CSI 3540

Structures, techniques et normes du Web

JavaScript, documents XHTML, CSS 2 et DOM

Objectif:

- Bien comprendre le modèle objet de document (DOM HTML) ; l'API permettant à JavaScript (et aux autres langages) de manipuler la page courante de l'agent utilisateur

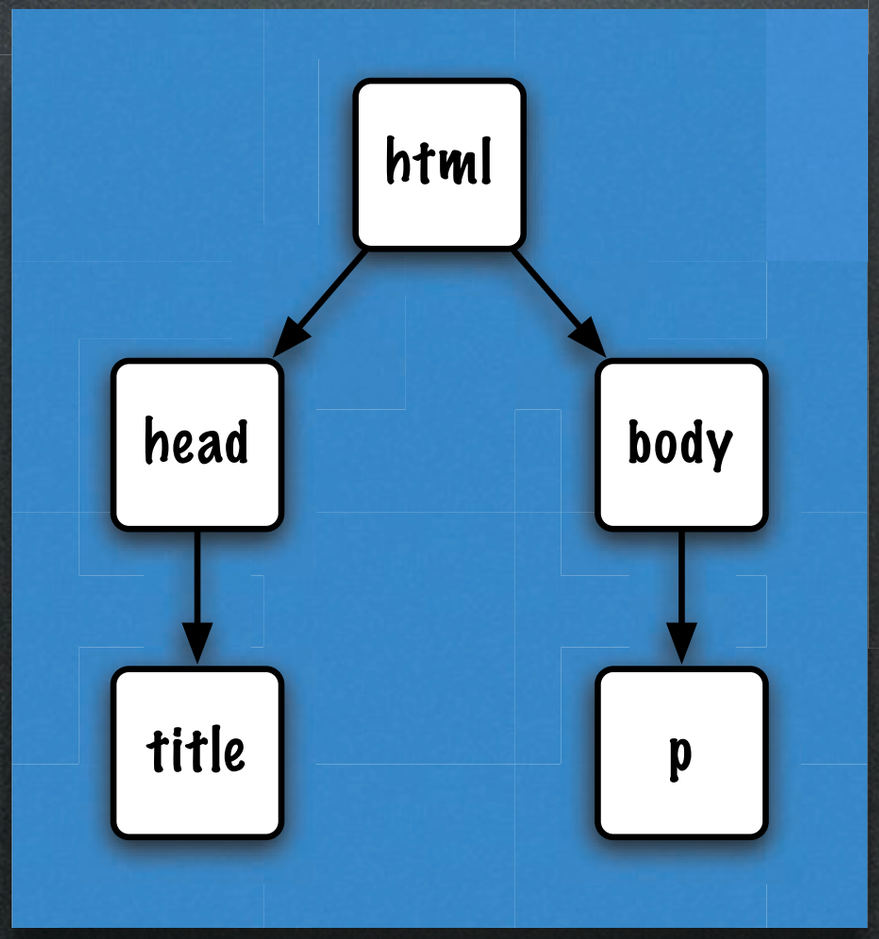
Lectures:

