JavaScript et HTML

Cours 7

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/lp-js

Plan

- tables HTML (suite)
- exemple (calculette)
- callbacks, async, promises
- exemple (liste sélective)

HTML exemple (1/3)

Une page HTML

Un sous-titre

Un paragraphe normal mais avec du **g r a s** et de l'*italique* ainsi qu'un logo JS: dans une autre page

ou directement ici

- un premier item d'une liste non numérotée
- et le deuxième

Inscrits	IFT3220	IFT6810
Femmes	6	7
Hommes	44	16

HTML

- l'élément `table` se décompose en lignes `<tr>` (*table rows*)
- chaque ligne est composée de en-têtes `<th>` (*table headers*) ou de données `<td>` (*table data*)

Inscrits	IFT3220	IFT6810
Femmes	6	7
Hommes	44	16

```
<table width="180" border="1" cellspacing="2" cellpadding="0">
   <tr>
     <th class="typeDonnee">Inscrits</th> <th>IFT3220</th> <th>IFT6810</th>
   </tr>
   <tr>
      <th>Femmes</th> <td>6</td> <td>7</td>
   </tr>
   <tr>
      <th>Hommes</th> <td>44</td> <td>16</td>
   </tr>
</table>
```

HTML

- l'élément `table` se décompose en lignes `<tr>` (*table rows*)
- chaque ligne est composée de en-têtes `<th>` (*table headers*) ou de données `<td>` (*table data*)

Inscrits	IFT3220	IFT6810
Femmes	6	7
Hommes	44	16
Total		73

```
<table width="180" border="1" cellspacing="2" cellpadding="0">
   <tr>
     <th class="typeDonnee">Inscrits</th> <th>IFT3220</th> <th>IFT6810</th>
   </tr>
   <tr>
      <th>Femmes</th> <td>6</td> <td>7</td>
   </tr>
   <tr>
      <th>Hommes</th> <td>44</td> <td>16</td>
   </tr>
   <tr>
       <th>Total</th> <td colspan="2">73</td>
   </tr>
</table>
```

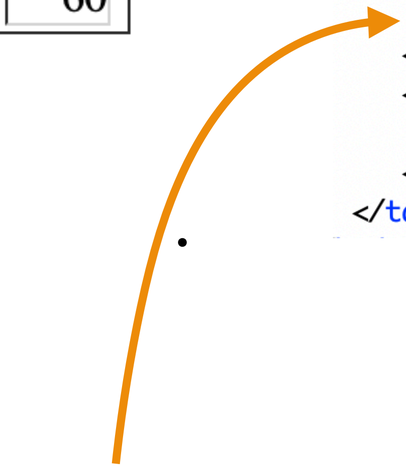
extension sur 2 colonnes

HTML

- l'élément `table` se décompose en lignes `<tr>` (*table rows*)
- chaque ligne est composée de en-têtes `<th>` (*table headers*) ou de données `<td>` (*table data*)

Inscrits	IFT3220	IFT6810	Total
	6	7	13
	44	16	60

```
<table width="180" border="1" cellspacing="2" cellpadding="0">
   <tr>
     <th class="typeDonnee">Inscrits</th> <th>IFT3220</th> <th>IFT6810</th> <th>Total</th>
     <th
   </tr>
   <tr>
      <th rowspan="2"></th> <td>6</td> <td>7</td> <td>13</td>
   </tr>
   <tr>
      <td>44</td> <td>16</td> <td>60</td>
   </tr>
</table>
```



Calculette

- dans `calc.html` il y une `table` avec le style `calc.css` et le scri*pt* `calc.js`

```
<html>
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <title>Calculette</title>
        <link rel="stylesheet" href="calc.css" type="text/css" charset="utf-8">
        <script type="text/javascript" charset="utf-8" src="calc.js"></script>
    </head>


    <body id="calculette">
        <table border="1" cellspacing="5" cellpadding="5">
            <tr><th id="display" colspan="4">Header</th></tr>
            <tr><td>C</td><td>&lArr;</td><td>/</td><td>*</td></tr>
            <tr><td>7</td><td>8</td>     <td>9</td><td>-</td></tr>
            <tr><td>4</td><td>5</td>     <td>6</td><td>+</td></tr>
            <tr><td>1</td><td>2</td>     <td>3</td><td rowspan="2">=</td></tr>
            <tr><td colspan="2">0</td><td>.</td></tr>
        </table>
    </body>
</html>
```

Header			
C	⇐	/	*
7	8	9	-
4	5	6	+
1	2	3	=
0		.	

