JavaScript
Regex*

IFNTI Sokodé – L3

04 Mai 2021

1 NTUI javascript et les formulaires

JavaScript est très souvent utilisé pour vérifier les données d'un formulaires avant de les envoyées aux serveurs.

Nous avons vu comment répondre aux événement de type `change` mais aussi de type `click`
Certaines données sont faciles à vérifier (champs vide, valeur minimum/maximum). Mais d"autres peutvent être plus complexe (email syntaxiquement correcte).
Je vous renvoie rapidement vers la documenttaion de w3school [1] qui est bien faite pour donner les méthodes de validation de chaine de caractère.

Un outils très puissant pour la validation de chaine de caractère sont les **expressions régulières** souvent appelé par le petit nom (regex pour **reg**ular **exp**ression). Il s'agit d'un sujet complexe. Nous aborderons ici une courte introduction.

2 Regex

Les expressions régulières sont des schémas ou des motifs utilisés pour effectuer des recherches et des remplacements dans des chaines de caractères.

Ces schémas ou motifs sont tout simplement des séquences de caractères dont certains vont disposer de significations spéciales et qui vont nous servir de schéma de recherche. Concrètement, les expressions régulières vont nous permettre de vérifier la présence de certains caractères ou suites de caractères dans une expression.

En JavaScript, les expressions régulières sont avant tout des objets appartenant à l'objet global constructeur RegExp. Nous allons donc pouvoir utiliser les propriétés et méthodes de ce constructeur avec nos expressions régulières.

Notez déjà que nous n'allons pas être obligés d'instancier ce constructeur pour créer des expressions régulières ni pour utiliser des méthodes avec celles-ci.

Nous allons également pouvoir passer nos expressions régulières en argument de certaines méthodes de l'objet String pour effectuer des recherches ou des remplacements dans une chaine de caractère.

2.1 Notre première RegExp

Nous disposons de deux façons de créer nos expressions régulières en JavaScript : on peut soit déclarer nos expressions régulières de manière littérale, en utilisant des slashs comme caractères d'encadrement, soit appeler le constructeur RegExp().

De manière générale, on préfèrera comme souvent utiliser une écriture littérale tant que possible pour des raisons de performance.

*Repris des cours de Pierre Giraud et d'openclassroom

1. `https://www.w3schools.com/js/js_string_methods.asp`

```
let masque1 = /Pierre/;
let masque2 = new RegExp('Pierre');
```

Dans le code ci-dessus, on définit deux expressions régulières en utilisant les deux méthodes décrites précédemment. On les enferme dans des variables masque1 et masque2. Notez que les termes « masque de recherche », « schéma de recherche » et « motif de recherche » seront utilisés indifféremment et pour décrire nos expressions régulières par la suite.

Dans cet exemple, nos deux expressions régulières disposent du même motif qui est le motif simple /Pierre/. Ce motif va nous permettre de tester la présence de « Pierre » c'est-à-dire d'un « P » suivi d'un « i » suivi d'un « e » suivi d'un « r » suivi d'un autre « r » suivi d'un « e » dans une chaîne de caractères.

2.2 Méthode match() de l'objet String

La méthode match() de l'objet String va nous permettre de rechercher la présence de caractères ou de séquences de caractères dans une chaîne de caractères.

Pour cela, nous allons lui passer un objet représentant une expressions régulière en argument et match() va renvoyer un tableau avec les correspondances entre notre masque et la chaîne de caractères c'est-à-dire un tableau contenant des caractères ou séquences de caractères trouvés dans la chaîne de caractères qui satisfont à notre masque de recherche.

Dans le cas où aucune correspondance n'est trouvée, match() renverra la valeur null.

Notez que la méthode match() ne renvoie par défaut que la première correspondance trouvée. Pour que match() renvoie toutes les correspondances, il faudra utiliser l'option ou « drapeau » g qui permet d'effectuer des recherches globales.

ex : `chaine.match(masque)`

2.3 Exemples avec masques de classe

Nous avons évidemment le masque littéral que nous avons vu (`masque1 = /Pierre/`).
Nous allons utiliser une classe de caractères : rechercher toute lettre majuscule qui se situe dans l'intervalle « A-Z », c'est-à-dire en l'occurrence n'importe quelle lettre majuscule de l'alphabet (lettres accentuées ou avec cédille exclues.

Cela se fait avec la régex `/[A-Z]/`. Pour l'appliquer de manière globale il faut donc ajouter le drapeaux « g » soit `/[A-Z]/g`.

Expérimentons avec une chaine de caractère et dans chacun des paragraphe (p1-p2 et p3) les résultats de match des 3 masque.

2.4 Méthode search() de l'objet String

La méthode search() est légèrement différente. Elle permet d'effectuer une recherche dans une chaîne de caractères à partir d'une expression régulière fournie en argument.

