Quelques bibliothèques JavaScript simples et légères

API de validation de formulaire "*validanguage*"

M. Libes

journée JoSy
3 Nov. 2011

Quelques bibliothèques JavaScript

- Un site intéressant
  - *http://javascript.developpez.com/cours/librairies-javascript-vraiment-utiles*

- Différentes bibliothèques groupées par thèmes
  - Animation
  - ***Traitement des Dates et heures***
  - ***Validation de formulaires***
  - Graphiques et diagrammes
  - Tables HTML
  - Gestion des images
  - Maths....

Quelques bibliothèques JavaScript de validation de formulaire

Validation de formulaire

- **LiveValidation**

Cette librairie open-source légère vous permet d'effectuer des validations de champs de formulaire facilement et efficacement. Deux versions sont disponibles : une basée sur le *framework* Prototype et une autonome.

- **wForms**

Librairie non intrusive qui offre toutes les validations usuelles de formulaires sans avoir besoin de connaissances en programmation.

- **Validanguage**

Validanguage est une librairie open-source, non intrusive à gestion d'héritage, développée pour devenir le framework de validation de formulaires de référence.

- Autres librairies : **Yav**, **qForms JavaScript API**.

bibliothèques JavaScript de validation de formulaire

- LiveValidation
  - http://www.livevalidation.com/

- ***Validanguage***
  - http://www.drlongghost.com/validanguage.php

  "*Validanguage*" est une librairie open-source, développée pour devenir le framework de validation de formulaires de référence.

API de validation de formulaire

- Démo rapide, pour illustrer les fonctionnalités offertes
  - http://www.com.univ-mrs.fr/ssc/info/JS

Installation de l'API « validanguage »

- URL de téléchargement
  - *http://www.drlongghost.com/vd_downloads/103/validanguage.zip*
- Récupération possible en
  - version traduite « française » et autres langues
  - version compressée ou pas
- Déposer le ficher dans un répertoire du serveur WEB (path relatif à la racine du site web)
- Charger la bibliothèque dans la section *<head> .. </head>* et à l'intérieur de balises *<script> .. </script>* du fichier HTML

Utilisation de l'API « validanguage »

- Tous les éléments (champs, container...) d'un formulaire DOIVENT avoir un « ***id*** »...
  - Champ <*input id=id_du_champ name=... type=radio* >
  - Container
    - <span *id=id_du_champ*>..
    - <div *id=id_du_champ* >..

- Les fonctions de validation de l'API vont s'appliquer aux champs désignés par leur *« id »* (qui doit être unique)

  - <input ***id="nom"*** name="nom" type="text" class="champ_nom">

Utilisation de l'API « validanguage »

- 2 types d'API : 2 syntaxes et 2 modes d'utilisation
- ***API "comment-based"*** : On place les fonctions, les évenements, les paramètres ***dans des balises HTML en commentaires***

```
<!-- <validanguage target="id_du_champ" required="true"
    errorMsg="Champ obligatoire"> -->
```

- ***API orientée objet*** avec une syntaxe plus proche d'un LOO
  - Qui se place entre des balises <*script*> </*script*>

```
<script type= "text/javascript" >
    validanguage.el.<id_du_champ> = {
                <suite de règles et fonctions de validation> }
</script>
```

Fonctionnement de l'API « validanguage »

Les validations de l'API utilisent l'objet « ***validanguage.el*** » pour lancer une série de contrôles, de validations de la saisie ou de transformations de la saisie sur un champ du formulaire

```
<script type= ”text/javascript” >
    validanguage.el.date: {
        characters: {
        mode: 'allow',
        expression: 'numeric/-',
        suppress: true,
        errorMsg: 'caractères invalides'
        }
    };
</script>
```

Contrôle la saisie des caractères dans un champ dont l'id est « ***date*** » et vérifie que les caractères autorisés sont des numériques [0-9] ou « / » ou « - »

Principe général « validanguage »

- Le principe : des contrôles de validations sont lancés sur la valeur saisie dans champs lors de certains événements
  - ***required*** : true|false
  - ***expression :*** numeric | alpha | special | alphaUPPER
  - ***maxlength | minlength :*** <longueur>
  - ***suppress :*** true|false
  - ***mode :*** deny|allow
  - ***requiredAlternatives*** : pour choisir dans plusieurs choix de checkbox
  - ***regex :*** <une expression reguliere>
  - ***characters :*** contrôle sur les caractères rentrés
  - ***validation :*** lance des fonctions de validation
  - ***Transformation :*** lance des fonctions de transformation du texte
- etc...

Les déclencheurs d'Evénements « validanguage »

- onsubmit="true" lance la validation au moment quand le formulaire entier est envoyé...Si la validation échoue le formulaire n'est pas envoyé
- onblur=”true” : lance la validation quand on sort d'un champ
- onchange=”true” : lance la validation lors d'un changement dans une boite de sélection
- onclick=”true” : se déclenche quand on rentre dans un champ
- onkeydown=”true” : se déclenche quand on appuye une touche clavier
- onkeyup=”true” : se déclenche quand on relache une touche clavier
- ontyping=”true” : se déclenche après une *certaine durée* de saisie dans un champ
  - Après la durée de ***validanguage.settings.typingDelay***

API « validanguage » validation des caractères

- Validation de la saisie des ***caractères*** dans un champ

```
44          <input id="nom" name="nom" type="text" class="textinput"/>
45          <span id="nom_error" class="formerror"></span>
46          <script type="text/javascript" >
47          validanguage.el.nom = {
48              characters: { mode: 'allow', expression: 'alpha- ',
49                            onsubmit: true, onblur: true,
50                            required: true, errorMsg: 'caracteres invalides',
51                            onerror: 'showError', suppress: false
52                          },
```