Calculette

- le style `calc.css`

```css
html,body {
    height:100%;
    text-align:center;
}


table {
    margin-left:auto; margin-right:auto;
    font-size:25pt;
    font-weight: bold;
    background-color: black;
    font-family: Arial, "MS Trebuchet", sans-serif;
}

th {
    margin: 0px;
    background-color:grey;
    color: #EEE;
    text-align: right;
    height: 50px;
    padding: 5px 0px;
    max-width:300px;
}
```

```css
td {
    margin: auto auto;
    background-color: #EEE;
    color: green;
    text-align: center;
    padding: 10px;
    cursor: pointer;
    width:75px;
}

td:hover {
    background-color: #DDD;
}

td:active {
    background-color: #AAA;
}
```

Header			
C	⇐	/	*
7	8	9	-
4	5	6	+
1	2	3	=
0		.	

Calculette

- le style `calc.js`

```
let display; // texte de l'affichage lors du "load"

function calc(){
    try {
        display.nodeValue =  eval(display.nodeValue);
    } catch (e) {
        display.nodeValue = "erreur";
    }
}

function clear(){
    display.nodeValue="";
}

function backspace(){
    display.nodeValue = display.nodeValue.slice(0,-1);
}

function addKey(key){
    display.nodeValue += key;
}
```

```
function process(key){
    switch (key){
        // clés spéciales
        case "C": clear();     break;
        case "\b": backspace(); break;
        case "=": calc();      break;
        // clés ordinaires
        case "/": case "*":
        case "7": case "8": case "9": case "-":
        case "4": case "5": case "6": case "+":
        case "1": case "2": case "3":
        case "0": case ".": addKey(key);
    }
}

function keyClicked(event){
    process(this.firstChild.nodeValue);
}

function keyPressed(event){
    if(event.keyCode == 8){
        process("\b");
    } else if (event.keyCode == 13){
        process("=");
    } else
        process(String.fromCharCode(event.charCode));
    event.preventDefault();
}
```

`display.nodeValue` | `3 + 5 * 9 - 11 / 7`

Calculette

- le style `calc.js`

```
function initEventHandlers(){
    display = document.getElementById("display").firstChild;
    var tdList = document.getElementsByTagName("td");
    for (var i=0;i<tdList.length;i++)
        tdList.item(i).addEventListener("click",keyClicked,false);
}

window.addEventListener("load",initEventHandlers,false);
window.addEventListener("keypress",keyPressed,false);
```

Fonctions *callbacks*

- les continuations sont appelées *callbacks* en JavaScript

```
function additionner (x1, x2, myCallback) {
    let r = x1 + x2;
    myCallback(r);
  }

additionner (5, 5, console.log);
```

```
function afficherDemo (s) {
    document.getElementById("demo").innerHTML = s;
  }

additionner (5, 5, afficherDemo);
```

[la fonction `additionner` est indépendante de la fonction d'affichage]

Functions asynchrones

- un exemple de *callback* avec setTimeout (pour un *timeout*)

```
setTimeout(loveYou, 3000);

function loveYou() {
  console.log ("I love You !!")
}
```

- ou avec setInterval pour appels répétitifs

```
setInterval(tictac, 1000);

function tictac() {
  let d = new Date();
  document.getElementById("demo").innerHTML=
  d.getHours() + ":" +
  d.getMinutes() + ":" +
  d.getSeconds();
}
```

Functions asynchrones

- ou attente de la fin de lecture d'un fichier

```
function getFile (myCallback) {
    let req = new XMLHttpRequest();
    req.open('GET', "mycar.html");
    req.onload = function() {
      if (req.status == 200) {
        myCallback(this.responseText);
      } else {
        myCallback("Error: " + req.status);
      }
    }
    req.send();
}

getFile(afficherDemo);
```

lancer la lecture du fichier `mycar.html`

- voir `http://www.w3schools.com/js/js_asynchronous.asp`

  pour une description détaillée

Functions asynchrones

- les objets `Promise` permettent de distinguer un callback « succès » et un callback « échec »

```
let myPromise = new Promise (function (myResolve, myReject) {
    let req = new XMLHttpRequest();
    req.open('GET', "mycar.html");
    req.onload = function() {
      if (req.status == 200) {
        myResolve(req.response);
      } else {
        myReject("File not Found");
      }
    };
    req.send();
});

myPromise.then (afficherDemo, console.log))
```

- voir `http://www.w3schools.com/js/js_promise.asp`

  pour une description détaillée

Functions asynchrones

- les mots clés `async` et `await` sont des notations commodes pour les `Promise`

```
async function getFile() {
    let myPromise = new Promise(function(resolve) {
      let req = new XMLHttpRequest();
      req.open('GET', "mycar.html");
      req.onload = function() {
        if (req.status == 200) {
          resolve(req.response);
        } else {
          resolve("File not Found");
        }
      };
      req.send();
    });
    document.getElementById("demo").innerHTML = await myPromise;
  }

getFile()
```

- voir `http://www.w3schools.com/js/js_promise.asp`

  pour une description détaillée

Listes sélectives

- interaction avec des listes par CSS ou par JS

Test d'affichage sélectif d'une liste

Liste avec ouverture/fermeture en CSS

- item1
- item2
- item 3

Liste avec ouverture/fermeture en javascript

- item1...
- item2...
- item3...

