AJAX sous jQuery

1. En écriture concise : load()

Le framework jQuery a quasi réinventé l'écriture du code AJAX. Plus besoin de XMLHttpRequest et de ses complexités, jQuery nous offre un code d'une concision extrêm et facile à mettre en œuvre.

load(url, données, fonction de rappel);

Charge des données du serveur et les inclut dans l'élément sélectionné.

- `url` : une chaîne de caractères contenant l'URL du fichier à charger.
- `données` (optionnel) : spécifie des données à envoyer vers le serveur en même temps que la requête.
- `fonction de rappel` (optionnel) : une fonction à exécuter lorsque le processus est terminé.

```
$("bouton").click(function(){
$("#div1").load("demo.txt");
});
```

Au clic sur un bouton, les données du fichier demo.txt, situées sur le serveur, sont chargées dans la division identifiée par `div1`. D'une simplicité absolue !

Exemple

Affichons dans une division un texte situé sur le serveur par la méthode `load()`.

Le fichier à charger (lorem.htm) :

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer
id semper enim. Vestibulum metus felis, elementum sit amet
tristique non, pulvinar vitae metus.
```

Le code complet de la page :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>jQuery</title
```

© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

```
<meta charset="UTF-8">
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script>
$(document).ready(function(){
$("#bouton").click(function() {
$("#droit").load("lorem.htm");
});
});
</script>
<style>
button { margin-top: 10px;}
#gauche { width: 50px;
          float: left;}
#droit { width: 215px;
         height: 65px;
         border: 1px solid black;
         margin-top: 10px;
         padding: 3px;
         font-size: 0.8em;
         float: left;}
</style>
</head>
<body>
<div id="gauche">
<button id="bouton">Lire</button>
</div>
<div id="droit">
</div>
</body>
</html>
```

Comparons le code mis en œuvre par jQuery et celui utilisé par le JavaScript traditionnel pour initier une requête AJAX.

jQuery

```
$(document).ready(function({
$("#bouton").click(function() {
$("#droit").load("lorem.htm");
});
})
```

JavaScript traditionnel

```
function getxhr(){
var xhr = null;
if (window.XMLHttpRequest){
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status ==
200){
var contenu = xhr.responseText;
alert(contenu)
var cible =
document.getElementById("droit");
cible.innerHTML = contenu;
}
}
xhr.open("GET", "lorem.htm", true);
```


© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

```
xhr.send(null);
}
```

La concision du code jQuery par rapport au JavaScript traditionnel est flagrante !

2. En écriture complète : ajax()

Cette méthode permet d'effectuer une requête AJAX en maîtrisant, grâce aux nombreuses options disponibles, les différents paramètres et étapes de celle-ci.

ajax(options)

```
$.ajax({
type: "POST",
url: "test.htm",
dataType: "html",
success: function(data ) {
$("div").html(data);
}
});
```

Passons en revue les nombreuses options disponibles :

- `url` (obligatoire) : une chaîne de caractères contenant l'adresse de la requête.
- `type` (optionnel) : une chaîne de caractères qui définit la méthode HTTP à utiliser pour la requête (GET ou POST). La valeur par défaut est GET. D'autres méthodes d'envoi HTTP peuvent être utilisées, comme PUT ou DELETE, mais celles-ci ne sont pas supportées par tous les navigateurs.
- `dataType` (optionnel) : une chaîne de caractères qui spécifie le format des données qui seront renvoyées par le serveur (xml, html, json ou script). Si rien n'est spécifié, jQuery utilisera le type MIME pour déterminer le format adéquat, soit responseXML ou responseText. Les types disponibles sont :
  - "xml" : retourne un document XML qui pourra être traité par jQuery.
  - "html" : retourne du code HTML au format texte.
  - "script" : évalue la réponse en JavaScript et retourne cette dernière au format texte.
  - "json" : évalue la réponse en JSON et retourne un objet JavaScript.
- `ifModified` (optionnel) : une valeur booléenne qui indique que le serveur doit vérifier si les données retournées sont différentes de la dernière requête avant de renvoyer le fichier avec succès. Par défaut, cette option vaut `false`.
- `timeout` (optionnel) : nombre de millisecondes après lequel la requête est considérée comme non réussie.
- `global` (optionnel) : une valeur booléenne qui permet le déclenchement du gestionnaire d'événements global d'AJAX. Par défaut, la valeur est `true`. Avec une valeur `false`, les déclenchements d'événements de type `ajaxStart()` ou `ajaxStop()` sont ignorés.
- `beforeSend` (optionnel) : une fonction qui doit être exécutée avant l'envoi de la requête. Ceci permet de modifier l'objet XMLHttpRequest avant qu'il ne soit envoyé, pour spécifier, par exemple, des en-têtes HTTP personnalisés.
- `error` (optionnel) : une fonction qui doit être appelée en cas d'échec de la requête. La fonction dispose de trois arguments: l'objet XMLHttpRequest, une chaîne de caractères décrivant le type d'erreur rencontré et un objet d'exception, dans le cas où ce dernier a été généré.
- `success` (optionnel) : fonction à appeler si la requête s'exécute avec succès. Un seul argument est passé en paramètre, soit les données retournées par le serveur.

© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

- `complete` (optionnel) : fonction à exécuter lorsque la requête se termine. La fonction dispose de deux arguments : l'objet XMLHttpRequest et une chaîne de caractères décrivant le type de succès de la requête.
- `data` (optionnel) : données à envoyer au serveur. L'objet doit être formé de paires de la forme clé/valeur. Les données sont converties en chaîne de caractères (si elles ne le sont pas déjà). Voir l'option `processData` ci-après pour empêcher ce processus automatique.
- `processData` (optionnel) : valeur booléenne qui indique si les données de l'option `data` doivent être converties en chaîne de caractères. La valeur par défaut est `true`. Pour empêcher la conversion, passez cette option à `false`.
- `contentType` (optionnel) : chaîne de caractères contenant le MIME des données lorsque des données sont envoyées au serveur. Par défaut, le MIME `application/x-www-form-urlencoded` est retenu.
- `async` (optionnel) : une valeur booléenne qui indique si la requête doit s'effectuer de façon asynchrone ou synchrone. La valeur par défaut pour une requête AJAX est bien entendu `true`.

D'autres options sont encore disponibles mais d'un usage moins fréquent et qui dépassent largement le cadre de ce chapitre. Énumérons simplement :

- `cache` (optionnel) : une valeur booléenne qui, lorsqu'elle est à `false`, empêche la page chargée d'être mise dans le cache du navigateur.
- `password` (optionnel) : dans le cas où l'accès HTTP de la requête nécessite un mot de passe.
- `username` (optionnel) : dans le cas où l'accès HTTP de la requête nécessite un nom d'utilisateur (username).
- `scriptCharset` (optionnel) : force les requêtes du type script à être interprétées avec un charset particulier.
- `xhr` (optionnel) : permet de créer le ActiveXObject (Internet Explorer) ou le XMLHttpRequest (pour les autres).
- etc.

Exemple

Au clic sur un lien, faisons apparaître un contenu à partir du serveur par la méthode `ajax()`.

Le fichier de départ se présente comme suit :

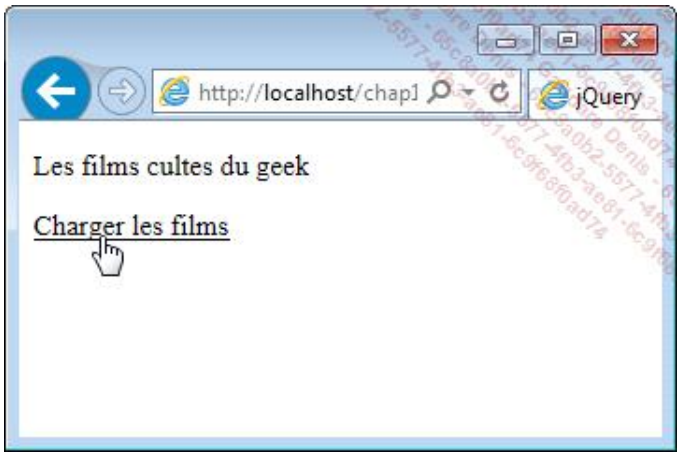
```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>jQuery</title>
<meta charset="UTF-8">
<style>
a { color: black;}
li { list-style-type: none;}
</style>
</head>
<body>
<p>Les films cultes du geek</p>
<a href="#">Charger les films</a>
<div id="contenu">
</div>
</body>
</html>
```