- N'autorise que des caractères alphabétique [a-z-]
  - Diverses classes prévues
    - *numeric* [0-9], *alphaUPPER* [A-Z], *special* [,;:!#..]
- Contrôle la saisie en sortant du champ et lors du « *submit* » général
- ***required: true*** → Champ obligatoire

API « validanguage » fonctions de validation

- Validanguage possède un ensemble de fonctions de validation prédéfinies... très appréciables !
  - permettent de contrôler des champs de nature diverse

    - *validanguage.**validateDate***
    - *validanguage.**validateEmail***
    - *validanguage.**validatePasswordStrength***
    - *validanguage.**validateURL***
    - *validanguage.**validateNumeric***
    - *validanguage.**validateCreditCard***
    - *validanguage.**validateUSPhoneNumber***

API « validanguage » fonctions de validation

- Exemple contrôle de Dates *validanguage.**validateDate***
  - Controle les dates dans le futur ou le passé
  - Le format *jj/mm/aaaa* ou *aaaa/jj/mm*
  - L'année sur 2 ou 4 digits
  - Les délimiters **/ -**

```
120   <script type="text/javascript" >
121     validanguage.el.date  = {
122       validations: [
123         { name: "validanguage.validateDate( text, { dateOrder: 'dmy', rejectDatesInTh
    eFuture: true, twoDigitYearsAllowed: true, allowedDelimiters: '/' } )",
124           errorMsg: "format de date incorrect", onerror: 'showError', onsuccess:"remo
    veError",
125           onsubmit: true,
126           onblur: true
127         } ],
128       errorMsg: 'Date Invalide', onerror: 'showError', onsuccess:"removeError"
129       };
130   </script>
```

API « validanguage » fonctions de transformation

- Validanguage permet d'utiliser des fonctions de ***transformation*** du texte saisi
- Permet de formater l'information dans les champs de saisie comme on le souhaite
  - exemple formater un numéro de téléphone automatiquement en (0)4 91-22-44-99
  - Convertir automatiquement du texte en Majuscules
- Nb: Les transformations sont effectuées avant les validations

```
53        transformations: [
54            { onkeyup: true,
55              name: "validanguage.substituteText('lower', 'upper')"
56            }
57        ],
```

API « validanguage » fonctions de transformation

- formater un numéro de téléphone automatiquement en
  - (04)-91-41-77-99
  - Une bonne transformation facile évite une mauvaise regexp difficile ;-)

```
48  <script type="text/javascript" >
49    validanguage.el.phone = {
50      characters: { mode: 'allow', expression: 'numeric-() ',
51        onsubmit: true, required:true, onkeyup: true,
52        errorMsg: "Que des chiffres", suppress:false },
53      //regex: { onsubmit: true, onblur: true, expression: '\\d{2,2}-\\d{2,2}-\\d{2,2}
  -\\d{2,2}-\\d{2,2}' },
54      transformations: [ { onblur: true, name: "validanguage.format('(xx)-xx-xx-xx-xx'
  , '-() '  )" } ],
55      onsubmit:true, onblur:true, required: true,
56      errorMsg: 'format de numero de telephone invalide', onerror: 'showError', onsucc
  ess:"removeError"
57      };
58  </script>
```

API « validanguage » gestion des erreurs

- Pour afficher un texte d'erreur à proximité du champ où la validation d'une saisie a échoué, on se sert du DOM
  - On crée un container   quelque part dans le formulaire... près du champ à contrôler...

```
<input id="nom" name="nom" type="text" class="textinput"/>
<span id="nom_error" class="formerror"> ici </span>
```

  - les messages d'erreur seront affichés dans ce container « *nom_error* », grâce à une fonction d'écriture qui désignera et écrira dans ce container

API « validanguage » gestion des erreurs

- Les handlers « *onError* » et « *onSuccess* » permettent de lancer 2 fonctions différentes pour gérer une erreur : par exemple afficher ou effacer un message d'erreur

```
<input id="nom" name="nom" type="text" class="textinput"/>
<span id="nom_error" class="formerror"></span>
<script type="text/javascript" >
validanguage.el.nom = {
    characters: { mode: 'allow', expression: 'alpha- ',
                  onsubmit: true, onblur: true,
                  required: true, errorMsg: 'caracteres invalides',
                  onerror: 'showError', suppress: false
                },
```

gestion des erreurs

- La fonction javascript ***getElementById*** permet de désigner l'id (« ***this.id*** ») du container courant, et on affiche du texte (« errorMsg ») avec la fonction ***innerHTML***

```
function showError(errorMsg) {
   errorMsg="<font size=-1 color=red><EM>"+errorMsg+"</font></EM>";
   document.getElementById( this.id+"_error" ).innerHTML = errorMsg;
}
```

- Idem sur l'événement « ***onSuccess*** », on appelle la fonction « ***removeError*** » qui efface le message d'erreur dans le container

```
function removeError( ) {
   document.getElementById( this.id + '_error' ).innerHTML = '';
}
```

API « validanguage » Conclusions

- Une API riche en fonctionnalités, qui permet de
  - gagner beaucoup de temps de programmation grâce aux validations et transformations prédéfinies
  - Réaliser des interfaces de programmation ergonomiques (je n'ai pas parlé des éléments qu'on peut rendre visibles ou invisibles)

- Néanmoins... une syntaxe pas toujours très facile
  - Attention à la multitude d'options qui parfois peuvent se contrecarrer
- Bien lire la documentation, pas toujours claire sur le fonctionnement... analyser les exemples de démo fournis