Test d'affichage sélectif d'une liste

Liste avec ouverture/fermeture en CSS

- item1
  - sous-item 1.1
  - sous-item 1.2
  - sous-item 1.3
- item2
- item 3

Liste avec ouverture/fermeture en javascript

- item1...
- item2...
- item3...

Listes sélectives

```
<!DOCTYPE html>
<html>

<head>
  <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
  <link rel="stylesheet" href="slists.css" type="text/css" charset="utf-8"/>
  <title>Test d'affichage sélectif</title>
  <script type="text/javascript" charset="utf-8" src="slists.js"></script>
</head>
```

```
<body>
  <h1>Test d'affichage sélectif d'une liste</h1>

  <h2>Liste avec ouverture/fermeture en CSS</h2>
  <ul>
  <li class="dynCSS">item1
      <ul>
      <li>sous-item 1.1</li>
      <li>sous-item 1.2</li>
      <li>sous-item 1.3</li>
      </ul>
  </li>
  <li class="dynCSS">item2
      <ul>
      <li>sous-item 2.1</li>
      <li>sous-item 2.2</li>
      </ul>
  </li>
  <li class="dynCSS">item 3</li>
  </ul>
```

```
  <h2>Liste avec ouverture/fermeture en javascript</h2>
  <ul>
  <li><span class="toggle">item1</span><span style="display:inline">...</span>
      <ul>
      <li>sous-item 1.1</li>
      <li>sous-item 1.2</li>
      <li>sous-item 1.3</li>
      </ul>
  </li>
  <li><span  class="toggle">item2</span><span style="display:inline">...</span>
      <ul>
      <li><a href="#test2.1">sous-item 2.1</a></li>
      <li><a href="#test2.2">sous-item 2.2</a></li>
      </ul>
  </li>
  <li><span class="toggle">item3</span><span style="display:inline">...</span></li>
  </ul>

</body>
</html>
```

Listes sélectives

```
<body>
  <h1>Test d'affichage sélectif d'une liste</h1>

  <h2>Liste avec ouverture/fermeture en CSS</h2>
  <ul>
  <li class="dynCSS">item1
      <ul>
      <li>sous-item 1.1</li>
      <li>sous-item 1.2</li>
      <li>sous-item 1.3</li>
      </ul>
  </li>
  <li class="dynCSS">item2
      <ul>
      <li>sous-item 2.1</li>
      <li>sous-item 2.2</li>
      </ul>
  </li>
  <li class="dynCSS">item 3</li>
  </ul>
```

affichage des noeuds

```
li.dynCSS:hover {color: red}
li.dynCSS:hover ul{display: block}
li.dynCSS:hover ul li{color: green}

/*pour la partie javascript*/
li {color: blue}
li ul {display: none}
li ul li {color: green}
```

Listes sélectives

```
function next (name, node) {
    for (let r = node; r != null; r = r.nextSibling)
      if (r.nodeName == name)
          return r;
    return null;
}

function toggle (e) {
    let item = e.target;
    let ulNode = next ("UL", item);
    if (item.nextSibling.style.display == "inline") {
        if (ulNode != null)
            ulNode.style.display = "block";
        item.nextSibling.style.display = "none";
    } else {
        if (ulNode != null)
            ulNode.style.display = "none";
        item.nextSibling.style.display = "inline";
    }
}

window.addEventListener ("load", function() {
    let ts = document.getElementsByClassName ("toggle");
    for (let i = 0; i < ts.length; ++i)
        ts[i].addEventListener ("click", toggle, false);
    },false)
```

```
<h2>Liste avec ouverture/fermeture en javascript</h2>
<ul>
<li><span class="toggle">item1</span><span style="display:inline">...</span>
    <ul>
    <li>sous-item 1.1</li>
    <li>sous-item 1.2</li>
    <li>sous-item 1.3</li>
    </ul>
</li>
<li><span  class="toggle">item2</span><span style="display:inline">...</span>
    <ul>
    <li><a href="#test2.1">sous-item 2.1</a></li>
    <li><a href="#test2.2">sous-item 2.2</a></li>
    </ul>
</li>
<li><span class="toggle">item3</span><span style="display:inline">...</span></li>
</ul>

</body>
</html>
```

Prochain cours

- un bon tutoriel JavaScript: `http://www.programiz.com/javascript`
- un autre tutoriel JavaScript: `http://www.w3schools.com/js`
- exceptions, modules, prototypes en JavaScript
- quelques bibliothèques JavaScript
- essais de pages web sexy !