

Documentation de la librairie prototype, version 1.4.0

Traduction pour <http://www.developpez.com> par [SpaceFrog \(Steven K. Martin\)](#), [Olivier Lance \(Accueil \)](#) et [Denis Cabasson \(Developpez.com \)](#) du document [Developer Notes for prototype.js](#) de Sergio Pereira.

Un grand merci à [Bisûnûrs](#) pour sa relecture complète de l'article.

par [Sergio Pereira](#)

Date de publication : 24/04/2007

Dernière mise à jour : 25/04/2007

Prototype est une librairie Javascript de bas niveau permettant de faire des manipulations DOM ou de l'AJAX facilement. Elle permet de faire abstraction complète des différents navigateurs et de leurs bugs ou spécificités en proposant des méthodes fonctionnant sur tous les navigateurs. Toutes les opérations les plus courantes sont outillées de façon appropriée pour pouvoir être utilisées facilement. Ce framework est de plus fortement orienté objet, ce qui en rend la lecture et l'utilisation d'autant plus aisées et naturelles pour tous les développeurs habitués à ce type de programmation.

De quoi s'agit-il ?

- 1 - Les fonctions utilitaires
 - 1-A - Utilisation de la fonction `$()`
 - 1-B - Utilisation de la fonction `$F()`
 - 1-C - Utilisation de la fonction `$A()`
 - 1-D - Utilisation de la fonction `$H()`
 - 1-E - Utilisation de la fonction `$R()`
 - 1-F - Utilisation de la fonction `Try.these()`
- 2 - L'objet Ajax
 - 2-A - Utilisation de la classe `Ajax.Request`
 - 2-B - Utilisation de la classe `Ajax.Updater`
- 3 - Enumérer... Youhou ! Super ! Génial !
 - 3-A - Boucles en syntaxe Ruby
 - 3-B - Vos tableaux dopés aux stéroïdes
- 4 - Des livres que je recommande fortement
- 5 - Référence de prototype.js
 - 5-A - Extensions des classes JavaScript existantes
 - 5-A-1 - Extensions pour la classe `Object`
 - 5-A-2 - Extensions pour la classe `Number`
 - 5-A-3 - Extensions pour la classe `Function`
 - 5-A-4 - Extensions pour la classe `String`
 - 5-A-5 - Extensions pour la classe `Array`
 - 5-A-6 - Extensions pour l'objet DOM `document`
 - 5-A-7 - Extensions pour l'objet `Event`
 - 5-B - Nouveaux objets et classes définis par prototype.js
 - 5-B-1 - L'objet `PeriodicalExecuter`
 - 5-B-2 - L'objet `Prototype`
 - 5-B-3 - L'objet `Enumerable`
 - 5-B-4 - L'objet `Hash`
 - 5-B-5 - La classe `ObjectRange`
 - 5-B-6 - L'objet `Class`
 - 5-B-7 - L'objet `Ajax`
 - 5-B-7-a - L'objet `Ajax.Responders`
 - 5-B-7-b - La classe `Ajax.Base`
 - 5-B-7-c - La classe `Ajax.Request`
 - 5-B-7-d - L'objet/argument `options`
 - 5-B-7-e - La classe `Ajax.Updater`
 - 5-B-7-f - La classe `Ajax.PeriodicalUpdater`
 - 5-B-8 - L'objet `Element`
 - 5-B-8-a - La classe `Element.ClassNames`
 - 5-B-9 - L'objet `Abstract`
 - 5-B-9-a - La classe `Abstract.Insertion`
 - 5-B-10 - L'objet `Insertion`
 - 5-B-10-a - La classe `Insertion.Before`
 - 5-B-10-b - La classe `Insertion.Top`
 - 5-B-10-c - La classe `Insertion.Bottom`
 - 5-B-10-d - La classe `Insertion.After`
 - 5-B-11 - L'objet `Field`
 - 5-B-12 - L'objet `Form`
 - 5-B-12-a - L'objet `Form.Element`
 - 5-B-12-b - L'objet `Form.Element.Serializers`
 - 5-B-13 - La classe `Abstract.TimedObserver`
 - 5-B-13-a - La classe `Form.Element.Observer`
 - 5-B-13-b - La classe `Form.Observer`

- 5-B-14 - La classe Abstract.EventObserver
 - 5-B-14-a - La classe Form.Element.EventObserver
 - 5-B-14-b - La classe Form.EventObserver

De quoi s'agit-il ?

Au cas où vous ne l'auriez pas encore utilisée, prototype.js est une librairie JavaScript écrite par Sam Stephenson. Ce code très bien pensé et bien écrit, conforme aux standards, facilite la création de pages riches et hautement interactives en Web 2.0.

Si vous avez essayé d'utiliser cette librairie récemment, vous avez pu vous rendre compte que sa documentation n'est pas son point fort. Comme de nombreux développeurs avant moi, j'ai apprivoisé prototype.js en lisant la source du code et en l'expérimentant. J'ai pensé qu'il serait sympa de prendre des notes au fur et à mesure de mon apprentissage et de les partager avec tout le monde.

Je propose également une référence non officielle pour les objets, classes et extensions fournis par cette librairie.

En lisant les exemples et les références, les développeurs familiers du langage Ruby pourront constater une similitude intentionnelle entre les classes Ruby et de nombreuses extensions mises en oeuvre dans cette librairie.

1 - Les fonctions utilitaires

La librairie comporte de nombreux objets prédéfinis et fonctions utilitaires. Le but évident de ces fonctions est de vous épargner des frappes répétées.

1-A - Utilisation de la fonction `$()`

La fonction `$()` est un raccourci pratique pour la fonction DOM très souvent utilisée `document.getElementById()`. Tout comme la fonction DOM, celle-ci retourne l'élément dont l'id est passé en paramètre.

Contrairement à la fonction DOM, celle-ci va plus loin. Vous pouvez lui passer plus d'un seul id et `$()` retournera un tableau (Array) des éléments requis. L'exemple ci-dessous illustre cela :

```
<html>
<head>
<title> Page de Test </title>
<script type='text/javascript' src="prototype-1.4.0.js"></script>

<script type='text/javascript'>
function test1()
{
    var d = $('monDiv');
    alert(d.innerHTML);
}

function test2()
{
    var divs = $('monDiv','monAutreDiv');
    for(i=0; i<divs.length; i++)
    {
        alert(divs[i].innerHTML);
    }
}
</script>
</head>

<body>
    <div id="monDiv">
        <p>Ceci est un paragraphe</p>
    </div>
    <div id="monAutreDiv">
        <p>Ceci est un autre paragraphe</p>
    </div>

    <input type="button" value="Test1" onclick="test1();"><br>
    <input type="button" value="Test2" onclick="test2();"><br>

</body>
</html>
```

Une autre chose appréciable avec cette fonction, c'est que l'on peut aussi lui passer indifféremment l'id de l'élément ou l'élément lui-même, ce qui la rend très utile lors de la création d'autres fonctions qui acceptent également l'une ou l'autre forme de paramètres.

1-B - Utilisation de la fonction `$F()`

La fonction `$F()` est un autre raccourci appréciable. Elle retourne la valeur de n'importe quel contrôle de saisie, tel que `input type='text'` ou `select`. La fonction accepte indifféremment en argument l'id de l'élément ou l'élément lui-même.

```
<script type='text/javascript'>
function test3()
```

```

    {
        alert( $F('utilisateur') );
    }
</script>

<input type="text" id="utilisateur" value="Marc Assin"><br>
<input type="button" value="Test3" onclick="test3();"><br>

```

1-C - Utilisation de la fonction \$A()

La fonction **\$A()** convertit l'unique paramètre qu'elle reçoit en un objet Array.

Cette fonction, combinée avec les extensions de la classe Array, facilite la conversion de n'importe quelle liste énumérable en tableau (Array) ou sa copie. Une suggestion d'utilisation est de convertir une NodeList DOM en un tableau (Array) qui pourra être parcouru plus efficacement. Voir l'exemple ci-dessous :

```

<script type='text/javascript'>
    function MontreOptions(){
        var LaNodeList = $('Employes').getElementsByTagName('option');
        var Noeuds = $A(LaNodeList);

        Noeuds.each(function(noeud){
            alert(noeud.nodeName + ': ' + noeud.innerHTML);
        });
    }
</script>

<select id="Employes" size="10" >
    <option value="5">Martin, Steven</option>
    <option value="8">Hauchon, Paul </option>
    <option value="1">Dupré, Jean</option>
</select>

<input type="button" value="Afficher les options" onclick="MontreOptions();" >

```

1-D - Utilisation de la fonction \$H()

La fonction **\$H()** convertit les objets en objets Hash énumérables qui s'apparentent à des tableaux associatifs.

```

<script type='text/javascript'>
    function testHash()
    {
        //creation de l'objet
        var a = {
            premier: 10,
            second : 20,
            troisieme : 30
        };

        // maintenant transformons le en hash
        var h = $H(a);
        alert(h.toString()); //affiche: premier=10&second=20&troisieme=30
    }
</script>

```

1-E - Utilisation de la fonction \$R()

La fonction **\$R()** est simplement un raccourci pour **new ObjectRange(limitebasse,limitehaute, exclurelimites)**.