- Web Technologies (2007) § 5 pages 268-306

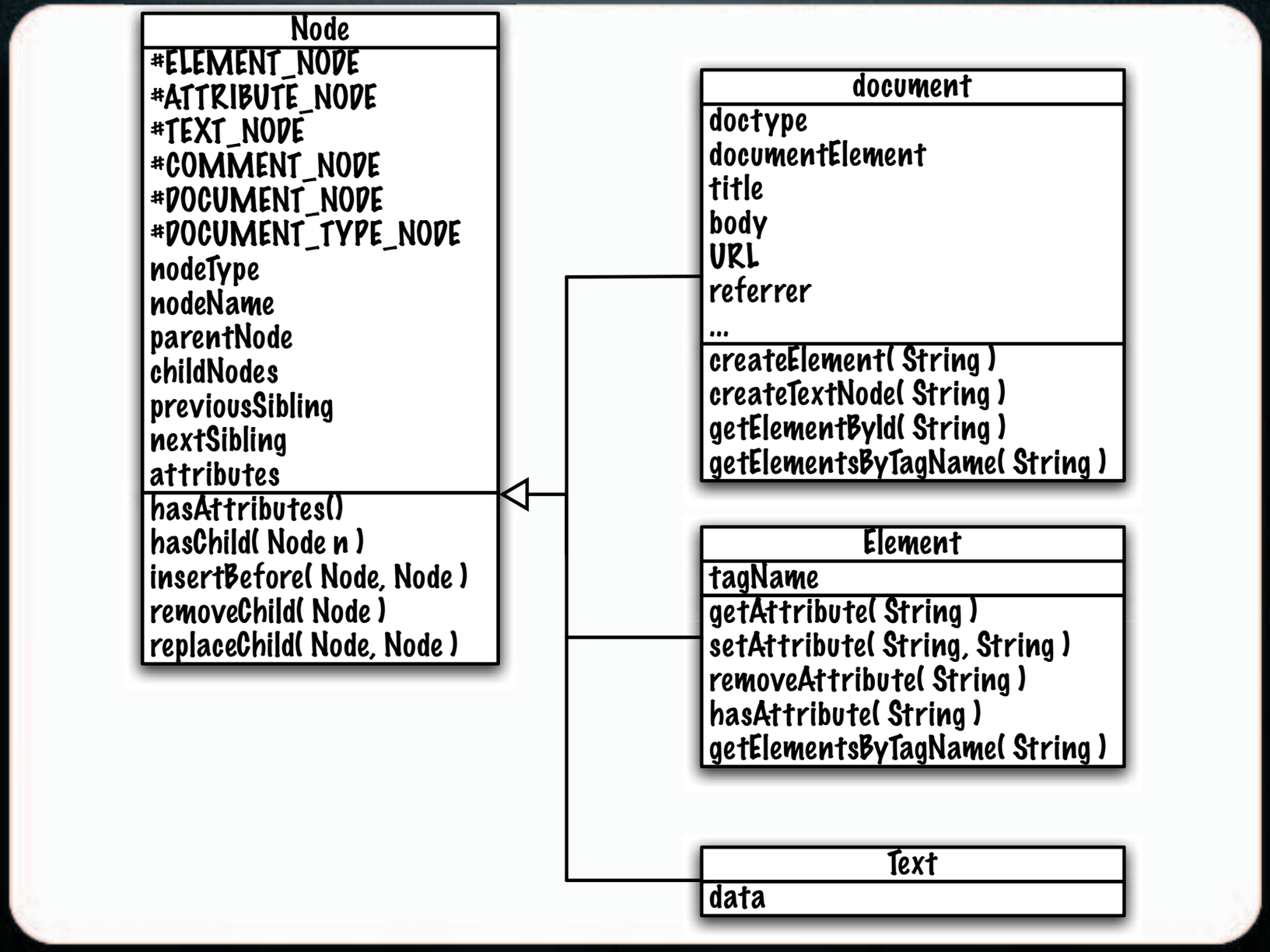
DOM - Qu'est-ce que c'est?

- Laissons parler les auteurs de la spécification :
  - "(...) une **interface indépendante de la plateforme et du langage** qui permet aux programmes et aux scripts l'**accès et la mise à jour dynamique du contenu et de la structure des documents**."

