2 {
  width:200px;
  height:200px;
  background-color:#88FF11;
  float:left;
  margin-left:10px;
}
#box3 {
  width:200px;
  height:200px;
  background-color:#FF1188;
  float:left;
  margin-left:10px;
}
.clear {
  clear:both;
}
```

Pour que notre feuille de style soit prise en compte, il va nous falloir éditer notre fichier `"application.rhtml"` en rajoutant cette ligne dans la balise **head** :

```
<%= stylesheet_link_tag("style") %>
```

Créons maintenant un contrôleur nommé `drag_drop`, puis éditons les fichiers comme suit :

drag_drop_controller.rb

```
class DragDropController < ApplicationController
  def index
    @mes_items = [1, 2, 3, 4]
  end
  def make_draggable_init
    @mes_items = [1, 2, 3, 4]
    render :action => "make_draggable_init"
  end
end
```

index.rhtml

```
<% @body_extras = ' onload="' + remote_function(:url=>{:action=>'make_draggable_init'}) + '"' %>
<p>Les boites ne peuvent recevoir les éléments que dans cet ordre :</p>
<ul>
  <li>Boite1 => Boite2</li>
  <li>Boite2 => Boite3</li>
  <li>Boite3 => Boite1</li>
</ul>

<div id="box1">
  <p><b>Boite 1 :</b></p>
  <% @mes_items.each do |elt| %>
    <span id="element_<%= elt%>" class="boite1">Element_<%= elt%><br /></span>
  <% end %>
</div>
<div id="box2">
  <p><b>Boite 2 :</b></p>
</div>
<div id="box3">
  <p><b>Boite 3 :</b></p>
</div>
<div class="clear"></div>

<div id="valeur_boite"></div>
<%= image_tag "ajax-loader.gif", :id=>"wait_icon", :style=>"display:none" %>

<%= drop_receiving_element "box1",
  :url=>{:action=>"ajoute_elt_box", :target=>"box1", :class=>"boite1"},
  :before => "Element.toggle('wait_icon')",
  :complete => "Element.toggle('wait_icon')",
  :accept=>'boite3'
%>
<%= drop_receiving_element "box2",
  :url=>{:action=>"ajoute_elt_box", :target=>"box2", :class=>"boite2"},
  :before => "Element.toggle('wait_icon')",
  :complete => "Element.toggle('wait_icon')",
  :accept=>'boite1'
%>
<%= drop_receiving_element "box3",
  :url=>{:action=>"ajoute_elt_box", :target=>"box3", :class=>"boite3"},
  :before => "Element.toggle('wait_icon')",
  :complete => "Element.toggle('wait_icon')",
  :accept=>'boite2'
%>
```

make_draggable_init.rjs

```
@mes_items.each { |elt|
  page.draggable "element_#{elt}", :revert=>true
}
```

ajoute_elt_box.rjs

```
page.remove params[:id]

page.insert_html :bottom,
  params[:target],
  "<span id='#{params[:id]}' class='#{params[:class]}'>#{params[:id].capitalize}<br /></span>"

page.draggable params[:id], :revert=>true
```

Je sais, ça fait beaucoup d'un coup, mais en fait, rien de compliqué.

Nous nous contentons d'afficher dans notre vue trois div différents. Dans le premier, nous plaçons nos éléments.

Ce qui rajoute énormément au code est en fait quasiment trois fois la même chose : nous déclarons pour chacune des boîtes que celles-ci seront désormais considérées comme des receveurs.

Pour chacune des boîtes, nous précisons l'action à exécuter (lors de la réception d'un élément), les paramètres à transmettre à cette action, d'afficher une image pendant le chargement puis de la masquer, et enfin quelles classes CSS accepter.

Attention, il s'agit ici bien de la classe de l'élément, et non pas de son identifiant.

Notez également la présence de "**@body_extras**" (défini lors de l'introduction dans "*application.rhtml*"), qui va nous permettre au chargement de la page de rendre l'ensemble des éléments de la boîte 1 glissable grâce à l'action et au RJS "**make_draggable_init**".

Dans le RJS, le paramètre "*:revert=>true*" permet aux éléments de revenir en place lorsqu'ils sont relâchés. Si nous l'avions omis, il seraient restés à l'endroit où ils ont été relâchés, y compris hors des boîtes.

Penchons-nous maintenant sur le RJS "*ajoute_elt_box*".

Comme vous le constatez, trois actions sont effectuées : la suppression de l'élément, la création d'un nouvel élément dans la boîte cible, puis le fait de rendre déplaçable ce nouvel élément.

On pourrait imaginer que sans le paramètre "**:revert**", la suppression serait inutile, mais c'est se tromper : Script.aculo.us se contente de masquer l'élément.

La suppression est donc indispensable pour s'assurer de la cohérence de nos id.

Notons également que lors d'un déposé sur une boîte, l'id de l'élément déposé est automatiquement transmis.

Mais comment récupérer le contenu d'une boîte ?

Il n'existe pas de méthode "**serialize**" pour les Drag & Drop, comme il en existe un pour les listes triables.

Nous allons donc devoir faire autrement. Voici l'astuce (après plus d'une semaine à courir après...) :

A rajouter à index.rhtml

