Javascript

- Langage à prototype
- Langage interprété, s'exécute dans une machine virtuelle
- embarqué dans les navigateurs web
- syntaxe proche de C/Java
- surprises syntaxiques : signification de `this`, portée des variables => langage piégeux avec ses ficelles à connaître. Lire ce chapitre du livre sur node.js pour comprendre plus finement le problème.

Référence : Eloquent Javascript

navigateur :balise script dans code html

```
<script type="text/javascript"> code javascript </script>
<script type="text/javascript" src="/path/to/file.js" />
```

Le code javascript est évalué dans le contexte de la page HTML : DOM, Events

node.js

node.js permet l'exécution de code javascript hors d'un navigateur.

Séparer le code javascript du reste de la page

- contenu : html
- style : css
- comportement : javascript

Unobtrusive Javascript (UJS) : le javascript se greffe sur une page HTML sans modifier son contenu, en s'appuyant sur des marqueurs, de la même manière que les styles CSS.

AJAX : XMLHttpRequest

Ajax : un évènement javascript génère l'exécution d'une requête HTTP, dont le résultat va pouvoir modifier la page courante, sans rechargement complet de la page.

Cela permet d'obtenir des interfaces plus réactives.

```
1  var req = new XMLHttpRequest(); # object
2  req.open("GET", "/path/to/resource", true); # last arg : asynchronous or not ?
3  req.send(); # req.send(body)
4
```

cycle de l'appel

- états successifs dans `XMLHttpRequest#readyState`
- 4 : appel terminé

callback à positionner :

```
XMLHttpRequest#onreadystatechange = function() { //do stuff based on readyState value }
```

La fonction est appelée à chaque changement de valeur de `XMLHttpRequest#readyState`

À 4, l'appel est terminé.

XMLHttpRequest : propriétés

onreadystatechange: The callback function that's notified of state changes.
readyState: State within the request cycle.
responseText: The response from the server, as a String.
responseXML: The response from the server, as a Document Object Model, provided that the response was valid XML.
status: HTTP response code received from the server, e.g "404".
statusText: HTTP response code description received from the server, e.g. "Not Found".

XMLHttpRequest : methodes

abort(): Stops the request and resets its readyState back to zero.
getAllResponseHeaders(): Returns a string of all response headers, separated by a newline as in the original message.
getResponseHeader(headerField): Returns the value for a particular header field.
open(requestMethod, url, asynchronousFlag, username, password): Prepares XMLHttpRequest.
  Only the first two parameters are required. username and password can be used for authentication.
send(bodyContent): Sends the message along with specified body content (null if no body content is to be sent, e.g. for GET requests).
setRequestHeader(headerField, headerValue): Sets a request header.

jQuery

jQuery est une librairie javascript minimaliste, permettant de :

- sélectionner des éléments dans la page avec un sélecteur CSS . `$("form#search input")` : bien adapté à de l'UJS
- manipuler des éléments de la page : `$("div#result").html("nouveau contenu")`
- gérer des évènements javascript : clavier, souris ...
- faire de l'ajax jQuery.ajax
- et d'autres choses ...

exemple get : formulaire de recherche et remplissage du résultat

```
<input id="searchbox" type="text" value="Message à rechercher"/>

<div id="result"/>

<script type="text/javascript">
  callsearch = funtion(s) {
    $.ajax({
      type: "GET",
      url: "/posts",
      data: "search="+s
    }).done(function(html_result){
      $("#result").html(html_result);
    });
  }
  $("#searchbox").keydown(function(event) {
    if (event.keyCode == '13') {
      event.preventDefault();
      callsearch($(this).val());
    }
  }
</script>
```

rails et javascript

Depuis rails version 3.0, rails marque les éléments (formulaire et lien) à gérer par des requêtes Ajax.

marquage HTML pour UJS :

- data-method : la méthode REST pour les requêtes HTTP a générer, positionnée par `form_tag` et `link_to` : `<%= form_tag …, .method => … %>`
- data-confirm : le message de confirmation avant certaines acitons
- data-remote : si true, soumettre avec AJAX, utilisé dans `link_to` et `form_tag`
- data-disable-with: désactiver un élément d'un formulaire

exemple :

```
<%= link_to "destroy", article_path(@article), :method => :delete, :remote => true %>
```

donne

```
<a href="/articles/1" data-method="delete" data-remote="true" rel="nofollow">destroy</a>
```

À partir de ce marquage, la librairie fournie par rails (qui s'appuier sur jQuery) va génerer les requêtes Ajax.

Principe global de fonctionnement

1. L'application génère une page html contenant du javascript pouvant générer une requête Ajax.
2. L'utilisateur interagit avec la page, la requête Ajax est générée.
3. L'appli rails reçoit la requête, une action d'un contrôleur la traite. Un template javascript est appelé pour générer du code javascript, qui est retourné au navigateur client.
4. Le code javascript est exécuté sur la navigateur, dans le contexte de la page html initiale.

contrôleur : gérer la réponse à une requête AJAX

```
def index
  @posts = Post.all
  respond_to do |format|
    format.html # index.html.erb
    format.js # index.js.erb
  end
end
```

Les requêtes Ajax sont repérés par un entête spécial dans la requête HTTP :

```
X-Requested-With: XMLHttpRequest
```

Cet entête permet au serveur rails de détecter le format javascript, et de le rendre accessible dans les contrôleurs dans un bloc `respond_to`.

Vues javascript

On peut utiliser un template pour générer du javascript, il suffit d'avoir une extension `.js.erb`. Le contenu de ce fichier doit être du javascript. Pour produire du contenu avec du code, il faut utiliser la méthode `escape_javascript()` qui permet de formatter correctement une chaîne de caractères.