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="fr-CA" >
    <head>
      <title>
          Mon premier document
      </title>
    </head>
    <body>
      <p>
          Hello world!
      </p>
    </body>
</html>
```



Node
#ELEMENT_NODE #ATTRIBUTE_NODE #TEXT_NODE #COMMENT_NODE #DOCUMENT_NODE #DOCUMENT_TYPE_NODE nodeType nodeName parentNode childNodes previousSibling nextSibling attributes
hasAttributes() hasChild(Node n) insertBefore(Node newNode, Node aNode) removeChild(Node aNode) replaceChild(Node newNode, Node aNode)



Principaux objets

The WebKit Open Source Project

http://webkit.org/

webkit android

bio :: start | Grand dictionnaire termi… | Document Object Model … | The WebKit Open Source…

The WebKit Open Source Project

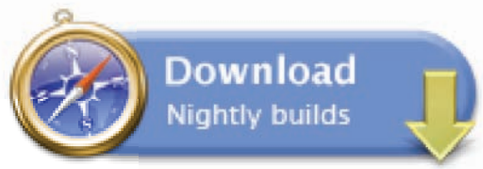
Welcome to the website for the WebKit Open Source Project!

WebKit is an open source web browser engine. WebKit is also the name of the Mac OS X system framework version of the engine that's used by Safari, Dashboard, Mail, and many other OS X applications. WebKit's HTML and JavaScript code began as a branch of the KHTML and KJS libraries from KDE. This website is also the home of S60's S60 WebKit development.

Getting involved

There are many ways to get involved. You can:

- download the latest nightly build
- install developer tools and then check out and build the source code



Once you have either of these, you can help by:

- reporting bugs you find in the software
- providing reductions to bugs
- submitting patches for review

More info

More information about WebKit can be found on its wiki. You can help here too, by adding information that can help others learn about WebKit. If you have more questions, contact us.

Webkit

- Engin pour **XML**, **XHTML**, **CSS**, **XSLT**, **JavaScript** et **DOM**
- Code source libre
- Semble suivre les standards W3C

Webkit

- Rejeton des bibliothèques de programmes de KDE (Système de fenêtrage pour Linux)
- Apple l'utilise pour **Safari**, **Dashboard**, **Mail**, mais aussi l'**iPhone** et l'**iPod Touch**
- Navigateur **S60** de Nokia
- **Android** (le nouvel OS de Google pour les téléphones portables)

Plan

1. **DOM 2 EVENT**
    1. Objet **Event**
    2. **stopPropagation()**
    3. **preventDefault()**

DOM 2 EVENT

Syntaxiquement différent sous IE6

- Les **attributs** des éléments **XHTML** associés aux événements intrinsèques (**onmouseover**) sont l'un des mécanismes pour assigner des gestionnaires d'événements aux éléments

- **La spécification du modèle objet de document (DOM) niveau 2 Events** [http://www.yoyodesign.org/doc/w3c/dom2-events/Overview.html]

- **Avantages** :
  - Découplage entre le document et de la gestion des événements (spécifiquement attributs événements intrinsèques vs addEventListner)
  - Multiples gestionnaires pour un même élément
  - L’ajout de nouveaux types d’événements nécessite l’ajout de nouveaux attributs au vocabulaire