Se référer à la documentation sur la classe ObjectRange pour une explication complète de cette classe. Toutefois, étudions un exemple simple qui montre également l'utilisation d'itérations au travers de la méthode each. Vous

trouvez plus d'informations sur cette méthode dans la documentation sur l'objet énumérable.

```
<script type='text/javascript'>
  function demoDollar_R(){
    var indice = $R(10, 20, false);
    indice.each( function(valeur, index) {
      alert(valeur);
    });
  }
</script>

<input type="button" value="Comptage Count" onclick="demoDollar_R();" >
```

1-F - Utilisation de la fonction Try.these()

La fonction **Try.these()** facilite l'appel à différentes fonctions ... heu ... jusqu'à ce que l'une d'entre elles fonctionne. Elle accepte plusieurs fonctions en paramètre et les appelle l'une après l'autre dans l'ordre jusqu'à ce que l'une d'entre elles fonctionne, elle retourne alors le résultat de la fonction dont l'appel à été fructueux.

Dans l'exemple ci-dessous, la fonction **xmlNode.text** est implémentée dans certains navigateurs et **xmlNode.textContent** est implémentée dans les autres navigateurs. En utilisant la fonction **Try.These()** il nous est possible d'obtenir le résultat de celle qui fonctionne.

```
<script>
function getValeurNoeud(noeudXML){
  return Try.these(
    function() {return noeudXML.text;},
    function() {return noeudXML.textContent;}
  );
}
</script>
```

2 - L'objet Ajax

Les fonctions utilitaires évoquées ci-dessus sont sympathiques, mais soyons honnêtes elles ne sont pas des plus avancées. Vous pourriez probablement en faire autant, et vous avez peut-être des fonctions similaires dans vos propres scripts. Mais ces fonctions ne sont que la partie émergée de l'iceberg.

Je suis sûr que votre intérêt pour prototype.js est principalement dirigé vers ses capacités en AJAX. Alors je vais vous expliquer en quoi cette librairie va vous faciliter la vie pour implémenter AJAX.

L'objet Ajax est un objet prédéfini créé par la librairie pour englober et simplifier le code compliqué nécessaire pour la mise en oeuvre de la technologie AJAX. Cet objet comporte de nombreuses classes qui intègrent les fonctionnalités de la technologie AJAX. Passons en revue certaines de ces classes.

2-A - Utilisation de la classe Ajax.Request

Si vous n'utilisez pas de librairie, vous devez certainement avoir recours à un code important pour créer un objet **XMLHttpRequest**, puis suivre son statut de façon asynchrone, puis en extraire la réponse et enfin traiter la réponse. Et estimez-vous heureux si vous n'avez pas à implémenter cela pour plusieurs navigateurs.

Pour faciliter les fonctionnalités d'AJAX, la bibliothèque définit la classe [Ajax.Request](#).

Admettons que vous ayez une application qui communique avec le serveur au travers de l'url *http://votreserveur/application/cherche_ventes?idEmploye=1234&annee=1998*, qui retourne une réponse XML comme suit :

```
<?xml version="1.0" encoding="utf-8" ?>
<reponse-ajax>
  <reponse type="object" id="detailsProduit">
    <ventes-mensuelles>
      <ventes-employe>
        <id-employe>1234</id-employe>
        <annee-mois>1998-01</annee-mois>
        <ventes>$8,115.36</ventes>
      </ventes-employe>
      <ventes-employe>
        <id-employe>1234</id-employe>
        <annee-mois>1998-02</annee-mois>
        <ventes>$11,147.51</ventes>
      </ventes-employe>
    </ventes-mensuelles>
  </reponse>
</reponse-ajax>
```

Communiquer avec le serveur pour récupérer ce XML est relativement simple en utilisant l'objet [Ajax.Request](#). L'exemple ci-dessous montre comment :

```
<script>
function rechercheVentes() {

    var idEmploye = $F('listeEmployes');
    var annee = $F('listeAnnees');
    var url = 'http://votreserveur/application/cherche_ventes';
    var parametres = 'idEmploye=' + idEmploye + '&annee=' + annee;

    var myAjax = new Ajax.Request(
        url,
        {
            method: 'get',
            parameters: parametres,
            onComplete: afficheReponse
        }
    );
}
```

```

    );
}
function afficheReponse(requete) {
    //affiche le XML dans le textarea
    $('resultat').value = requete.responseText;
}
</script>
<select id="listeEmployes" size="10" onchange="rechercheVentes()">
    <option value="5">Buchanan, Steven</option>
    <option value="8">Callahan, Laura</option>
    <option value="1">Davolio, Nancy</option>
</select>
<select id="listeAnnees" size="3" onchange="rechercheVentes()">
    <option selected="selected" value="1996">1996</option>
    <option value="1997">1997</option>
    <option value="1998">1998</option>
</select>
<br>
<textarea id='resultat' cols=60 rows=10 ></textarea>

```

Avez-vous vu le second paramètre passé au constructeur de l'objet [Ajax.Request](#) ? Le paramètre **{method: 'get', parameters: pars, onComplete: afficheReponse}** représente un objet anonyme en notation littérale (aussi connu sous le nom de JSON). Ce que je veux dire c'est que nous passons un objet qui possède une propriété nommée **method** qui contient le string 'get', une autre propriété nommée **parameters** qui contient la chaîne d'interrogation de la requête HTTP et une propriété/méthode **onComplete** qui contient la fonction **afficheReponse**.

Il y a d'autres propriétés que l'on peut renseigner dans cet objet, telle que **asynchronous** qui peut prendre les valeurs **true** ou **false** et détermine si l'appel AJAX vers le serveur doit être fait de façon asynchrone ou non (la valeur par default est **true**).

Ce paramètre définit les options pour l'appel AJAX. Dans notre exemple, nous appelons l'URL dans le premier paramètre par une commande HTTP GET, en passant la chaîne d'interrogation contenue dans la variable **parameters**, et l'objet [Ajax.Request](#) appellera la fonction **afficheReponse** lorsqu'elle aura terminé la récupération de la réponse.

Comme vous le savez, **XMLHttpRequest** rend compte de sa progression au cours de l'appel HTTP. Cet avancement se décompose en quatre étapes différentes : *Loading* (En Chargement), *Loaded* (Chargé), *Interactive* (Interactif) ou *Complete* (Achévé). On peut demander à l'objet **Ajax.Request** d'appeler une fonction personnelle à n'importe laquelle de ces étapes, l'étape Complete étant la plus utilisée. Pour informer l'objet de la fonction, ajouter simplement la propriété/méthode appelée **onXXXX** dans les options de requête, tout comme le **onComplete** dans notre exemple. La fonction passée sera appelée par l'objet avec deux paramètres, le premier sera le **XMLHttpRequest** (aussi connu sous l'appellation de XHR) lui-même, le second sera l'évaluation X-JSON du header HTTP de la réponse (s'il y en a un).

```

<script>
    var mesGestionnairesGlobaux = {
        onCreate: function(){
            Element.show('systemeAttente');
        },
        onComplete: function() {
            if(Ajax.activeRequestCount == 0){
                Element.hide('systemeAttente');
            }
        }
    };

    Ajax.Responders.register(mesGestionnairesGlobaux);
</script>

<div id='systemeAttente'>
    <img src='sablier.gif'>Chargement en cours...
</div>

```

2-B - Utilisation de la classe Ajax.Updater

Si vous avez un script serveur qui peut renvoyer des informations préformatées en HTML, alors la bibliothèque peut vous rendre la vie encore plus facile avec la classe [Ajax.Updater](#). Avec celle-ci, vous n'avez qu'à indiquer quel élément doit être alimenté par le code HTML renvoyé par l'appel AJAX. Un exemple vaut mieux qu'un long discours :

```
<script>
function chercheHTML() {

    var url = 'http://votreserveur/application/chercheHTML';
    var parametres = 'unParametre=ABCD';

    var myAjax = new Ajax.Updater(
        'emplacement',
        url,
        {
            method: 'get',
            parameters: parametres
        }
    );

}
</script>

<input type='button' value='ChercheHTML' onclick="chercheHTML()" >
<div id="emplacement"></div>
```

Comme vous pouvez le voir, le code ressemble beaucoup à l'exemple précédent, à l'exception de la fonction **onComplete** et de l'id de l'élément passé au constructeur. Changeons un peu ce code pour montrer comment il est possible de gérer les erreurs serveur côté client.

Nous allons ajouter quelques options supplémentaires à l'appel, en spécifiant une fonction qui sera appelée en cas d'erreur. Cela se fait grâce à l'option **onFailure**. Nous préciserons également que notre élément **emplacement** ne doit être alimenté qu'en cas de succès de l'appel. Pour parvenir à nos fins, nous allons transformer le premier paramètre d'un simple id d'élément à un objet possédant deux propriétés, **success** (à utiliser quand tout s'est bien passé) et **failure** (à utiliser quand quelque chose se passe mal). Nous n'utiliserons pas la propriété **failure** dans notre exemple, mais uniquement la fonction **rapporteErreur** dans l'option **onFailure**.

```
<script>
function chercheHTML() {

    var url = 'http://votreserveur/application/chercheHTML';
    var parametres = 'unParametre=ABCD';

    var myAjax = new Ajax.Updater(
        'emplacement',
        url,
        {
            method: 'get',
            parameters: parametres,
            onFailure: rapporteErreur
        }
    );

    function rapporteErreur(requete) {
        alert('Désolé, une erreur s'est produite.');
```

Si les balises HTML renvoyées par votre code côté serveur sont accompagnées de code JavaScript, l'objet [Ajax.Updater](#) peut évaluer ce dernier. Afin que l'objet interprète la réponse comme du code JavaScript, ajoutez

simplement **evalScripts: true**; à la liste des propriétés du dernier paramètre du constructeur de l'objet. Un avertissement toutefois : ces blocs de script ne seront pas ajoutés aux scripts de la page. Comme l'option **evalScripts** le suggère les scripts seront **évalués**. Quelle est la différence vous demandez-vous ? Imaginons que la requête exécutée renvoie quelque chose comme :

```
<script language="javascript" type="text/javascript">
    function disBonjour(){
        alert('Bonjour!');
    }
</script>
<input type=button value="Clique Moi" onclick="disBonjour()">
```

Si vous avez essayé cela auparavant, vous savez que cela ne fonctionne pas. La raison en est que le bloc de script sera évalué, et évaluer un script tel que le précédent ne créera pas de fonction nommée **disBonjour**. Cela ne fera rien. Pour créer cette fonction nous devons changer notre script pour **créer** la fonction. Voyez ci-dessous :

```
<script language="javascript" type="text/javascript">
    disBonjour = function(){
        alert('Bonjour!');
    };
</script>
<input type=button value="Clique Moi" onclick="disBonjour()">
```

Notez que dans l'exemple précédent nous n'utilisons pas le mot-clef **var**. Si nous l'avions utilisé, cela aurait créé un objet fonction qui aurait été local au bloc de script (dans IE tout du moins). Sans le mot-clef **var**, l'objet fonction est global à la fenêtre, ce que nous souhaitons.