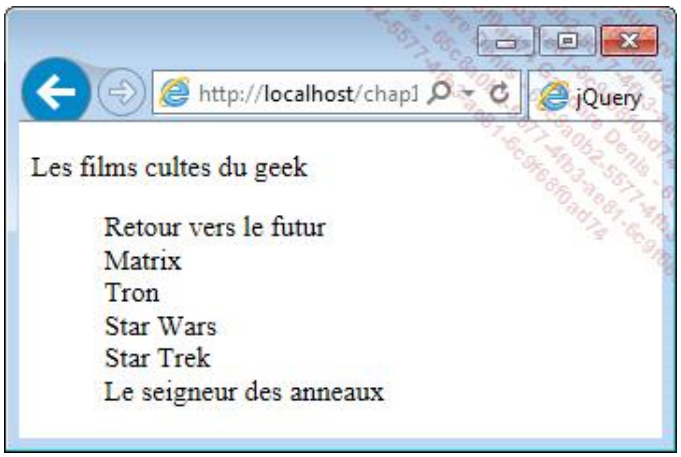
Le fichier à charger (films.htm) comporte la liste suivante :

© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

```
<ul>
<li>Retour vers le futur</li>
<li>Matrix</li>
<li>Tron</li>
<li>Star Wars</li>
<li>Star Trek</li>
<li>Le seigneur des anneaux</li>
</ul>
```

Le script jQuery doit lancer une requête AJAX vers le fichier films.htm et en afficher le contenu dans la division contenu de la page.






```
<script>
$(document).ready(function(){
$('a').click(function() {
$('a').hide();
$("#contenu").empty();
$.ajax({
url: "films.htm",
async: true,
type: "GET",
global: false,
cache: false,
```

© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

```
success: function(html){
$("#contenu").append(html)
}
});
});
});
</script>
```

Détaillons ce script.

```
$(document).ready(function(){
```

Chargement du DOM.

```
$('a').click(function() {
```

Au clic sur le lien <a>,

```
$('a').hide();
```

Le lien et son contenu sont cachés.

```
$("#contenu").empty();
```

Le contenu de la division contenu est vidé.

```
$.ajax({
url: "films.htm",
async: true,
type: "GET",
global: false,
cache: false,
success: function(html){
$("#contenu").append(html)
}
});
```

La requête AJAX de jQuery charge le fichier situé à l'URL films.htm. Le processus est effectué selon un mode asynchrone. La méthode HTTP retenue est GET. Le gestionnaire d'événements global d'AJAX est désactivé. Le fichier rapatrié du serveur n'est pas mis en cache. Enfin, si la requête s'est déroulée avec succès, les données du fichier chargé par celle-ci sont ajoutées à la division contenu.

```
});
});
```

Fin du script.

Le fichier final devient :

```
<!DOCTYPE html>
```


© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

```
<html lang="fr">
<head>
<title>jQuery</title>
<meta charset="UTF-8">
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script>
$(document).ready(function(){
$('a').click(function() {
$('a').hide();
$("#contenu").empty();
$.ajax({
url: "films.htm",
async: true,
type: "GET",
global: false,
cache: false,
success: function(html){
$("#contenu").append(html)
}
});
});
});
</script>
<style>
a { color: black;}
li { list-style-type: none;}
</style>
</head>
<body>
<p>Les films cultes du geek</p>
<a href="#">Charger les films</a>
<div id="contenu">
</div>
</body>
</html>
```

3. Les événements associés à la requête

Les événements introduits par jQuery permettront d'intervenir à toutes les étapes du processus de la requête.