```
document.implementation.hasFeature( "XML", "1.0" );
document.implementation.hasFeature( "HTML", "1.0" )
document.implementation.hasFeature( "Core", "2.0" )
document.implementation.hasFeature( "XML", "2.0" )
document.implementation.hasFeature( "HTML", "2.0" )
document.implementation.hasFeature( "XHTML", "2.0" )
document.implementation.hasFeature( "Views", "2.0" )
document.implementation.hasFeature( "StyleSheets", "2.0" )
document.implementation.hasFeature( "CSS", "2.0" )
document.implementation.hasFeature( "CSS2", "2.0" )
document.implementation.hasFeature( "Events", "2.0" )
document.implementation.hasFeature( "UIEvents", "2.0" )
document.implementation.hasFeature( "MouseEvents", "2.0" )
document.implementation.hasFeature( "MutationEvents", "2.0" )
document.implementation.hasFeature( "HTMLEvents", "2.0" )
document.implementation.hasFeature( "Range", "2.0" )
document.implementation.hasFeature( "Traversal", "2.0" )
document.implementation.hasFeature( "Core", "3.0" )
document.implementation.hasFeature( "XML", "3.0" )
document.implementation.hasFeature( "Events", "3.0" )
document.implementation.hasFeature( "UIEvents", "3.0" )
document.implementation.hasFeature( "MouseEvents", "3.0" )
document.implementation.hasFeature( "TextEvents", "3.0" )
document.implementation.hasFeature( "KeyboardEvents", "3.0" )
document.implementation.hasFeature( "MutationEvents", "3.0" )
document.implementation.hasFeature( "MutationNameEvents", "3.0" )
document.implementation.hasFeature( "HTMLEvents", "3.0" )
document.implementation.hasFeature( "LS", "3.0" )
document.implementation.hasFeature( "LS-Async", "3.0" )
document.implementation.hasFeature( "Validation", "3.0" )
document.implementation.hasFeature( "XPath", "3.0" )
document.implementation.hasFeature( "HTML", "5.0" )
document.implementation.hasFeature( "XHTML", "5.0" )
```

JavaScript

XML 1.0 : true
HTML 1.0 : true
Core 2.0 : true
XML 2.0 : true
HTML 2.0 : true
XHTML 2.0 : true
Views 2.0 : true
StyleSheets 2.0 : true
CSS 2.0 : true
CSS2 2.0 : true
Events 2.0 : true
UIEvents 2.0 : true
MouseEvents 2.0 : true
MutationEvents 2.0 : true
HTMLEvents 2.0 : true
Range 2.0 : true
Traversal 2.0 : true
Core 3.0 : false
XML 3.0 : false
Events 3.0 : false
UIEvents 3.0 : false
MouseEvents 3.0 : false
TextEvents 3.0 : true
KeyboardEvents 3.0 : false
MutationEvents 3.0 : false
MutationNameEvents 3.0 : false
HTMLEvents 3.0 : false
LS 3.0 : false
LS-Async 3.0 : false
Validation 3.0 : false
XPath 3.0 : true
org.w3c.dom.svg 1.0 : false
org.w3c.dom.svg 1.1 : false
WebForms 2.0 : false
HTML 5.0 : false
XHTML 5.0 : false

OK

Contexte

Objet hôte **Event**

- Lorsqu'un événement survient, un objet hôte **Event** est créé, il contient entre autres:
  - **type** d'événement : click, mouseover, ...
  - **target** : référence vers le noeud de l'arbre du document ayant généré cet événement
- L'objet est envoyé aux gestionnaires (listeners)

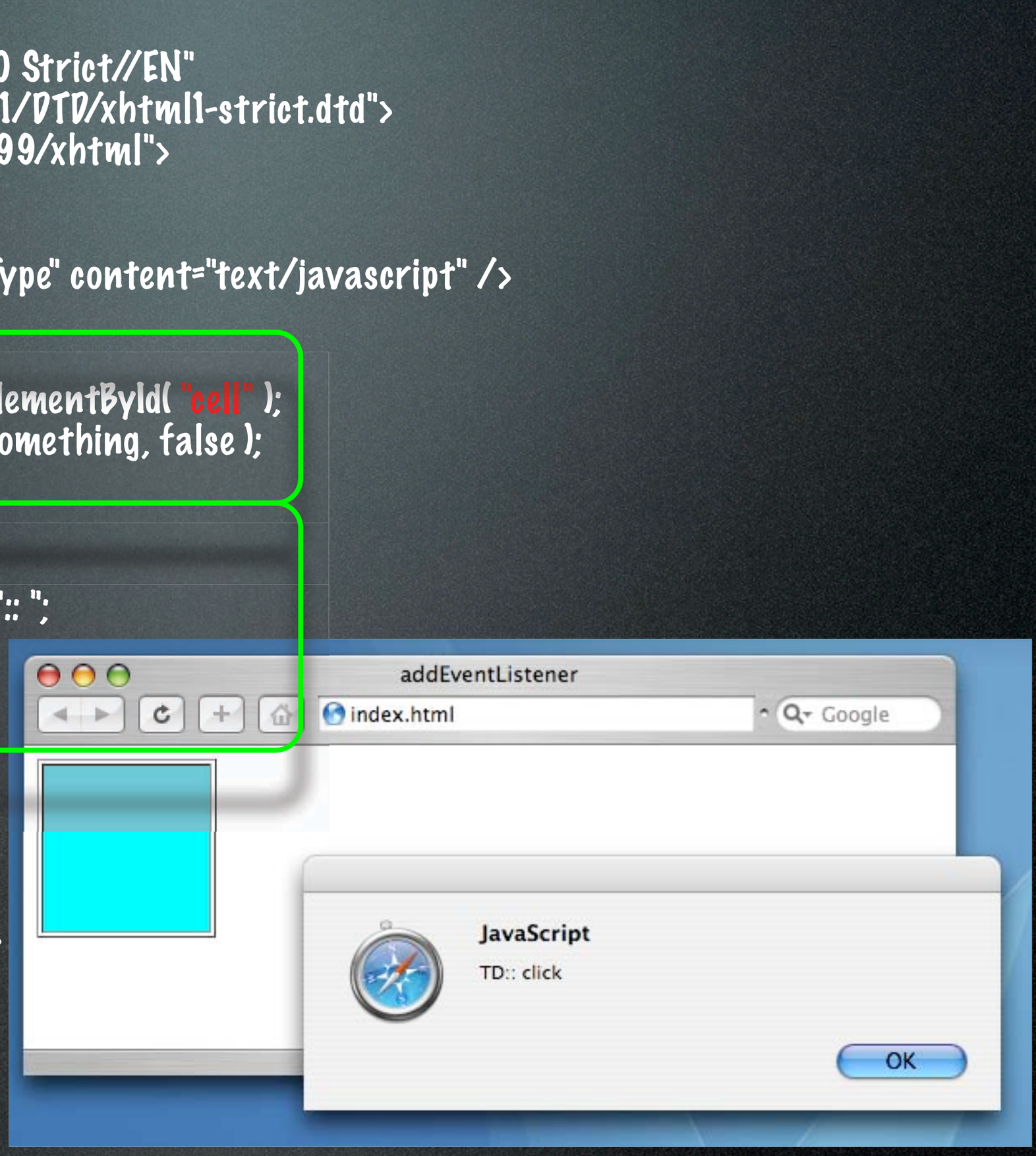
```
// Introduite dans DOM niveau 2 :
interface Event {