3 - Enumérer... Youhou ! Super ! Génial !

Nous sommes tous habitués aux boucles *for*. Vous savez, créer un tableau à la main, le remplir avec des éléments du même type, créer une boucle de contrôle (*for*, *foreach*, *while*, *repeat*, etc), accéder à chaque élément de manière séquentielle, par son index numérique, et faire quelque chose avec cet élément.

Si vous y réfléchissez bien, chaque fois que vous utilisez un tableau dans votre code, cela signifie que, tôt ou tard, vous utiliserez ce tableau dans une boucle.

Ne serait-ce pas agréable si ces tableaux offraient plus de fonctionnalités quant à ces itérations ? Oui, ce le serait, et bon nombre de langages de programmation fournissent de telles fonctionnalités pour leurs tableaux ou leurs structures équivalentes (comme les collections et les listes).

Eh bien justement *prototype.js* nous propose l'objet [Enumerable](#), qui implémente pour nous une pléthore de trucs et astuces à utiliser lors des itérations sur les données.

La bibliothèque *prototype.js* va un peu plus loin, et étend la classe *Array* avec toutes les méthodes de [Enumerable](#).

3-A - Boucles en syntaxe Ruby

Classiquement en JavaScript, si vous voulez afficher les éléments d'un tableau de manière séquentielle, vous pouviez tout à fait écrire quelque chose comme ceci :

```
<script>
function montreListe() {
    var simpsons = ['Homer', 'Marge', 'Lisa', 'Bart', 'Meg'];

    for(i=0;i<simpsons.length;i++){
        alert(simpsons[i]);
    }
}
</script>


```

Avec notre nouveau meilleur ami, *prototype.js*, nous pouvons réécrire la boucle ainsi :

```
function montreListe(){
    var simpsons = ['Homer', 'Marge', 'Lisa', 'Bart', 'Meg'];

    simpsons.each(
        function(membreDeLaFamille){
            alert(membreDeLaFamille);
        }
    );
}
```

Vous pensez sans doute "mon dieu... quelle syntaxe étrange pour faire la même chose". Bon, dans l'exemple d'au-dessus, oui, il n'y a rien de bien extraordinaire. Après tout, il n'y a pas grand chose à modifier dans un exemple aussi trivial. Mais continuez de lire néanmoins.

Afin de continuer : vous voyez cette fonction qui est passée en argument à la méthode **each** ? Commençons dès maintenant à la désigner comme fonction **itératrice**.

3-B - Vos tableaux dopés aux stéroïdes

Comme nous l'avons mentionné auparavant, il est très courant que tous les éléments de vos tableaux soient du même type, avec les mêmes propriétés et méthodes. Voyons comment nous pouvons tirer partie des fonctions itératrices avec nos tout nouveaux tableaux.

Trouver un élément suivant un critère :

```
<script>
function trouveEmployeParId(idEmploye){
    var listBox = $('listeEmployes')
    var options = listBox.getElementsByTagName('option');
    options = $A(options);
    var opt = options.find(
        function(employe){
            return (employe.value == idEmploye);
        }
    );
    alert(opt.innerHTML); //affiche le nom de l'employe
}
</script>

<select id="listeEmployes" size="10" >
    <option value="5">Buchanan, Steven</option>
    <option value="8">Callahan, Laura</option>
    <option value="1">Davolio, Nancy</option>
</select>

<input type="button" value="Trouve Laura" onclick="trouveEmployeParId(8);" >
```

Maintenant, améliorons-le encore un peu. Voyons comment filtrer des éléments dans des tableaux, pour ne retrouver qu'un membre de chaque élément.

```
<script>
function montreLiensLocaux(paragraphe){
    paragraphe = $(paragraphe);
    var liens = $A(paragraphe.getElementsByTagName('a'));
    //trouve les liens qui ne commencent pas par 'http'
    var liensLocaux = liens.findAll( function(lien){
        var debut = lien.href.substring(0,4);
        return debut != 'http';
    });
    //maintenant, les textes des liens
    var textes = liensLocaux.pluck('innerHTML');
    //rassemblons les dans une seule chaîne
    var resultat = textes.inspect();
    alert(resultat);
}
</script>

<p id="duTexte">
Ce <a href="http://unAutreSite.com/page.html">texte</a> a
<a href="#ancreLocal">beaucoup</a> de
<a href="#autreAncre">liens</a>. Certain sont
<a href="http://OuVousVoulez.com/page.html">externes</a>
et d'autre sont <a href="#uneAncre">locaux</a>
</p>
<input type="button" value="Trouve les liens locaux" onclick="montreLiensLocaux('duTexte')">
```

Cela ne demande qu'un peu de pratique pour devenir complètement dépendant de cette syntaxe. Voyez les références des objets [Enumerable](#) et [Array](#) pour toutes les fonctions disponibles.

4 - Des livres que je recommande fortement

Certains livres sont tout simplement trop bons pour ne pas passer le mot. Les livres suivants m'ont beaucoup aidé à acquérir les nouvelles compétences nécessaires pour créer des applications AJAX, et aussi consolider les compétences que je pensais déjà maîtriser. Je pense qu'un bon livre constitue un bon investissement, qui est vite rentabilisé.

5 - Référence de prototype.js

5-A - Extensions des classes JavaScript existantes

Une des façons pour la bibliothèque prototype.js d'ajouter des fonctionnalités est d'étendre les classes JavaScript existantes.

5-A-1 - Extensions pour la classe Object

méthode	type	arguments	description
extend(destination,source)	statique	destination : un objet, source: un objet	Propose une solution pour implémenter l'héritage en copiant toutes les propriétés et méthodes de source vers destination .
inspect(objetCible)	statique	objetCible : un objet	Retourne une chaîne lisible représentant objetCible. Par défaut, retourne la valeur de toString si l'objet donné n'a pas de définition d'une méthode d'instance inspect .

5-A-2 - Extensions pour la classe Number

méthode	type	arguments	description
toColorPart()	instance	(aucun)	Retourne la représentation hexadécimale de ce nombre. Particulièrement utile pour convertir les composants RVB dans leur représentation HTML.
succ()	instance	(aucun)	Retourne le nombre suivant. Cette fonction est utilisée dans le cas d'une itération.
times(iterateur)	instance	iterateur: un objet fonction conforme à Function(index)	Appelle la méthode iterateur de façon répétée en lui passant l'index courant comme argument index .

démonstration de Number.times()

```
<script>
function demoTimes(){
  var n = 10;
  n.times(function(index){
    alert(index);
  });
  /*****
   * on aurait également pu utiliser:
   * (10).times( ... );
   *****/
}

```

démonstration de Number.times()

```

</script>
<input type=button value="Test Number.times()" onclick="demoTimes()">

```

5-A-3 - Extensions pour la classe Function

méthode	type	arguments	description
bind(objet)	instance	objet: l'objet qui possède la méthode	Retourne une instance de la fonction pré-liée à l'objet propriétaire de la fonction (=méthode). La fonction retournée aura les mêmes arguments que l'originale.
bindAsEventListener(objet)	instance	objet: l'objet qui possède la méthode	Retourne une instance de la fonction pré-liée à l'objet propriétaire de la fonction (=méthode). La fonction retournée aura l'objet event courant comme argument.