`ajaxSend()`	Assigne une fonction qui sera exécutée avant l'envoi de la requête.
`ajaxStart()`	Assigne une fonction lorsque la requête AJAX débute.
`ajaxStop()`	Assigne une fonction lorsque toutes les requêtes AJAX sont terminées.
`ajaxSuccess()`	Assigne une fonction qui sera exécutée à chaque fois qu'une requête AJAX se termine avec succès.
`ajaxComplete()`	Assigne une fonction qui sera exécutée lorsque le processus global de la requête AJAX sera terminé.
`ajaxError()`	Assigne une fonction lorsque la requête AJAX échoue.

Depuis la version 1.8 de jQuery, les événements associés à une requête AJAX ne peuvent être reliés qu'à `document`.

© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

```
$(document).ajaxSend(function() {
$(this).show();
});
```

Exemple

Affichons un message avant que la requête ne débute.


```
<!DOCTYPE html>
<html>
<head>
<title>jQuery</title>
<script src="http://code.jquery.com/jquery-1.11.0.js"></script>
<style>
a { color: black;}
#message { width: 200px;
           border: 1px solid black;
           margin-top: 20px;
           background-color: rgb(195,215,235);
           padding-left: 5px;}
</style>
<script>
$(document).ready(function(){
$("#message").hide();
$(document).ajaxSend(function(){
$("#message").append("La requête AJAX va débuter !<br>").show();
});
$("a").click(function(){
$("#resultat").load("a.html");
});
});
</script>
</head>
<body>
<h3>ajaxSend()</h3>
<a href="#">Démarrer Ajax</a>
<div id="message"> </div>
</body>
</html>
```


© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

Passons en revue les différentes lignes du code jQuery :

```
$(document).ready(function(){
```

Chargement de jQuery.

```
$("#message").hide();
```

La division `message`, qui contiendra le message associé à l'événement `ajaxSend()`, est cachée au chargement de la page.

```
$("#resultat").hide();
```

La division `resultat` qui contiendra le contenu du fichier chargé est cachée, car ce contenu n'a pas d'importance dans cet exemple.

```
$(document).ajaxSend(function() {
$(this).append('La requête AJAX débute !<br>').show();
});
```

Lors de l'événement `ajaxSend()`, la phrase "La requête AJAX va débuter !" est ajoutée à la division `message` et celle-ci est alors rendue visible.

```
$("a").click(function() {
$("#resultat").load("a.html");
});
```

Au clic sur le lien, le fichier a.html est chargé et inséré dans la division `resultat`. Pour rappel, cette division a été cachée en début de script.

```
});
```

Fin de jQuery.

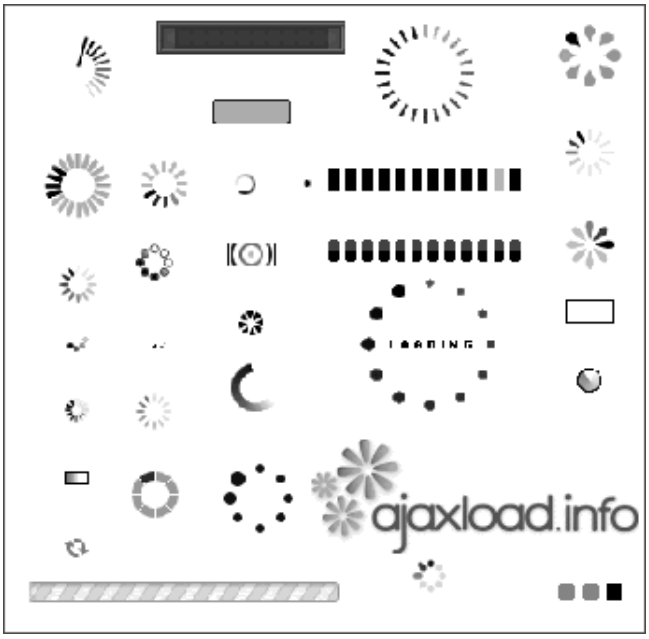
4. L'ajout d'une icône de chargement

Les requêtes AJAX peuvent parfois entraîner une relative lenteur au chargement du fichier externe en fonction de la taille du fichier, de l'encombrement du serveur et/ou de la qualité de la connexion.

