RÉSUMÉ 6 - GESTION DES ÉVÉNEMENTS AVEC JAVASCRIPT

Lien vers l'activité : Gestion des événements avec javascript

www.infoforall.fr - Dernière modif. : 26 01 2020

GESTION DES ÉVÉNEMENTS

1 - LOGIQUE DE PROGRAMMATION ÉVÉNEMENTIELLE

La **programmation séquentielle** vient de fait qu'un programme se déroule ligne par ligne et que même dans une ligne certaines évaluations sont prioritaires sur d'autres.

Dans cette logique, le déroulé du programme est prévisible.

La **programmation événementielle** fonctionne différemment : on programme séquentiellement une interface. Lors de cette programmation, on déclenche la surveillance de certaines événements.

Le programme fonctionne alors sur le principe suivant :

- On déclare certaines surveillances d'événements (comme si on clique sur cette image, il faut lancer la fonction bidule)
- TANT QUE la variable générale de boucle est affectée à VRAIE
    - On surveille les événements définis plus haut et on déclenche les fonctions correspondantes si l'événement a lieu.

Cette fois, c'est l'utilisateur qui va imposer le déroulé du programme. On ne sait pas à l'avance sur quoi il va cliquer ou le texte qu'il va taper.

2 - LISTE DES ÉVÉNEMENTS

- `click` : on clique et on relâche sur la balise
- `dblclick` : pareil mais en clic double.
- `mousedown` : on clique gauche sans relâcher sur la balise.
- `mouseup` : on relâche gauche sur la balise.
- `mouseover` : on survole la balise.
- `mousemove` : on déplace la souris sur l'élément.
- `mouseout` : on fait sortir la souris de la balise.
- `keydown` : on appuie sans relâcher sur l'une des touches.
- `keyup` : on relâche une touche.
- `keypress` : on appuie et relâche une touche.
- `focus` : on vient de mettre le focus ou cibler l'élément, par exemple en cliquant sur l'élément.
- `blur` : on annule ce ciblage, par exemple en cliquant ailleurs.

- `load` : la balise est correctement chargée.

3 - CRÉATION D'UNE SURVEILLANCE D' ÉVÉNEMENTS : DANS LE HTML

- Cette méthode présente l'avantage de la simplicité.
- Le désavantage vient du fait qu'on ne peut pas désactiver une surveillance : le déclenchement est codé en dur dans le code.
- Attention : il faut rajouter **on** devant l'événement
  - l'événement `load` est associé à l'attribut `onload` en HTML
  - ...
- Attention : on transmet ici la fonction javascript à activer sous forme d'un string qui sera ensuite interprété : on peut donc transmettre des arguments

Quelques exemples de cette fonctionnalité.

Exemple 1 **onload** - déclenchement d'une fonction après que la page soit entièrement chargée :

```
HTML  <body onload="demarrage()">
```

```
JS01  function demarrage() {
JS02      alert("Hello World !");
JS03  }
```

Exemple 2 **onclick** - modification d'une largeur d'image après un click sur l'image :

```
HTML  <img id="toto" onclick="image_cliquee_1()" src="accueil_HTML5_150.png">
```

```
JS01  function image_cliquee_1() {
JS02      var reference_balise = document.getElementById('toto');
JS03      reference_balise.width = 200;
JS04  }
```

Lien trinket : **https://trinket.io/html/dda3068394**

Remarque : si vous n'imposez que la largeur ou la hauteur à une balise **inline-block**, la balise reste proportionnée correctement. Ici, la modification de la largeur provoque ainsi la modification automatique de la hauteur.