```
<%= link_to_remote "Voir la valeur de la boîte 1",
  :url=>{:action=>"voir_valeurs"},
  :with=>"val_liste=" + Element.findChildren($('box1'), null, false,
'span').map(function(elt){return elt.id;})"
%>
<%= link_to_remote "Voir la valeur de la boîte 2",
  :url=>{:action=>"voir_valeurs"},
  :with=>"val_liste=" + Element.findChildren($('box2'), null, false,
'span').map(function(elt){return elt.id;})"
%>
<%= link_to_remote "Voir la valeur de la boîte 3",
  :url=>{:action=>"voir_valeurs"},
  :with=>"val_liste=" + Element.findChildren($('box3'), null, false,
'span').map(function(elt){return elt.id;})"
%>
```

L'idée est de récupérer via le DOM tous les enfants de chaque boîte, qui seront renvoyés sous la forme d'un tableau d'éléments, non exploitable directement avec Rails.

Il faut donc le mapper en JavaScript (qui fonctionne sur le même principe que la fonction map de Ruby) pour obtenir un tableau exploitable depuis Rails.

Pensez tout de même à rajouter dans votre contrôleur l'action qui va avec :

```
def voir_valeurs
  render(:update) { |page|
    page.replace_html "valeur_boite", params['val_liste'].inspect
  }
end
```


IX - Petit mélange des deux

Pour conclure ce tutorial sur l'utilisation d'AJAX avec Rails, nous allons réaliser un dernier exemple en mixant la liste triable et le Drag & Drop : une liste triable dont les éléments peuvent être mis dans une corbeille.

Si vous le souhaitez, vous pouvez [télécharger une image de corbeille ici](#).

Comme pour l'image d'attente, placez-la dans le répertoire "*public/images/*" de votre application.

Nous irons plutôt vite pour cet exemple, vous devriez normalement assimiler la totalité des concepts.

Créons un contrôleur nommé "**list_trash**", puis éditons les fichiers comme suit :

list_trash_controller.rb

```
class ListTrashController < ApplicationController
  def index
    @liste_region = Region.find :all, :limit=>8
  end

  def delete_element
    # Récupère les informations de l'élément à supprimer
    infos_element = params[:id]

    render(:update) { |page|
      page.remove(infos_element)
    }
  end

  def save_liste
    # Récupère les valeurs de la liste et les formate
    # ex : "ma_liste[]=1&ma_liste[]=2&ma_liste[]=4"
    #      => ["1", "2", "4"]
    vals_liste = params[:val_liste]
    vals_liste = vals_liste.gsub!('ma_liste[]=', '').split('&') unless vals_liste.empty?

    render(:update) { |page|
      page.replace_html "list-info", "Les valeurs sauvegardées seraient les suivantes :<br />#{vals_liste.inspect}"
    }
  end
end
```

index.rhtml

```
<% @body_extras = ' onload="' + remote_function(:url=>{:action=>'make_sortable'}) + '" %>

<ul id="ma_liste">
  <% @liste_region.each do |region| %>
    <li id="item_<%= region.id %>"><%= region.nom %></li>
  <% end %>
</ul>
<br />
<%= image_tag "trash.png", :id=>"poubelle", :width=>"64px", :height=>"64px" %>
<%= image_tag "ajax-loader.gif", :id=>"wait_icon", :style=>"display:none" %>
<br />
<%= drop_receiving_element "poubelle",
  :url=>{:action=>"delete_element"},
  :before => "Element.show('wait_icon')",
  :complete => "Element.hide('wait_icon')"%>
```

index.rhtml