Voyons une des ces extensions en action:

démonstration de Function.bindAsEventListener()

```

<input type='checkbox' id='maChk' value=1> Test?
<script>
  //déclaration de la classe
  var SurveillantCheckbox = Class.create();

  //définition de l'implémentation de notre classe
  SurveillantCheckbox.prototype = {

    initialise: function(chkBox, message) {
      this.chkBox = $(chkBox);
      this.message = message;

      //assignement de notre méthode à l'événement
      this.chkBox.onclick =
        this.montrerMessage.bindAsEventListener(this);
    },

    montrerMessage: function(evt) {
      alert(this.message + ' (' + evt.type + ')');
    }
  };

  var surveillant = new SurveillantCheckbox('maChk', 'Changement');
</script>

```

5-A-4 - Extensions pour la classe String

méthode	type	arguments	description
stripTags()	instance	(aucun)	Retourne la chaîne de caractères avec les balises HTML ou XML supprimées.
stripScripts()	instance	(aucun)	Retourne la chaîne de caractères avec les blocs <script /> supprimés.

méthode	type	arguments	description
<code>escapeHTML()</code>	instance	(aucun)	Retourne la chaîne de caractères avec les caractères spéciaux HTML propres.
<code>unescapeHTML()</code>	instance	(aucun)	L'inverse de <code>escapeHTML()</code> .
<code>extractScripts()</code>	instance	(aucun)	Retourne un objet Array contenant tous les blocs <code><script /></code> trouvés dans la chaîne de caractères.
<code>evalScripts()</code>	instance	(aucun)	Evalue chacun des blocs <code><script /></code> trouvés dans la chaîne de caractères.
<code>toQueryParams()</code>	instance	(aucun)	Sépare une chaîne d'interrogation en Array associatif, indexé par le nom du paramètre (semblable à une table de hachage).
<code>parseQuery()</code>	instance	(aucun)	Même chose que <code>toQueryParams()</code> .
<code>toArray()</code>	instance	(aucun)	Sépare la chaîne de caractères en un Array de ses caractères.
<code>camelize()</code>	instance	(aucun)	Convertit une chaîne délimitée par des tirets en une chaîne en camelCase. Cette méthode est utile, par exemple, dans le cas des propriétés de style.

5-A-5 - Extensions pour la classe Array

Pour commencer, [Array](#) hérite de [Enumerable](#), si bien que toutes les méthodes pratiques, définies dans les objets [Enumerable](#), sont disponibles. De plus, les méthodes ci-dessous sont également implémentées.

méthode	type	arguments	description
<code>clear()</code>	instance	(aucun)	Vide ce tableau et le retourne.
<code>compact()</code>	instance	(aucun)	Retourne ce tableau sans les éléments qui sont null ou undefined . Cette méthode ne modifie pas le tableau lui-même.
<code>first()</code>	instance	(aucun)	Retourne le premier élément du tableau.
<code>flatten()</code>	instance	(aucun)	Retourne une version plate, à une dimension du tableau. Cet aplatissage se fait en cherchant tous les éléments qui sont eux-mêmes des tableaux

méthode	type	arguments	description
			et en ajoutant leurs éléments dans le tableau final, récursivement.
indexOf(valeur)	instance	valeur : ce que vous recherchez	Retourne la position (commençant à 0) de la valeur donnée si elle est présente dans le tableau. Retourne -1 si la valeur n'est pas trouvée.
inspect()	instance	(aucun)	Surchargée pour retourner une chaîne formatée représentant le tableau et ses éléments.
last()	instance	(aucun)	Retourne le dernier élément du tableau.
reverse([aSoiMeme])	instance	aSoiMeme : indique si le tableau lui-même doit être inversé	Retourne le tableau dans l'ordre inverse. Si aucun argument n'est fourni, ou que l'argument est true , le tableau lui-même est inversé. Sinon, il reste inchangé.
shift()	instance	(aucun)	Retourne le premier élément et le supprime du tableau, réduisant la taille du tableau de 1.
without(valeur1 [, valeur2 [, .. valeurN]])	instance	valeur1 ... valeurN : les valeurs à exclure du tableau si présentes.	Retourne le tableau, à l'exception des éléments qui sont dans la liste des arguments.

5-A-6 - Extensions pour l'objet DOM document

méthode	type	arguments	description
getElementsByClassName(nomClasse [, elementParent])	nomClasse	nomClasse : nom de la classe CSS associée aux éléments, elementParent : élément ou id de l'élément contenant les éléments recherchés	Retourne tous les éléments qui sont associés au nom de classe CSS donné. Si aucun elementParent n'est donné, tout le corps du document sera inspecté.

5-A-7 - Extensions pour l'objet Event

propriété	type	description
KEY_BACKSPACE	Number	8 : Constante. Code pour la touche Retour Chariot.
KEY_TAB	Number	9 : Constante. Code pour la touche Tabulation.
KEY_RETURN	Number	13 : Constante. Code pour la touche Entrée.
KEY_ESC	Number	27 : Constante. Code pour la touche

propriété	type	description
		Echap.
KEY_LEFT	Number	37 : Constante. Code pour la touche Flèche gauche.
KEY_UP	Number	38 : Constante. Code pour la touche Flèche haute.
KEY_RIGHT	Number	39 : Constante. Code pour la touche Flèche droite.
KEY_DOWN	Number	40 : Constante. Code pour la touche Flèche vers le bas.
KEY_DELETE	Number	46 : Constante. Code pour la touche Suppr.
observers	Array	Liste des observateurs mis en cache. Fait partie des détails de l'implémentation interne de l'objet.

méthode	type	arguments	description
isLeftClick(event)	static	event : un objet Event	Retourne true si le bouton gauche a été cliqué.
pointerX(event)	static	event : un objet Event	Retourne la coordonnée X (abscisse) du pointeur de la souris sur cette page.
pointerY(event)	static	event : un objet Event	Retourne la coordonnée Y (ordonnée) du pointeur de la souris sur cette page.
stop(event)	static	event : un objet Event	Utilisez cette méthode pour arrêter le comportement par défaut d'un événement et stopper sa propagation.
findElement(event, nomBalise)	static	event : un objet Event, nomBalise : nom de la balise désirée	Parcourt l'arbre DOM vers le haut pour trouver le premier élément ayant la balise donnée, en commençant par l'élément à l'origine de l'évènement.
observe(element, nom, observateur, utiliseCapture)	static	element : objet ou id, nom : nom de l'évènement (comme 'click', 'load', ...), observateur : fonction pour traiter l'évènement, utiliseCapture : si true , gère l'évènement dans la phase de capture et si false dans la phase de 'bubbling'	Ajoute un gestionnaire d'évènement à un élément.
stopObserving(element, nom, observateur, utiliseCapture)	static	element : objet ou id, nom : nom de l'évènement (comme 'click', 'load', ...), observateur : fonction pour traiter l'évènement, utiliseCapture : si true , gère l'évènement dans la phase de capture et si	Supprime un gestionnaire d'évènement à un élément.

méthode	type	arguments	description
		false dans la phase de 'bubbling'	
_observeAndCache(element, nom, observateur, utiliseCapture)	static		Méthode privée. Ne vous en souciez pas.
unloadCache()	static	(aucun)	Méthode privée. Ne vous en souciez pas. Supprime tous les observateurs en cache de la mémoire.

Voyons comment utiliser cet objet pour ajouter un gestionnaire d'évènement sur l'évènement 'load' de l'objet **window**.

```
<script>
  Event.observe(window, 'load', montreMessage, false);

  function montreMessage() {
    alert('Page chargée.');
```

5-B - Nouveaux objets et classes définis par prototype.js

Cette bibliothèque vous aide également d'une autre façon en vous proposant de nombreux objets qui permettent une conception orientée objet ou qui implémentent des fonctionnalités habituellement utilisées.

5-B-1 - L'objet PeriodicalExecuter

Cet objet propose la logique nécessaire pour appeler une fonction donnée de façon répétée, à intervalle donné.

méthode	type	arguments	description
[constructeur](rappel, intervalle)	constructeur	rappel : une méthode sans arguments, intervalle : nombre de secondes	Crée une instance de cet objet qui appellera la méthode de façon répétée.

propriété	type	description
callback	Function()	La méthode à appeler. Aucun paramètre ne lui sera passé.
frequency	Number	C'est en fait l'intervalle de temps en secondes.
currentlyExecuting	Boolean	Indique si un appel de la méthode est en cours.

5-B-2 - L'objet Prototype

L'objet **Prototype** n'a aucun rôle important, autre que déclarer la version actuelle de la librairie.

propriété	type	description
Version	String	La version de la bibliothèque.

propriété	type	description
emptyFunction	Function()	Un objet Function vide.
K	Function(obj)	Un objet Function qui retourne le paramètre.
ScriptFragment	String	Une expression régulière permettant d'identifier les scripts.

5-B-3 - L'objet Enumerable

L'objet `Enumerable` permet d'écrire du code plus lisible pour itérer sur les éléments d'une structure de type liste.

Beaucoup d'autres objets étendent l'objet `Enumerable` pour profiter de son interface très utile.

