KIT DE SURVIE JAVASCRIPT - TD #3

Pierre BIASUTTI
IUT Informatique, Licence Pro

1 Plugin jQuery

1.1 Les plugins jQuery

jQuery est une bibliothèque Javascript constituée de nombreuses fonctions de bases, ainsi que de plugins. Ces plugins permettent notamment d'étendre les fonctionnalités de jQuery à la manipulation poussée d'objet du DOM par exemple (jQuery UI, https://jqueryui.com/)

Exercice 1. A l'aide de jQuery UI (à télécharger à l'adresse ci-dessus):
a) Créez un **div** conteneur au bords noirs, et un autre **div** dont la couleur de fond est rouge. Rendre le **div** rouge "draggable" (cf. jQuery UI). Afficher ces coordonnées en temps réel dans un autre élément du DOM. On souhaite que lorsque l'on déplace le **div** rouge, les coordonnées se mettent à jour.
b) Créez une liste d'éléments. Rendre cette liste "sortable" (cf. jQuery UI). Créez un bouton permettant d'afficher l'ordre de la liste lorque l'on clique dessus.

1.2 Créer un plugin jQuery

Un plugin simple Afin de simplifier l'écriture de nouveaux plugins, et d'assurer ainsi l'extension de la bibliothèque, jQuery permet l'écriture de plugin très simplement. Toutes les objets jQuery reposent sur le ***jQuery.fn***. De ce fait, rajouter une fonction à cet objet revient à rajouter une méthode à jQuery directement (cf td01, rappel sur les objets). On peut donc créer une nouvelle méthode (plugin) de la manière suivante:

Exemple

```
1 jQuery.fn.nouvelleFonction = function(parametre){
2         console.log(parametre);
3 }
4 ...
5 $('a').nouvelleFonction('Du texte'); // affiche ''Du texte''
```

Conflits Certaines bibliohtèque Javascript utilisent aussi le symbole $ comme alias pour certains objets. Utiliser cet alias dans l'écriture d'un nouveau plugin peut ainsi mener à des problèmes de conflits. Afin d'éviter tout problème lié à une potentielle confusion au niveau de l'interpreteur, il est courant de définir un plugin jQuery dans un contexte particulier, comme montré ci-après:

Exemple

```
1 (function($){
2         $.fn.nouvelleFonction(...){...};
3 })(jQuery);
```

Ici, le symbole $ ne peut définir que l'objet jQuery, réglant tout problème de conflit.

Standard jQuery et écriture chainée Comme vous avez pu le voir dans les TDs précédents, les fonctions jQuery retournent souvent des Objets jQuery afin de permettre une écriture chainée des instructions. Pour maintenir la possibilité d'écrire de manière chainée après l'appel d'un nouveau plugin, on adopte généralement l'écriture suivante:

Exemple

```
(function($){
        $.fn.nouvelleFonction(parametre){
                this.each(function(){
                        ...
                });

                return this;
        };
})(jQuery);
```

Ici, on itère le code sur chaque élément du sélécteur, et on renvoie la liste des éléments du sélécteur une fois le traitement terminé.

Exercice 2. Ecrire un plugin jQuery qui agrandit la police d'un élément du sélécteur à chaque fois qu'il est survolé.

2 AngularJS

AngularJS (https://angularjs.org/) est une bibliothèque Javascript conçue pour palier le manque de dynamicité du HTML. Cette bibliothèque permet notamment d'étendre le "vocabulaire" HTML pour créer un environnement dynamique, lisible et rapide à développer. La plupart des autres bibliothèques Javascript (comme jQuery) permettent de rendre les pages plus dynamiques en proposant des surcouches aux méthodes déjà présentes en Javascript. Ici, l'objectif de AngularJS est de simplifier les manipulation entre Javascript et DOM, en considérant la page web comme une application, dans laquelle le HTML ne sert que de récipient (vue) aux données.

Notez que AngularJS existe maintenant sous le nom d'Angular (https://angular.io/), qui est un framework complet dérivé du Javascript permettant de générer des applications selon un pattern MVC bien séparé. Cependant, sa mise en place ainsi que son fonctionnement dépassent le spectre de ce cours.

Pour tous les exercices de cette partie, nous utiliserons la version d'AngularJS disponible à l'adresse suivante: *https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js*.