Cette méthode va retourner la position à laquelle a été trouvée la première occurrence de l'expression recherchée dans une chaîne de caractères ou -1 si rien n'est trouvé.

Remplaçons les match() par des search() dans le code précédent pour regarder cette différence.

2.5 Méthode replace() de l'objet String

La méthode replace() permet de rechercher un caractère ou une séquence de caractères dans une chaîne et de les remplacer par d'autres caractère ou séquence. On va lui passer une expression régulière et une expression de remplacement en arguments.

Cette méthode renvoie une nouvelle chaîne de caractères avec les remplacements effectués mais n'affecte pas la chaîne de caractères de départ qui reste inchangée.

Tout comme pour match(), seule la première correspondance sera remplacée à moins d'utiliser l'option g dans notre expression régulière.

Expérimentons ! !

2.6 Méthode split() de l'objet String

La méthode split() permet de diviser ou de casser une chaîne de caractères en fonction d'un séparateur qu'on va lui fournir en argument.

Cette méthode va retourner un tableau de sous chaînes créé à partir de la chaîne de départ. La chaîne de départ n'est pas modifiée.

2.7 Les méthodes exec() et test() de l'objet RegExp

La méthode exec() de RegExp va rechercher des correspondances entre une expression régulière et une chaîne de caractères.

Cette méthode retourne un tableau avec les résultats si au moins une correspondance a été trouvée ou null dans le cas contraire.

La méthode test() de RegExp va également rechercher des correspondances entre une expression régulière et une chaîne de caractères mais va cette fois-ci renvoyer le booléen true si au moins une correspondance a été trouvée ou false dans le cas contraire.

ATTENTION Ici, exec et test sont des méthodes des regexp donc la syntaxe c'est `masque.test(chaine)`

3 Validation html

Depuis HTML version 5, il est possible d'ajouter de la validation directement dans le code HTML, sans avoir besoin d'écrire la moindre ligne de JavaScript.

Pour cela, différents attributs sont ajoutés et permettent d'empêcher la soumission d'un formulaire si toutes les validations ne sont pas respectées.

3.1 L'attribut type pour les inputs

Pour valider les informations saisies dans une balise input , il est possible d'utiliser l'attribut type.

L'attribut type de la balise input ne prend pas seulement comme valeurs text et password . Cela peut aussi être email , tel , URL , date et bien d'autres.

Lorsque vous ajoutez un élément input avec un attribut type="email" , le navigateur empêchera la soumission du formulaire si ce n'est pas une adresse email correcte.

3.2 Les attributs de validation simples

En fonction du type de l' input , vous pouvez utiliser différents attributs pour perfectionner votre validation :
- **min / max** : fonctionne avec des champs de type nombre ou date. Cela permet de définir une valeur minimum et une valeur maximum autorisées ;
- **required** : fonctionne avec à peu près tous les types de champs. Cela rend obligatoire le remplissage de ce champ ;
- step : fonctionne avec les dates ou les nombres. Cela permet de définir une valeur d'incrément lorsque vous changez la valeur du champ via les flèches ;
- **minlength / maxlength** : fonctionne avec les champs textuels (text , url , tel , email ...). Cela permet de définir un nombre de caractères minimum et maximum autorisé.

3.3 le patern

Il est possible d'utiliser les regexp directement dans les balises html avec l'attribut `pattern`
exemple : `<input type="text" pattern="/Pierre/" />` empêchera à l'utilisateur de rentrer autre chose que « Pierre ». OK ce n'est pas très pratique mais avec un masque plus complexe cela est très puissant.

3.4 Exercice

A partir du code téléchargé :

1. nous souhaitons dans un premier temps valider le champ Code du formulaire. A chaque lettre saisie dans le champ ayant pour ID code nous voulons vérifier que la valeur du champ commence bien par CODE- grâce à une Regex que voici : `/^CODE-/`. Si la valeur commence bien par CODE- alors nous affichons dans l'élément ayant pour ID code-validation : Code valide, sinon nous affichons dans cet élément Code invalide.
2. Maintenant que nous savons si notre code est valide ou non, nous voudrions griser (grâce à l'attribut disabled) le bouton de soumission (L'input de type submit ayant pour ID submit-btn) quand le code est invalide, et le dégriser quand le code est valide.
   Cela signifie que nous allons devoir ajouter un attribut disabled="true" au bouton de soumission quand le code est invalide. Et supprimer cet attribut disabled quand le code est valide.
3. Finalement, nous avons un champ Email et nous voudrions le rendre obligatoire et obliger l'utilisateur à entrer une adresse email valide. Il faudra aussi empêcher le formulaire de se soumettre s'il n'est pas valide.
   Mais nous voudrions faire tout ça uniquement avec le HTML5, sans utiliser de code JavaScript.
   Vous pouvez changer le type du champ email plutôt que d'utiliser une Regex via pattern