méthode	type	arguments	description
<code>each(iterateur)</code>	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Appelle la méthode <code>iterateur</code> en passant chacun des éléments de la liste comme premier argument et l'indice dans la liste comme second argument.
<code>all([iterateur])</code>	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code> , optionnel	Cette méthode permet de tester toute une collection de valeurs en utilisant une fonction donnée. all retournera false si la fonction <code>iterateur</code> retourne false ou null pour un des éléments. Elle retournera true autrement. Si aucun <code>iterateur</code> n'est donné, alors le test sera si l'élément est lui-même différent de false ou null . Vous pouvez simplement le penser comme : "vérifie que tous les éléments sont non-faux".
<code>any([iterateur])</code>	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code> , optionnel	Cette méthode permet de tester toute une collection de valeurs en utilisant une fonction donnée. any retournera true si la fonction <code>iterateur</code> ne retourne pas false ou null pour un des éléments. Elle retournera false autrement. Si aucun <code>iterateur</code> n'est donné, alors le test sera si l'élément est lui-même différent de false ou null . Vous pouvez simplement le penser comme : "vérifie

méthode	type	arguments	description
			que tous les éléments sont non-faux".
collect(iterateur)	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Appelle la méthode <code>iterateur</code> pour chacun des éléments de la collection et retourne tous les résultats dans un <code>Array</code> , un élément résultat pour chaque élément dans la collection, dans le même ordre.
detect(iterateur)	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Appelle la méthode <code>iterateur</code> pour chacun des éléments de la collection et retourne le premier élément qui a provoqué le retour de <code>true</code> par l'iterateur (ou plus précisément, non-faux). Si aucun des éléments ne retourne vrai, detect retourne null.
entries()	instance	(aucun)	Idem que toArray() .
find(iterateur)	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Idem que detect() .
findAll(iterateur)	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Appelle la méthode <code>iterateur</code> pour chacun des éléments de la collection et retourne un <code>Array</code> de tous les éléments pour lesquels l'iterateur a retourné une valeur équivalente à true . Cette méthode est le complémentaire de reject() .
grep(motif [, itérateur])	instance	motif : une expression régulière utilisée pour tester les éléments, itérateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Teste la valeur en string de chacun des éléments de la collection contre le motif de l'expression régulière. Cette méthode retourne un <code>Array</code> contenant tous les éléments qui correspondent à l'expression régulière. Si une méthode itérateur est donnée, l' <code>Array</code> contiendra le résultat de l'appel à l'itérateur avec chacun des éléments qui étaient conformes à l'expression régulière.
include(obj)	instance	obj : un objet	Cherche l'objet dans la collection courante. Retourne true si l'objet est

méthode	type	arguments	description
			trouvé, false autrement.
inject(valeurInitiale, itérateur)	instance	valeurInitiale : n'importe quel objet, faisant office de valeur initiale, itérateur : un objet fonction se conformant à la signature <code>Function(accumulateur, valeur, indice)</code>	Combine tous les éléments de la collection en utilisant la méthode itérateur. L'itérateur est appelé en passant le résultat de l'appel précédent via l'argument accumulateur. Le premier appel utilise valeurInitiale comme argument accumulateur . Le dernier résultat est le retour de la méthode.
invoke(nomMethode [, arg1 [, arg2 [...]]])	instance	nomMethode: nom de la méthode qui va être appelée dans chaque élément, arg1..argN: arguments qui seront passés en paramètres de l'appel.	Appelle la méthode spécifiée par nomMethode dans chaque élément de la collection, en passant les arguments donnés (arg1 à argN), et retourne les résultats dans un objet Array .
map(itérateur)	instance	itérateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Idem que collect() .
max([itérateur])	instance	itérateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Retourne l'élément avec la plus grande valeur dans la collection ou avec le plus grand résultat de l'appel à l'itérateur pour chacun des éléments de la collection, si un itérateur est donné.
member(obj)	instance	obj : un objet	Idem que include() .
min([itérateur])	instance	itérateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Retourne l'élément avec la plus petite valeur dans la collection ou avec le plus petit résultat de l'appel à l'itérateur pour chacun des éléments de la collection, si un itérateur est donné.
partition([itérateur])	instance	itérateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Retourne un Array contenant deux autres vecteurs. Le premier vecteur contiendra tous les éléments pour lesquels l'itérateur a retourné true et le second vecteur contiendra les éléments restants. Si aucun itérateur n'est donné, le premier vecteur contiendra tous les éléments équivalents à true et l'autre vecteur, les éléments restant.

méthode	type	arguments	description
pluck(nomPropriété)	instance	nomPropriété : nom de la propriété qui sera lue dans chacun des éléments. Peut également être l'indice de la propriété.	Recherche la valeur de la propriété passée en argument dans chacun des éléments de la collection et retourne le résultat dans un objet Array .
reject(iterateur)	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Appelle la méthode <code>iterateur</code> sur chacun des éléments de la collection et retourne un Array avec tous les éléments pour lesquels la fonction <code>iterateur</code> a retourné une valeur équivalente à false . Cette méthode est l'opposé de findAll() .
select(iterateur)	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Idem que findAll() .
sortBy(iterateur)	instance	iterateur : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Retourne un Array contenant les éléments triés selon le résultat de l'appel à l'iterateur.
toArray()	instance	(aucun)	Retourne un Array avec tous les éléments de la collection.
zip(collection1[, collection2[, ... collectionN]] [,transformation])	instance	collection1 .. collectionN: énumérations qui seront fusionnées, transformation : un objet fonction se conformant à la signature <code>Function(valeur, indice)</code>	Fusionne chacune des collections données avec la collection courante. La fusion retourne un nouveau vecteur, avec le même nombre d'éléments que la collection courante et chacun des éléments de ce vecteur est un vecteur (appelons les sous-vecteur) constitué des éléments avec le même indice dans chacune des collections fusionnées. Si une fonction transformation est donnée, chaque sous-vecteur sera transformé par cette fonction avec d'être fusionné. Exemple rapide : <code>[1,2,3].zip([4,5,6],[7,8,9]).inspect()</code> retourne <code>"[[1,4,7],[2,5,8],[3,6,9]]"</code> .

5-B-4 - L'objet Hash

L'objet [Hash](#) implémente une structure de type table de hachage, c'est-à-dire une collection de paires clés/valeur.

Chacun des éléments d'un objet [Hash](#) est un vecteur avec deux éléments : d'abord la clé, puis la valeur. Chaque élément a également deux propriétés, **key** et **value** qui sont assez explicites.

méthode	type	arguments	description
keys()	instance	(aucun)	Retourne un Hash avec les clés de tous les éléments.
values()	instance	(aucun)	Retourne un Hash avec les valeurs de tous les éléments.
merge(autreHash)	instance	autreHash : un objet Hash	Combine l'objet hash avec un autre objet hash passé en argument et retourne l'objet hash résultant.
toQueryString()	instance	(aucun)	Retourne tous les éléments de la table de hachage dans une chaîne formatée comme une chaîne de paramètres, i.e. 'cle1=valeur1&cle2=valeur2&cle3=v
inspect()	instance	(aucun)	Surchargée pour retourner une chaîne formatée représentant l'objet hash avec ses paires clé/valeur.

5-B-5 - La classe ObjectRange

Hérite de [Enumerable](#)

Représente une plage de valeurs avec les limites inférieures et supérieures.

propriété	type	description
start	(tous)	La borne inférieure de la plage de valeurs.
end	(tous)	La borne supérieure de la plage de valeurs.
exclusive	Booléen	Détermine si les limites sont dans la plage de valeurs.

méthode	type	arguments	description
[constructeur](debut, fin, exclusif)	constructeur	debut: la borne inférieure, fin: la borne supérieure, exclusif: inclure les limites dans la plage?	Crée un objet plage de valeurs, allant de debut à fin . Il est important de noter que debut et fin doivent être des objets du même type, ayant une méthode succ() .
include(valeurRecherchee)	instance	valeurRecherchee: la valeur qu'on recherche	Vérifie si la valeur donnée fait partie de la plage de valeur. Retourne true ou

méthode	type	arguments	description
			false.

5-B-6 - L'objet Class

L'objet **Class** est utilisé pour déclarer les autres classes de cette bibliothèque. Utiliser cet objet en créant une nouvelle classe permet à la nouvelle classe d'avoir une méthode **initialize()** qui fait office de constructeur.

Voir l'exemple ci-dessous :

```
//déclaration de la classe
var MaClasseExemple = Class.create();

//définition du reste de l'implémentation de la classe
MaClasseExemple.prototype = {
  initialize: function(message) {
    this.message = message;
  },
  afficheMessage: function() {
    alert(this.message);
  }
};

//maintenant, utilisons un de ces objets
var monBavard = new MaClasseExemple('Bonjour tout le monde.');
```

méthode	type	arguments	description
create(*)	instance	(tous)	Définit le constructeur pour la nouvelle classe.

5-B-7 - L'objet Ajax

Cet objet sert de base et d'espace de nom pour plusieurs autres classes qui apportent des fonctionnalités AJAX.

propriété	type	description
activeRequestCount	Number	Le nombre de requêtes Ajax en cours.

méthode	type	arguments	description
getTransport()	instance	(aucun)	Retourne un nouvel objet XMLHttpRequest .

5-B-7-a - L'objet Ajax.Responders

Hérite de *Enumerable*

Cet objet maintient une liste d'objets qui vont être appelés quand un évènement de type Ajax sera déclenché. Vous pouvez utiliser cet objet, par exemple, pour définir un gestionnaire d'erreur global pour les opérations AJAX.

propriété	type	description
responders	Array	Le tableau des objets enregistrés pour les notifications Ajax.

méthode	type	arguments	description
register(repondeurAjoute)	instance	repondeurAjoute : objet avec les méthodes qui vont être appelées	L'objet passé dans l'argument repondeurAjoute doit contenir des méthodes nommées d'après les événements Ajax (c.a.d. onCreate , onComplete , onException , etc.) Quand l'évènement correspondant est déclenché tous les objets enregistrés qui possèdent la méthode avec le nom approprié verront cette méthode appelée.
unregister(repondeurSupprime)	instance	repondeurSupprime : objet à supprimer de la liste	L'objet passé dans l'argument repondeurSupprime sera supprimé de la liste des objets enregistrés.
dispatch(evenementRetour, requete, transport, json)	instance	evenementRetour : nom de l'évènement ajax déclenché, requete : l'objet Ajax.Request responsable de l'évènement, transport: l'objet XMLHttpRequest qui a véhiculé (ou vehicule) l'appel AJAX, json: l'entete X-JSON de la réponse (si présent)	Parcourt la liste des objets enregistrés, cherchant ceux qui ont la méthode donnée par l'argument evenementRetour . Ensuite, chacune de ces méthodes est appelée en passant en arguments les 3 autres paramètres. Si la réponse AJAX contient une entête HTTP X-JSON avec du contenu JSON, alors il sera évalué et passé dans l'argument json . Si l'évènement est onException l'argument transport contiendra l'exception et l'argument json ne sera pas utilisé.