Remarque 2 : si vous voulez que les images soient redimensionnées après chargement de l'image, il suffit de mettre ceci dans le HTML :

```
HTML  <img id="toto" onload="image_cliquee_1()" src="accueil_HTML5_150.png">
```

4 - CRÉATION D'UNE SURVEILLANCE D' ÉVÉNEMENTS : DANS LE JAVASCRIPT

- Cette méthode présente l'avantage de la clarté : on ne mélange pas le HTML et le javascript : toute la gestion est dans le javascript
- Cette méthode permet de n'activer la gestion de l'événement qu'à partir d'un certain moment
- Cette méthode permet de désactiver l'événement lorsqu'on ne veut plus qu'il soit géré
- On doit placer :
  - la référence de la balise à surveiller

- un point
- la méthode **addEventListener**
- un argument-string correspondant au type d'événement
- un argument correspondant à la référence d'une fonction en mémoire : uniquement son nom, pas de parenthèses et donc pas de possibilité* de transmettre un argument qu'on pourrait stocker dans un paramètre
- (*) nous verrons plus tard avec les fonctions lambda que c'est vrai mais un peu faux !

Exemple 1 **load** - déclenchement d'une fonction après que la page soit entièrement chargée :

```
JS01  function demarrage() {
JS02      alert("Hello World !");
JS03  }
JS04  window.addEventListener("load",demarrage);
```

EXEMPLE 2 CLICK - DÉCLENCHEMENT D'UNE FONCTION APRÈS UN CLIC SUR UNE IMAGE :

```
HTML  <img id="toto"  src="accueil_HTML5_150.png">
```

```
JS01  function image_cliquee_1() {
JS02      var reference_balise = document.getElementById('toto');
JS03      alert(reference_balise.src);
JS04      reference_balise.width = 200;
JS05  }
JS06  document.getElementById('toto').addEventListener("click",image_cliquee_1);
```

On voit donc qu'ici, on ne doit que fournir un **id** dans la balise HTML. On donne l'ordre de gérer cette surveillance directement dans Javascript. Du coup, on peut également annuler la surveillance en utilisant le code suivant :

```
JS01  document.getElementById('toto').removeEventListener("click",image_cliquee_
```

THIS (HORS PROGRAMME POUR LA NSI)

Pratique pour les projets mais hors programme pour les sujets papiers (je pense)

THIS DANS LE HTML

Cet opérateur contient la référence de l'objet au sein duquel cet opérateur est évalué. Cette notion de 'lieu d'exécution' est loin d'être anécdotique à comprendre. Voici des exemples avec l'utilisation de **this** lorsqu'il est présent dans la valeur d'un attribut HTML. Ici, c'est facile : le contexte d'exécution est la balise elle-même puisque le **this** est lu dans la balise.

```
HTML  <img onclic="image_cliquee(this)" src="accueil_HTML5_150.png">
HTML  <img onload="image_cliquee(this)" src="accueil_js_150.png">
```

On voit qu'on lance l'appel à la fonction **image_cliquee** en fournissant un argument, ici **this**.

La fonction doit donc avoir un paramètre pour recevoir la référence **this**.

```
JS01  function image_cliquee(reference_balise) {
JS02      reference_balise.height = 200;
JS03  }
```

Comme on peut le voir, inutie de faire une recherche de référence : la référence de la balise activant la référence est automatiquement placée dans **this** . Du coup, pas besoin d'**id** dans les balises HTML.

THIS DANS LE JAVASCRIPT

Si on active la gestion des événements dans le javascript, il faut encore garder des **id** si on veut utiliser la méthode **getElementById**. Nous verrons comment nous en passer dans l'activité sur les tableaux de la partie DONNEES.

Lorsqu'on gère un événement avec **addEventListener**, l'opérateur **this** contient la référence de l'objet sur lequel on applique **addEventListener**.

```
HTML  <img id="image_1" src="accueil_HTML5_150.png">
HTML  <img id="image_2" src="accueil_js_150.png">
```

```
JS01  function image_cliquee() {
JS02          this.width = 100;
JS03  }
JS04
JS05  document.getElementById('image_1').addEventListener("click",image_cliquee)
JS06  document.getElementById('image_2').addEventListener("click",image_cliquee)
```