  // PhaseType
  const unsigned short      CAPTURING_PHASE   = 1;
  const unsigned short      AT_TARGET         = 2;
  const unsigned short      BUBBLING_PHASE    = 3;

  readonly attribute DOMString       type;
  readonly attribute EventTarget     target;
  readonly attribute EventTarget     currentTarget;
  readonly attribute unsigned short  eventPhase;
  readonly attribute boolean         bubbles;
  readonly attribute boolean         cancelable;
  readonly attribute DOMTimeStamp timeStamp;
  void stopPropagation();
  void preventDefault();
  void initEvent( in DOMString eventTypeArg,
                  in boolean canBubbleArg,
                  in boolean cancelableArg);
};
```

Gestionnaire d'événements

- Un gestionnaire est une fonction ayant un paramètre, une référence vers un événement (**Event**)
- **addEventListener(type, fun, capture)**

type: click, mouseover, ...
fun: le gestionnaire
capture: un boolean, lié au contrôle de la propagation des événements

```
<!DOCTYPE html
      PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>addEventListener</title>
  <meta http-equiv="Content-Script-Type" content="text/javascript" />
  <script type="text/javascript">
   function init() {
      var elt = window.document.getElementById( "cell" );
      elt.addEventListener( "click", doSomething, false );
   }
   function doSomething( event ) {
      var msg;
      msg = event.target.nodeName + ":: ";
      msg = msg + event.type;
      alert( msg );
   }
   window.onload = init();
  </script>
 </head>
 <body>
    <table cellpadding="50" border="1">
     <tr>
      <td id="cell" bgcolor="aqua"></td>
     </tr>
    </table>
 </body>
</html>
```



Remarques

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>addEventListener</title>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
    <script type="text/javascript">
      function init() {
          var elt = window.document.getElementById( "cell" );
          elt.addEventListener( "click", doSomething, false );
      }
      function doSomething( event ) {
          var msg;
          msg = event.target.nodeName + ":: ";
          msg = msg + event.type;
          alert( msg );
      }
      window.onload = init();
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

Sous Safari, à tout le moins, le corps du script est exécuté avant que le corps de la page n'ait été lu.

window.onload ...

vs