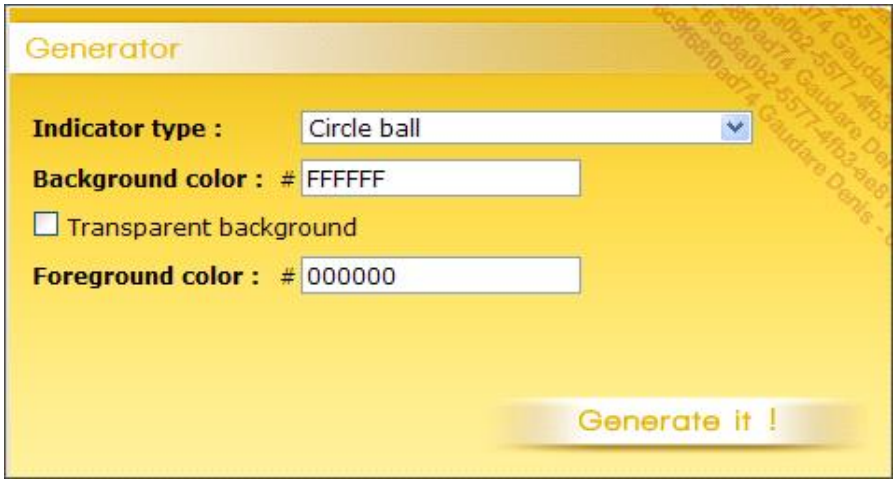
Ainsi, pour éviter que l'utilisateur ne soit désappointé par ces moments de latence, le concepteur peut prévoir une petite icône indiquant que le chargement est en cours. Cette icône est en fait une image gif animée qui apparaît à l'initialisation de la requête et qui disparaît lorsque la requête a abouti avec succès.

Le site Ajaxload (http://www.ajaxload.info/) propose une série impressionnante d'icônes animées de téléchargement.

© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare



Son interface permet de choisir le type d'animation (`Indicator type`), la couleur de fond (`Background color`) et la couleur de l'animation (`Foreground color`). Le bouton **Generate it !** crée l'image et en donne un aperçu. Le bouton **Download it !** vous permet de télécharger l'icône animée. Rien de plus simple. Parfait pour ne pas perdre de temps à créer soi-même ces images d'attente. Bien entendu, c'est entièrement gratuit.



Exemple

Soit le fichier de départ qui comporte simplement un bouton et une division destinée à recevoir le fichier une fois téléchargé.

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>jQuery</title>
<meta charset="UTF-8">
<style>
button { margin: 12px 0 12px 0}
#contenu { width: 250px;}
```

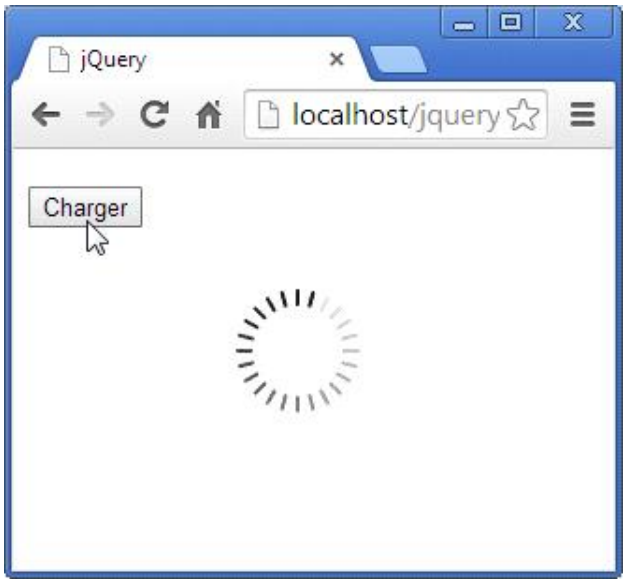

© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