5-B-7-b - La classe Ajax.Base

Cette classe est utilisée comme classe parente de la plupart des autres classes définies par l'objet [Ajax](#).

méthode	type	arguments	description
setOptions(options)	instance	options : options Ajax	Met en place les options demandées pour l'opération AJAX.

méthode	type	arguments	description
responselsSuccess()	instance	(aucun)	Renvoie true si l'opération AJAX a réussi, false sinon.
responselsFailure()	instance	(aucun)	L'inverse de responselsSuccess() .

5-B-7-c - La classe Ajax.Request

Hérite de *Ajax.Base*

Encapsule les opérations AJAX.

propriété	type	nature	description
Events	Array	statique	Liste des statuts/événements possibles reportés pendant une requête AJAX. Cette liste contient: 'Uninitialized', 'Loading', 'Loaded', 'Interactive', et 'Complete'.
transport	XMLHttpRequest	instance	L'objet XMLHttpRequest qui véhicule l'opération AJAX.
url	String	instance	L'URL cible de la requête.

méthode	type	arguments	description
[constructeur](url, options)	constructeur	url: l'URL à appeler, options: options AJAX	Crée une instance de cet objet qui appellera l' url donnée en utilisant les options données. L'évènement onCreate sera appelé pendant l'appel au constructeur. Important : Il est bon de noter que l'URL choisie est sujette à la politique de sécurité du navigateur. Dans beaucoup de cas, les navigateurs ne chercheront pas une URL sur un autre domaine que celui de la page courante. Idéalement, vous devez utiliser uniquement des URLs locales pour éviter d'avoir à modifier la configuration du navigateur de l'utilisateur. (Merci Clay).
evalJSON()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est utilisée en interne

méthode	type	arguments	description
			pour évaluer le contenu d'un éventuel header HTTP X-JSON présent dans la réponse AJAX.
evalResponse()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Si la réponse AJAX a un header Content-type text/javascript , le contenu de la réponse sera évalué par cette méthode.
header(nom)	instance	nom : nom du header HTTP	Retourne le contenu d'un header HTTP de la réponse AJAX. Appelez cette méthode quand la requête AJAX est terminée.
onStateChange()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est appelée par l'objet lui-même quand le statut de l'objet AJAX change.
request(url)	instance	url : URL de l'appel AJAX	Cette méthode ne doit pas être utilisée en externe. Elle est appelée par le constructeur.
respondToReadyState(readyState)	instance	readyState : numéro de l'état (1 à 4)	Cette méthode ne doit pas être utilisée en externe. Elle est appelée par l'objet lui-même quand le statut de l'objet AJAX change.
setRequestHeaders()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est appelée par l'objet lui-même pour mettre en place les entêtes HTTP envoyés dans la requête HTTP.

5-B-7-d - L'objet/argument options

Une partie importante des opérations AJAX provient de l'argument **options**. Il n'y a pas de classe **options** à proprement parler. Tout objet peut être utilisé, tant qu'il a les propriétés voulues. Il est courant de créer un objet anonyme pour les appels AJAX.

propriété	type	défaut	description
parameters	String	"	La liste des valeurs (formatées pour l'URL) passée à la requête.
asynchronous	Boolean	true	Indique que l'appel AJAX va être asynchrone.
postBody	String	indéfini	Contenu passé dans le corps de la requête dans le

propriété	type	défaut	description
			cas d'une requête POST.
requestHeaders	Array	indéfini	Liste des entêtes HTTP passés à la requête. Cette liste doit avoir un nombre pair d'éléments, chaque élément impair est le nom d'un entête et l'élément suivant est la valeur de cet entête. Exemple :['mon-entete1', 'voici la valeur', 'mon-autre-entete', 'autre valeur'].
onXXXXXXXX	Function(XMLHttpRequest, Object)	indéfini	Fonction personnalisée à appeler quand l'évènement/statut correspondant est atteint durant un appel AJAX. Par exemple var mesOptions = {onComplete: afficheReponse, onLoad: enregistreCharge} ; La méthode utilisée recevra un argument contenant l'objet XMLHttpRequest qui transporte l'opération AJAX et un second argument contenant les entêtes HTTP X-JSON évalués.
onSuccess	Function(XMLHttpRequest, Object)	indéfini	Fonction personnalisée à appeler quand l'appel AJAX finit normalement. La méthode utilisée recevra un argument contenant l'objet XMLHttpRequest qui transporte l'opération AJAX et un second argument contenant les entêtes HTTP X-JSON évalués.
onFailure	Function(XMLHttpRequest, Object)	indéfini	Fonction personnalisée à appeler quand l'appel AJAX finit avec un échec. La méthode utilisée recevra un argument contenant l'objet XMLHttpRequest qui transporte l'opération AJAX et un second argument contenant les entêtes HTTP X-JSON évalués.
onException	Function(Ajax.Request,	indéfini	Fonction personnalisée à

propriété	type	défaut	description
	exception)		appeler quand une exception survient du côté client de l'appel AJAX, comme une réponse non valide ou des arguments non valides. La méthode utilisée recevra deux arguments contenant l'objet Ajax.Request qui englobe l'opération AJAX ainsi que l'exception.
insertion	une classe Insertion	indéfini	Une classe qui détermine comment le nouveau contenu sera inséré. Cela peut être Insertion.Before , Insertion.Top , Insertion.Bottom ou Insertion.After . S'applique uniquement aux objets Ajax.Updater .
evalScripts	Boolean	indéfini, faux	Détermine si les blocs de script seront évalués au retour de la réponse. S'applique uniquement aux objets Ajax.Updater .
decay	Number	indéfini, 1	Détermine le ralentissement progressif dans la vitesse de rafraichissement de l'objet Ajax.PeriodicalUpdater quand la réponse reçue est la même que la précédente. Par exemple, si vous mettez 2, après qu'une mise à jour produise le même résultat que la précédente, l'objet attendra deux fois plus avant la prochaine mise à jour. Si cela se répète à nouveau, l'objet attendra quatre fois plus longtemps, et ainsi de suite. Laissez-le indéfini ou mettez 1 pour empêcher le ralentissement.
frequency	Number	indéfini, 2	Intervalle (pas fréquence) entre les rafraichissements, en secondes. S'applique uniquement aux objets Ajax.PeriodicalUpdater .

5-B-7-e - La classe Ajax.Updater

Hérite de [Ajax.Request](#)

Utilisée quand l'URL requêtée retourne du HTML que vous voulez injecter directement dans un élément donné de votre page. Vous pouvez également utiliser cet objet quand l'URL retourne des blocs **<script>** qui seront évalués au retour. Utilisez l'option **evalScripts** pour utiliser les scripts.

propriété	type	description
containers	Objet	Cet objet contient deux propriétés: containers.success sera utilisé quand l'appel AJAX sera un succès, et containers.failure utilisé autrement.

méthode	type	arguments	description
[constructeur](conteneur, url, options)	constructeur	conteneur: peut être l'id d'un élément, l'élément lui-même, ou un objet avec deux propriétés : un élément (ou id) object.success qui sera utilisé en cas de succès de l'appel AJAX et un élément (ou id) object.failure qui sera utilisé autrement. url: l'URL à appeler, options: options AJAX	Crée une instance de cet objet qui appellera l' url donnée en utilisant les options données.
updateContent()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est utilisée en interne au retour de la réponse. Elle met à jour l'élément approprié avec le contenu HTML ou appellera la fonction passée dans l'option insertion . La méthode sera appelée avec deux arguments, l'élément à mettre à jour et la réponse texte.

5-B-7-f - La classe [Ajax.PeriodicalUpdater](#)*Hérite de [Ajax.Base](#)*

Cette classe instancie et utilise un objet [Ajax.Updater](#) pour rafraichir un élément de la page ou pour effectuer tout autre tâche de [Ajax.Updater](#). Vérifiez la référence de [Ajax.Updater](#) pour plus d'informations.

propriété	type	description
container	Objet	Cette valeur sera passée directement au constructeur de Ajax.Updater .

propriété	type	description
url	String	Cette valeur sera passée directement au constructeur de Ajax.Updater .
frequency	Number	Intervalle (pas fréquence) entre les mises à jour, en secondes. Le défaut est de 2 secondes. Ce nombre sera multiplié par le decay courant en appelant l'objet Ajax.Updater .
decay	Number	Conserve le niveau de pourrissement (decay) courant quand la tâche est ré-exécutée.
updater	Ajax.Updater	Le dernier objet Ajax.Updater utilisé.
timer	Object	Le timer Javascript utilisé pour notifier à cet objet l'instant de la prochaine mise à jour.

méthode	type	arguments	description
[constructeur](conteneur, url, options)	constructeur	conteneur: peut être l'id d'un élément, l'élément lui-même ou un objet avec deux propriétés : un élément (ou id) object.success qui sera utilisé en cas de succès de l'appel AJAX et un élément (ou id) object.failure qui sera utilisé autrement. url: l'URL à appeler, options: options AJAX	Crée une instance de cet objet qui appellera l' url donnée en utilisant les options données.
start()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est utilisée en interne pour démarrer les tâches périodiques.
stop()	instance	(aucun)	Arrête l'exécution des tâches périodiques de l'objet. Après son arrêt, l'objet appellera la méthode donnée dans l'option onComplete (le cas échéant).
updateComplete()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est appelée par l'objet Ajax.Updater courant à la fin de sa requête. Elle permet de prévoir la prochaine exécution.
onTimerEvent()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est utilisée en interne à l'instant d'une nouvelle

méthode	type	arguments	description
			mise à jour.