```
<%= link_to_remote "Sauvegarder la liste",  
  :url=>{:action=>"save_liste"},  
  :before => "Element.show('wait_icon')",  
  :complete => "Element.hide('wait_icon')",  
  :with=>"val_liste=" + escape(Sortable.serialize('ma_liste'))"  
%>  
  
<p id="list-info"></p>
```

make_sortable.rjs

```
page.sortable 'ma_liste', :complete => visual_effect(:highlight, 'list')
```

En fait, les éléments d'une liste triable sont déjà compatibles avec les receveurs du Drag & Drop.

Il suffit donc de créer un receveur, et de créer les bonnes actions pour obtenir de très jolis effets

X - Allons plus loin

X-A - Créer ses propres helpers

Il est tout à fait possible de créer ses propres helpers pour la manipulation du document.

Voyons tout de suite un petit exemple simple.

Retournons dans notre contrôleur nommé "effets", et éditons comme suit le helper qui lui est associé "effets_helper.rb", situé dans le répertoire "app/helpers" de votre application :

effets_helper.rb

```
module EffetsHelper
  def make_big_and_green(target)
    page.call "new Effect.Scale", target, 200
    page.call "new Effect.Morph", target, :style=>'color:#00FF00;'
  end
end
```

Il ne nous reste plus qu'à rajouter dans notre fichier "index.rhtml" le code suivant :

index.rhtml

```
<%= link_to_function("Gros et vert") do |page|
  page.make_big_and_green("bout_de_page")
end
%>
```

Il va ainsi être possible de regrouper des morceaux de code pour les appeler depuis vos RJS en minimisant au maximum la redondance.

X-B - Créer ses propres fonctions Javascript

Revenons maintenant sur la manière de récupérer les valeurs d'une boîte dans le chapitre VIII.

Rappelez-vous :

```
<%= link_to_remote "Voir la valeur de la boîte 3",
  :url=>{:action=>"voir_valeurs"},
  :with=>'val_liste=' + Element.findChildren($('box3'), null, false,
  'span').map(function(elt){return elt.id;})"
%>
```

Pas très sexy ce **":with"** tout de même, d'autant qu'il est utilisé trois fois dans la page, et que nous aurions bien envie de l'utiliser aussi pour récupérer les valeurs de la liste triable.

Pour simplifier un peu toute cette syntaxe, nous allons éditer le fichier "*public/javascripts/application.js*" comme suit :

application.js

```
// Place your application-specific JavaScript functions and classes here
// This file is automatically included by javascript_include_tag :defaults
```

application.js

```
function get_id_of_childs(element_id, child_elt_type) {  
  // Nous sommes obligés de procéder comme suit pour IE  
  // pour gérer le cas où l'élément n'a pas d'enfants  
  
  my_elt = document.getElementById(element_id);  
  valeurs = Element.findChildren(my_elt, null, false, 'li');  
  if (valeurs == null) {  
    valeurs = [];  
  } else {  
    valeurs = valeurs.map(function(elt){return elt.id;});  
  }  
  return valeurs  
}
```

Il ne nous reste plus qu'à tester en insérant une nouvelle fonction dans notre fichier "*index.rhtml*" :

```
<%= link_to_remote "Voir la valeur de la boîte 3 - simple",  
  :url=>{:action=>"voir_valeurs"},  
  :with=>"val_liste=" + get_id_of_childs('box3', 'span')"  
%>
```

Avouez que la nouvelle version est nettement plus agréable à manipuler et facile à maintenir ;)

XI - Conclusion, remerciements et liens complémentaires

Vous avez maintenant toutes les clés en mains pour créer de belles interfaces fonctionnelles.

Si vous avez des questions ou des suggestions sur cet article, **contactez-moi**, ou posez votre question sur **le forum Ruby / Rails**.

Un énorme merci à **bolo** pour sa relecture technique, à **SpaceFrog** pour son aide en Javascript et à l'équipe de rédaction de developpez.com pour la relecture orthographique.

Et maintenant quelques petits liens qui vous permettront d'aller plus loin dans les points abordés dans cet article :

- Le blog de bolo : **Comment importer des fichier csv directement depuis un fichier de migration**
- La traduction de la **documentation de la librairie prototype, version 1.4.0**
- Le site officiel de **script.aculo.us**
- Le **plugin ARTS (Another RJS Testing System)**, qui vous permet de tester vos RJS