```
</style>
</head>
<body>
<button id="bouton">Charger</button>
<div id="contenu"></div>
</body>
</html>
```

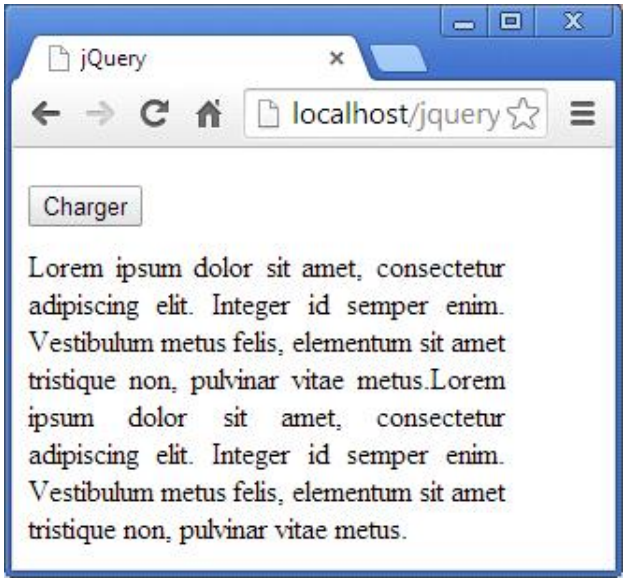
Le fichier à charger depuis le serveur s'appelle ici lorembis.htm. Lors de la phase de développement et de test en interne, l'apparition de l'icône de téléchargement risque d'être assez furtive. L'utilisation d'un fichier volumineux est conseillée afin de pouvoir apercevoir (ou plutôt entrapercevoir) celle-ci.

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer id semper
enim. Vestibulum metus felis, elementum sit amet tristique non, pulvinar
vitae metus.Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Integer id semper enim. Vestibulum metus felis, elementum sit amet
tristique non, pulvinar vitae metus.
...
```

Le script jQuery doit prendre en charge non seulement la requête AJAX, mais aussi l'apparition et la disparition de l'image de chargement.



© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare



L'image gif animée (ajax-loader.gif) est disponible en téléchargement depuis la page Informations générales.

```
<script>
$(document).ready(function() {
$("#bouton").click(function() {
$("#contenu").hide().load("lorembis.htm", function() {
$(this).fadeIn(4000);
});
return false;
});
$("<div id="loading"><img src="ajax-loader.gif"></div>")
.insertBefore("#contenu")
.ajaxStart(function() {
$(this).show();
})
.ajaxStop(function() {
$(this).hide();
});
});
</script>
```

Ce script nécessite quelques explications :

```
$(document).ready(function() {
```

Initialisation de jQuery.

```
$("#bouton").click(function() {
```

Au clic sur le bouton.

```
$("#contenu").hide().load("lorembis.htm", function() {
$(this).fadeIn(4000);
});
```


© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare

Une requête AJAX est envoyée vers le fichier lorembis.htm, et son contenu est inséré dans la division `contenu`. Un effet d'apparition progressive a été ajouté (`fadeIn(4000)`).

```
return false;
});
```

Fin de la partie de chargement.

```
$("<div id="loading"><img src="ajax-loader.gif"></div>")
.insertBefore('#contenu')
```

L'image ajax-loader.gif, incluse dans la division `loading`, est insérée avant la division `contenu` (`insertBefore`).

```
.ajaxStart(function() {
$(this).show();
})
```

Lorsque la requête AJAX démarre (`ajaxStart`), cette image (`this`) est rendue visible (`show()`).

```
.ajaxStop(function() {
$(this).hide();
});
```

Lorsque la requête AJAX se termine (`ajaxStop`), cette image (`this`) est alors cachée (`hide()`).

```
});
```

Fin du script jQuery.

Au final, voici la page définitive :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>jQuery</title>
<meta charset="UTF-8">
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script>
$(document).ready(function() {
$("#bouton").click(function() {
$("#contenu").hide().load("lorembis.htm", function() {
$(this).fadeIn(4000);
});
return false;
});
$("<div id="loading"><img src="ajax-loader.gif"></div>")
.insertBefore('#contenu')
.ajaxStart(function() {
$(this).show();
})
```

```
.ajaxStop(function() {
$(this).hide();
});
});
</script>
<style>
button {margin: 12px 0 12px 0}
#contenu { width: 250px;}
#loading { margin: 30px 0 0 30px;
           position: absolute;
           display: none;}
</style>
</head>
<body>
<button id="bouton">Charger</button>
<div id="contenu"></div>
</body>
</html>
```


© Editions ENI – Tous droits réservés – Copie personnelle de Denis Gaudare