2.1 Les bases d'AngularJS

Exemple

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js">
</script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p> // Peut aussi s'écrire <p>{{ name }}</p>
</div>

</body>
</html>
```

Le code ci-dessus présente un exemple simple de code avec l'utilisation d'AngularJS. On y découvre plusieurs **directives**:

- *ng-app:* définie le nom de l'application
- *ng-model:* lie le contenu d'un élément HTML de formulaire à une variable de l'application

- *ng-bind:* lie le contenu d'une variable à un élément HTML. Notez que l'écriture alternative de *ng-bind* ({{ expression }}) permet d'inclure le contenu d'une variable n'importe ou dans la page.

Ici, la variable ***name*** est une variable Javascript implicitement déclaré par AngularJS. Nous verrons plus tard comment y accéder.

Exercice 3. Ecrivez un script contenant un champ de texte (input texte). A l'aide d'AngularJS, faites en sorte que la couleur de fond du champ de texte change lorsque le contenu du champ correspond à une couleur CSS correcte. Exemple: si le champ contient "red", la couleur du fond est rouge, si le champ contient "toto" le fond n'a pas de couleur car "toto" n'est pas une couleur valide.

2.2 Des cas plus complexes

Initialiser des variables avec AngularJS AngularJS est principalement basé sur le modèle MVC. Ainsi, le code HTML représente la vue comme montré précédemment. Le controlleur doit être, quant à lui, défini en Javascript.

Exemple

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js">
</script>
<body>

<div ng-app="app01" ng-controller="controlleur01">
  <p>{{ name }}</p>
</div>

<script>
        var app = angular.module("app01", []);

        app.controller("controlleur", function($scope){
                $scope.name = "Jonathan";
        });
</script>

</body>
</html>
```

Le code ci-dessus montre un exemple d'initialisation de variable dans le controlleur, puis d'affichage dans la vue. De sorte à simplifier les exemples, vues et controlleurs sont définis dans le même fichier. En pratique, les deux entités sont le plus souvent séparées. Ici on définie un nom pour l'application (directive: ***ng-app***) et un nom pour le controlleur associé à cette vue (directive: ***ng-controller***). Le code Javascript qui suit permet de récupérer le module sur lequel on souhaite travailler, puis définie un comportement pour le controlleur. L'objet ***$scope*** représente alors l'environnement de la vue, à laquelle on peut donc ajouter des membres.

Tableaux Une variable du "scope" d'AngularJS peut aussi représenter un tableau. Dès lors, on peut parcourir celui-ci à l'aide de la directive ***ng-repeat***.

Exemple

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js">
</script>
<body>

<div ng-app="app01" ng-controller="controlleur01">
  <ul>
          <li ng-repeat="x in nombres">
            {{ x }}
          </li>
  </ul>
</div>

<script>
        var app = angular.module("app01", []);

        app.controller("controlleur", function($scope){
                $scope.nombres = [1 1 2 3 5 8 13 21];
        });
</script>

</body>
</html>
```

Fonctions On peut aussi associer une fonction à une variable du "scope" de la manière suivante:

Exemple

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js">
</script>
<body>

<div ng-app="app01" ng-controller="controlleur01">
  {{ ajouter1(12) }}
</div>

<script>
        var app = angular.module("app01", []);

        app.controller("controlleur", function($scope){
                $scope.ajouter = function(nbr){ return parseInt(nbr) + 1 };
        });
</script>

</body>
</html>
```

Exercice 4. Créez une vue contenant un formulaire avec les éléments suivants:

- un champ “nombre1” (input:text)
- un champ “nombre2” (input:text)
- une liste “operateur” (select, dont les options sont +, -, *, /)

On souhaite maintenant que lorsqu'un nombre est entré dans chaque champ de texte, le résultat de l'opération s'affiche quelque part dans la page.
Exemple: si “nombre1” vaut 12 et “nombre2” vaut 9, et opérateur et sur l'élement “*”, alors on doit afficher $9 * 12 = 108$.

AngularJS et AJAX AngularJS propose une extension des méthodes AJAX native de Javascript, de la même manière que jQuery. Cette extension permet le chargement de données distantes très simplement.

Exemple

```
app.controller("controlleur01", function($scope, $http){
        $http.get("page.json").then(function(response){
                ... // Response contient le résutlat de la requête
        });
});
```

Exercice 5. Le fichier json a l'adresse suivante contient la liste des pokémons en Anglais:
https://raw.githubusercontent.com/fanzeyi/Pokemon-DB/master/pokedex.json.
A l'aide d'AngularJS, écrivez un script permettant de récupérer cette liste et d'afficher le nom de chaque pokémon.

A ce stade, vérifiez vos codes avec le chargé de TD

Exercice 6. Gestion des erreurs dans un formulaire avec AngularJS. Demandez au chargé de TD

Exercice 7. Plugin d'autocomplétion jQuery. Demandez au chargé de TD

Exercice 8. Application Todo liste en Javascript. Demandez au chargé de TD