5-B-8 - L'objet Element

Cet objet propose des méthodes utilitaires pour manipuler les éléments du DOM.

méthode	type	arguments	description
addClassName(element, nomClasse)	instance	element: un élément ou son identifiant, nomClasse : nom d'une classe CSS	Ajoute le nom de classe donné aux classes appliquées à l'élément donné.
classNames(element)	instance	element: un élément ou son identifiant	Retourne un objet Element.ClassNames représentant les noms de classes CSS associées à l'élément donné.
cleanWhitespace(element)	instance	element: un élément ou son identifiant	Supprime tous les noeuds texte enfants de l'élément avec un contenu blanc.
empty(element)	instance	element: un élément ou son identifiant	Retourne une valeur Boolean indiquant si le corps de l'élément est blanc.
getDimensions(element)	instance	element: un élément ou son identifiant	Retourne les dimensions de l'élément. La valeur retournée est un objet avec deux propriétés : height (hauteur) et width (largeur).
getHeight(element)	instance	element: un élément ou son identifiant	Retourne la hauteur (offsetHeight) de l'élément.
getStyle(element, proprieteCss)	instance	element: un élément ou son identifiant, proprieteCss : le nom d'une propriété CSS (les formats prop-nom et propNom fonctionnent)	Retourne la valeur de la propriété CSS donnée pour l'élément ou null si la propriété est absente.
hasClassName(element, nomClasse)	instance	element: un élément ou son identifiant, nomClasse : nom d'une classe CSS	Retourne true (vrai) si l'élément a le nom de classe donnée dans les classes qui lui sont appliquées.
hide(elem1 [, elem2 [, elem3 [...]]])	instance	elemN: un élément ou son identifiant	Cache chacun des éléments en mettant style.display à 'none' .
makeClipping(element)	instance	element: un élément ou son identifiant	Supprime le défilement de l'élément donné.
makePositioned(element)	instance	element: un élément ou son identifiant	Change la propriété style.position de l'élément à 'relative' .
remove(element)	instance	element: un élément ou	Supprime l'élément du

méthode	type	arguments	description
		son identifiant	document.
removeClassName(element, nomClasse)	instance	element: un élément ou son identifiant, nomClasse : nom d'une classe CSS	Supprime le nom de classe des classes appliquées à l'élément.
scrollTo(element)	instance	element: un élément ou son identifiant	Fait défiler la fenêtre jusqu'à l'élément donné.
setStyle(element, hashProprieteCss)	instance	element: un élément ou son identifiant, hashProprieteCss : objet Hash avec les styles à appliquer	Change les propriétés CSS de l'élément d'après les valeurs de l'argument hashProprieteCss .
show(elem1 [, elem2 [, elem3 [...]]])	instance	elemN: un élément ou son identifiant	Affiche chacun des éléments en mettant style.display à ".".
toggle(elem1 [, elem2 [, elem3 [...]]])	instance	elemN: un élément ou son identifiant	Inverse la visibilité de chacun des éléments donnés.
undoClipping(element)	instance	element: un élément ou son identifiant	Rétablit le défilement de l'élément donné qui avait été supprimé avec makeClipping() .
undoPositioned(element)	instance	element: un élément ou son identifiant	Vide la propriété style.position de l'élément à ".".
update(element, html)	instance	element: un élément ou son identifiant, html : contenu HTML	Remplace le contenu HTML interne de l'élément avec l'argument html donné. Si le HTML donné contient des blocs <script> , ils ne seront pas inclus, mais évalués.
visible(element)	instance	element: un élément ou son identifiant	Retourne une valeur Boolean indiquant si l'élément est visible.

5-B-8-a - La classe Element.ClassNames

Hérite de [Enumerable](#)

Représente la collection des noms de classes CSS associées à un élément.

méthode	type	arguments	description
[constructeur](element)	constructeur	element: un élément de l'arbre DOM ou son id	Crée une instance de cet objet représentant les noms de classes CSS de l'élément donné.
add(nomClasse)	instance	nomClasse: un élément de l'arbre DOM ou son id	Ajoute le nom de classe CSS à la liste des classes appliquées à l'élément.
remove(nomClasse)	instance	nomClasse: un élément de l'arbre DOM ou son id	Supprime le nom de classe CSS de la liste des classes appliquées à l'élément.

méthode	type	arguments	description
set(nomClasse)	instance	nomClasse: un élément de l'arbre DOM ou son id	Associe le classe CSS à l'élément en supprimant toutes les autres classes CSS appliquées.

5-B-9 - L'objet Abstract

Cet objet sert de base pour d'autres classes de la bibliothèque. Il n'a ni propriétés, ni méthodes. Les classes définies dans cet objet sont également traitées comme des classes abstraites traditionnelles.

5-B-9-a - La classe Abstract.Insertion

Cette classe est utilisée comme classe de base pour d'autres classes qui proposent de l'insertion dynamique d'éléments. Cette classe est utilisée comme une classe abstraite.

méthode	type	arguments	description
[constructeur](element, contenu)	constructeur	element: un objet élément ou son id, contenu : HTML à insérer	Crée une instance de cet objet qui aidera à l'insertion dynamique du contenu.
contentFromAnonymousTable	instance	(aucun)	

propriété	type	nature	description
adjacency	String	statique, paramètre	Paramètre qui spécifie comment le contenu sera placé par rapport à l'élément. Les valeurs possibles sont : 'beforeBegin' (avant le début), 'afterBegin' (après le début), 'beforeEnd' (avant la fin) et 'afterEnd' (après la fin).
element	Object	instance	L'objet élément par rapport à qui l'insertion sera effectuée.
content	String	instance	Le contenu HTML qui sera inséré.

5-B-10 - L'objet Insertion

Cet objet sert de base pour d'autres classes de la bibliothèque. Il n'a ni propriétés, ni méthodes.

5-B-10-a - La classe Insertion.Before

Hérite de *Abstract.Insertion*

Insère du contenu HTML avant un élément.

méthode	type	arguments	description
[constructeur](element, contenu)	constructeur	element: un objet élément ou son id, contenu : HTML à insérer	Hérité de Abstract.Insertion . Créé une instance de cet objet qui aidera à l'insertion dynamique du contenu.

Le code suivant :

```
<br>Bonjour, <span id="personne" style="color:red;">Wiggum. Comment ça va?</span>
<script> new Insertion.Before('personne', 'Chef '); </script>
```

Produira le HTML suivant :

```
<br>Bonjour, Chef <span id="personne" style="color:red;">Wiggum. Comment ça va?</span>
```

5-B-10-b - La classe Insertion.Top

Hérite de [Abstract.Insertion](#)

Insère du contenu HTML comme premier enfant de l'élément. Le contenu sera directement après la balise d'ouverture de l'élément.

méthode	type	arguments	description
[constructeur](element, contenu)	constructeur	element: un objet élément ou son id, contenu : HTML à insérer	Hérité de Abstract.Insertion . Créé une instance de cet objet qui aidera à l'insertion dynamique du contenu.

Le code suivant :

```
<br>Bonjour, <span id="personne" style="color:red;">Wiggum. Comment ça va?</span>
<script> new Insertion.Top('personne', 'M. '); </script>
```

Produira le HTML suivant :

```
<br>Bonjour, <span id="personne" style="color:red;">M. Wiggum. Comment ça va?</span>
```

5-B-10-c - La classe Insertion.Bottom

Hérite de [Abstract.Insertion](#)

Insère du contenu HTML comme dernier enfant de l'élément. Le contenu sera juste avant la balise fermante de l'élément.

méthode	type	arguments	description
[constructeur](element, contenu)	constructeur	element: un objet élément ou son id, contenu : HTML	Hérité de Abstract.Insertion . Créé

méthode	type	arguments	description
		à insérer	une instance de cet objet qui aidera à l'insertion dynamique du contenu.

Le code suivant :

```
<br>Bonjour, <span id="personne" style="color:red;">Wiggum. Comment ça va?</span>
<script> new Insertion.After('personne', ' Que ce passe-t-il?'); </script>
```

Produira le HTML suivant :

```
<br>Bonjour, <span id="personne" style="color:red;">Wiggum. Comment ça va? Que ce passe-t-il?</span>
```

5-B-10-d - La classe Insertion.After

Hérite de [Abstract.Insertion](#)

Insère du contenu HTML après la balise de fermeture de l'élément.

méthode	type	arguments	description
[constructeur](element, contenu)	constructeur	element: un objet élément ou son id, contenu : HTML à insérer	Hérité de Abstract.Insertion . Créé une instance de cet objet qui aidera à l'insertion dynamique du contenu.

Le code suivant :

```
<br>Bonjour, <span id="personne" style="color:red;">Wiggum. Comment ça va?</span>
<script> new Insertion.After('personne', ' Etes-vous là?'); </script>
```

Produira le HTML suivant :

```
<br>Bonjour, <span id="personne" style="color:red;">Wiggum. Comment ça va?</span> Etes-vous là?
```

5-B-11 - L'objet Field

Cet objet propose des méthodes utilitaires pour travailler avec les champs des formulaires.

méthode	type	arguments	description
clear(champ1 [, champ2 [, champ3 [...]]])	instance	champN: un element champ de formulaire ou son id	Supprime la valeur de chacun des champs de formulaire passés.
present(champ1 [, champ2 [, champ3 [...]]])	instance	champN: un element champ de formulaire ou son id	Retourne true uniquement si tous les champs de formulaire passés contiennent des valeurs non vides.

méthode	type	arguments	description
focus(champ)	instance	champ: un element champ de formulaire ou son id	Déplace le focus sur le champ donné.
select(champ)	instance	champ: un element champ de formulaire ou son id	Sélectionne la valeur du champ, si le champ supporte la sélection.
activate(champ)	instance	champ: un element champ de formulaire ou son id	Déplace le focus et sélectionne la valeur du champ, si le champ supporte la sélection.

5-B-12 - L'objet Form

Cet objet propose des méthodes utilitaires pour travailler avec les formulaires et leurs champs.

méthode	type	arguments	description
serialize(formulaire)	instance	formulaire: un élément formulaire ou son id	Retourne une liste des champs et de leurs valeurs, formatés pour une URL, du type 'champ1=valeur1&champ2=valeur2'
findFirstElement(formulaire)	instance	formulaire: un élément formulaire ou son id	Retourne le premier élément champ (non inactif) du formulaire.
getElements(formulaire)	instance	formulaire: un élément formulaire ou son id	Retourne une Array contenant tous les champs de saisie du formulaire.
getInputs(formulaire[,nomType[,nom]])	instance	formulaire: un élément formulaire ou son id, nomType : le type des champs input, nom : le nom du champ	Retourne une Array contenant tous les éléments <input> du formulaire. La liste peut être éventuellement filtrée sur les attributs type et name .
disable(formulaire)	instance	formulaire: un élément formulaire ou son id	Désactive tous les champs du formulaire.
enable(formulaire)	instance	formulaire: un élément formulaire ou son id	Active tous les champs du formulaire.
focusFirstElement(formulaire)	instance	formulaire: un élément formulaire ou son id	Place le focus sur le premier élément visible et actif du formulaire.
reset(formulaire)	instance	formulaire: un élément formulaire ou son id	Réinitialise le formulaire. Identique à l'appel de la méthode reset() de l'élément form.

5-B-12-a - L'objet Form.Element

Cet objet propose des méthodes utilitaires pour travailler avec les éléments de formulaires, visibles ou non.

méthode	type	arguments	description
serialize(element)	instance	element: un objet élément ou son id	Retourne la paire nom/valeur de l'élément, de type 'nomElement=valeurElement' .
getValue(element)	instance	element: un objet élément ou son id	Retourne la valeur de l'élément.

5-B-12-b - L'objet Form.Element.Serializers

Cet objet propose des méthodes utilitaires utilisées en interne pour extraire les valeurs courantes des éléments de formulaire.

méthode	type	arguments	description
inputSelector(element)	instance	element: un objet ou son id d'un élément de formulaire ayant la propriété <i>checked</i> comme un bouton radio ou une case à cocher	Retourne un Array avec le nom et la valeur de l'élément du type ['nomElement', 'valeurElement'] .
textarea(element)	instance	element: un objet ou son id d'un élément de formulaire ayant la propriété <i>value</i> comme un champ texte, un bouton ou un champ mot de passe	Retourne un Array avec le nom et la valeur de l'élément du type ['nomElement', 'valeurElement'] .
select(element)	instance	element: un objet élément représentant un select	Retourne un Array avec le nom de l'élément et toutes les valeurs sélectionnées de l'élément du type ['nomElement', 'optSelect1 optSelect4 optSelect9'] .

5-B-13 - La classe Abstract.TimedObserver

Cette classe est utilisée comme classe de base pour les autres classes qui surveillent un élément jusqu'à ce que sa valeur (ou toute autre propriété que la classe dérivée définit) change. Cette classe est utilisée comme une classe abstraite.

Des surclasses peuvent être créées pour surveiller des choses comme la valeur entrée dans un élément ou une des propriétés de style, ou le nombre de lignes dans une table ou tout ce dont vous voulez suivre les changements.

Les classes dérivées doivent implémenter la méthode `getValue()` pour déterminer la valeur courante surveillée dans cet élément.

méthode	type	arguments	description
[constructeur](element, frequence, rappel)	constructeur	element: un objet élément ou son id, frequence : intervalle en secondes, rappel : méthode appelée	Crée un objet qui surveillera l'élément.

méthode	type	arguments	description
		quand l'élément change.	
registerCallback()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est appelée par l'objet lui-même pour débiter la surveillance de l'élément.
onTimerEvent()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est utilisée en interne pour vérifier périodiquement la valeur de l'élément.

propriété	type	description
element	Objet	L'élément qui est surveillé.
frequency	Number	C'est en fait l'intervalle en secondes entre deux vérifications.
callback	Function(Object, String)	Cette méthode est appelée quand la valeur de l'objet change. Elle recevra l'élément et sa nouvelle valeur.
lastValue	String	La valeur lors de la dernière vérification de l'objet.

5-B-13-a - La classe Form.Element.Observer

Hérite de [Abstract.TimedObserver](#)

Implémentation d'un [Abstract.TimedObserver](#) qui surveille la valeur des champs d'entrées d'un formulaire. Utilisez cette classe quand vous voulez surveiller un élément qui n'a pas d'évènement qui signale le changement de valeur. Sinon, vous pouvez utiliser la classe [Form.Element.EventObserver](#) à la place.

méthode	type	arguments	description
[constructeur](element, frequence, rappel)	constructeur	element: un objet élément ou son id, frequence : intervalle en secondes, rappel : méthode appelée quand l'élément change.	Hérité de Abstract.TimedObserver . Créé un objet qui surveille la propriété value de l'élément.
getValue()	instance	(aucun)	Retourne la valeur de l'élément.

5-B-13-b - La classe Form.Observer

Hérite de [Abstract.TimedObserver](#)

Implémentation d'un [Abstract.TimedObserver](#) qui surveille la valeur de tous les champs d'entrées d'un formulaire. Utilisez cette classe quand un des éléments du formulaire n'a pas d'évènement qui signale le changement de valeur. Sinon, vous pouvez utiliser la classe [Form.EventObserver](#) à la place.

méthode	type	arguments	description
[constructeur](element, frequence, rappel)	constructeur	element: un objet élément ou son id, frequence : intervalle en secondes, rappel : méthode appelée quand l'élément change.	Hérité de Abstract.TimedObserver . Créé un objet qui surveillera les valeurs des champs de formulaire.
getValue()	instance	(aucun)	Retourne la sérialisation de tous les éléments du formulaire.

5-B-14 - La classe [Abstract.EventObserver](#)

Cette classe est utilisée comme classe de base pour d'autres classes qui doivent exécuter une méthode quand un évènement de type changement de valeur survient sur un élément.

Plusieurs objets de type [Abstract.EventObserver](#) peuvent être liés à un même élément, sans qu'ils se détruisent l'un l'autre. Les méthodes seront exécutées dans l'ordre d'assignation à l'élément.

L'évènement déclenchant est **onclick** pour les boutons radio et les checkboxes, **onchange** pour les champs texte en général et les listes déroulantes.

Les classes dérivées doivent implémenter la méthode `getValue()` pour déterminer la valeur courante surveillée dans cet élément.

méthode	type	arguments	description
[constructeur](element, rappel)	constructeur	element: un objet élément ou son id, rappel : méthode appelée quand l'élément change.	Créé un objet qui surveillera les valeurs des champs de formulaire.
registerCallback()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est utilisée en interne pour lier l'objet au gestionnaire d'évènement de l'élément.
registerFormCallbacks()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. Elle est utilisée en interne pour lier l'objet à chacun des éléments du formulaire.
onElementEvent()	instance	(aucun)	Cette méthode ne doit pas être utilisée en externe. C'est la méthode qui est liée au gestionnaire d'évènement de l'élément.

propriété	type	description
element	Objet	L'élément qui est surveillé.
callback	Function(Object, String)	Cette méthode est appelée quand la valeur de l'objet change. Elle recevra

propriété	type	description
		l'élément et sa nouvelle valeur.
lastValue	String	La valeur lors de la dernière vérification de l'objet.

5-B-14-a - La classe `Form.Element.EventObserver`

Hérite de [Abstract.EventObserver](#)

Implémentation d'un [Abstract.EventObserver](#) qui exécute une méthode sur l'évènement approprié d'un champ de formulaire pour détecter les changements de cet élément. Si l'élément n'expose pas de gestionnaire d'évènement qui signale les changements, vous pouvez utiliser la classe [Form.Element.Observer](#) à la place.

méthode	type	arguments	description
[constructeur](element, rappel)	constructeur	element: un objet élément ou son id, rappel : méthode appelée quand l'évènement survient.	Hérité de Abstract.EventObserver . Créé un objet qui surveillera la propriété value de l'élément.
getValue()	instance	(aucun)	Retourne la valeur de l'élément.

5-B-14-b - La classe `Form.EventObserver`

Hérite de [Abstract.EventObserver](#)

Implémentation d'un [Abstract.EventObserver](#) qui surveille l'ensemble des champs d'un formulaire à travers leurs gestionnaires d'évènements. Si le formulaire contient des champs n'ayant pas de gestionnaire qui signale les changements, vous pouvez utiliser la classe [Form.Observer](#) à la place.

méthode	type	arguments	description
[constructeur](form, rappel)	constructeur	form: un objet formulaire ou son id, rappel : méthode appelée quand l'évènement survient.	Hérité de Abstract.EventObserver . Créé un objet qui surveillera les changements du formulaire.
getValue()	instance	(aucun)	Retourne la sérialisation de tous les champs du formulaire.