```
var elt = window.document.getElementById( "cell" );
elt.addEventListener( "click", doSomething, false );

function doSomething( event ) {
    var msg;
    msg = event.target.nodeName + ":: ";
    msg = msg + event.type;
    alert( msg );
}
```

Attributs événements intrinsèques et événements DOM 2

- load, unload (élément body)
- click, ~~**dblclick**~~, mousedown, mouseup, mouseover, mousemove, mouseout
- focus, blur (éléments : a, label, input, select, textarea, button)
- ~~**keypress**~~, ~~**keydown**~~, ~~**keyup**~~
- submit, reset
- select, change

Événements associés à la fenêtre

- Modules : **HTMLEvents**, **UIEvents**, **MutationEvents**
- Autres événements sont principalements associés avec la fenêtre (objet hôte **window**)

Événements associés à la fenêtre

- **error** : erreur, par exemple lors de la lecture d'une image, de l'exécution d'un script ...
- **resize** : l'aspect de la fenêtre a changé
- **scroll** : mouvement du curseur flottant
- Donnez des exemples d'applica

Afin de s'assurez que l'usager a lu toute la license. Le bouton est initialement inactif, puis lorsque l'usager déplace le curseur flottant jusqu'au bas du document, le bouton devient actif.

DOM 2 : Mouse EVENT

- Il y 6 événements liés à la souris : click, mousedown, mouseup, **mousemove**, mouseout

- Puisque les événements **intrinsèques de XHTML** existent, pourquoi ajouter des **événements DOM 2** ?

- Le gestionnaire, à l'aide de l'objet **Event**, obtient des informations supplémentaires

Propriété(s)	Description
clientX, clientY	Position dans la zone client de l'agent utilisateur
screenX, screenY	Position par rapport à la totalité de l'écran
altKey, ctrlKey, metaKey, shiftKey	Booléen, vrai si la clé correspondante était enfoncée lorsque l'événement fut généré
button	Le bouton enfoncé (gauche...)
detail	Nombre des fois ou le bouton a été enfoncé au même endroit
relatedTarget	Élément duquel on arrive ou dans lequel on entre

Objet **Event**

Faire glisser une image

index.html

```
<html>
 <head>
  <title>CSI 3540</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <link rel="stylesheet" type="text/css" href="index.css" title="Default" />
  <script type="text/javascript" src="index.js"></script>
 </head>
 <body onload="init()">
  <p>
   <h1>Gestion d'événements</h1>
  </p>
  <div id="outer">
   <div id="inner">
   <img src="bobs-1200x800.jpg" alt="Cedergren Laboratory"/>
   </div>
  </div>
 </body>
</html>
```

index.css

```
h1 {
    font: 20pt sans-serif;
}

#outer {
    height: 300px;
    width:  400px;
    border:  1px solid black;
    position: relative;
    overflow: hidden;
}

#inner {
    position: relative;
    left: 0px;
    top: 0px;
}
```

index.js

```
var dragging = false;
var top;
var left;
var dragStartTop;
var dragStartLeft;

function init() {

    // Ajout des gestionnaires d'événement

    var outer = document.getElementById( "outer" );

    outer.onmousedown = startMove;
    outer.onmousemove = processMove;
    outer.onmouseup = stopMove;

}
```

index.js

```
function startMove( event ) {

    var inner = document.getElementById( "inner" );
    inner.style.cursor = 'hand';

    dragStartLeft = event.clientX;
    dragStartTop = event.clientY;

    top = stripPx( inner.style.top );
    left = stripPx( inner.style.left );

    dragging = true;

    return false;
}
```

index.js

```
function processMove( event ) {

    var inner = document.getElementById("inner");
    if ( dragging ) {
        inner.style.top = top + ( event.clientY - dragStartTop );
        inner.style.left = left + ( event.clientX - dragStartLeft );
    }
}
```

index.js

```
function stopMove() {
    var inner = document.getElementById( "inner" );
    inner.style.cursor = "default";
    dragging = false;
}
```

index.js

```
// value a la forme 74px, la fonction retire le suffixe px

function stripPx( value ) {
    if ( value == "" ) return 0;
    return parseFloat( value.substring( 0, value.length - 2 ) );
}

// Adapté de J. Gehtland, B. Galbraith et D. Almaer (2006) Pragmatic Ajax:
//    A Web 2.0 Primer. The Pragmatic Programmers.
```

Objet **Event**

Faire glisser une image

Remarques

- Le nom de la méthode est **addEventListener** et non pas **setEventListener**, ainsi on peux associer plusieurs gestionnaires à un même élément!

Propagation

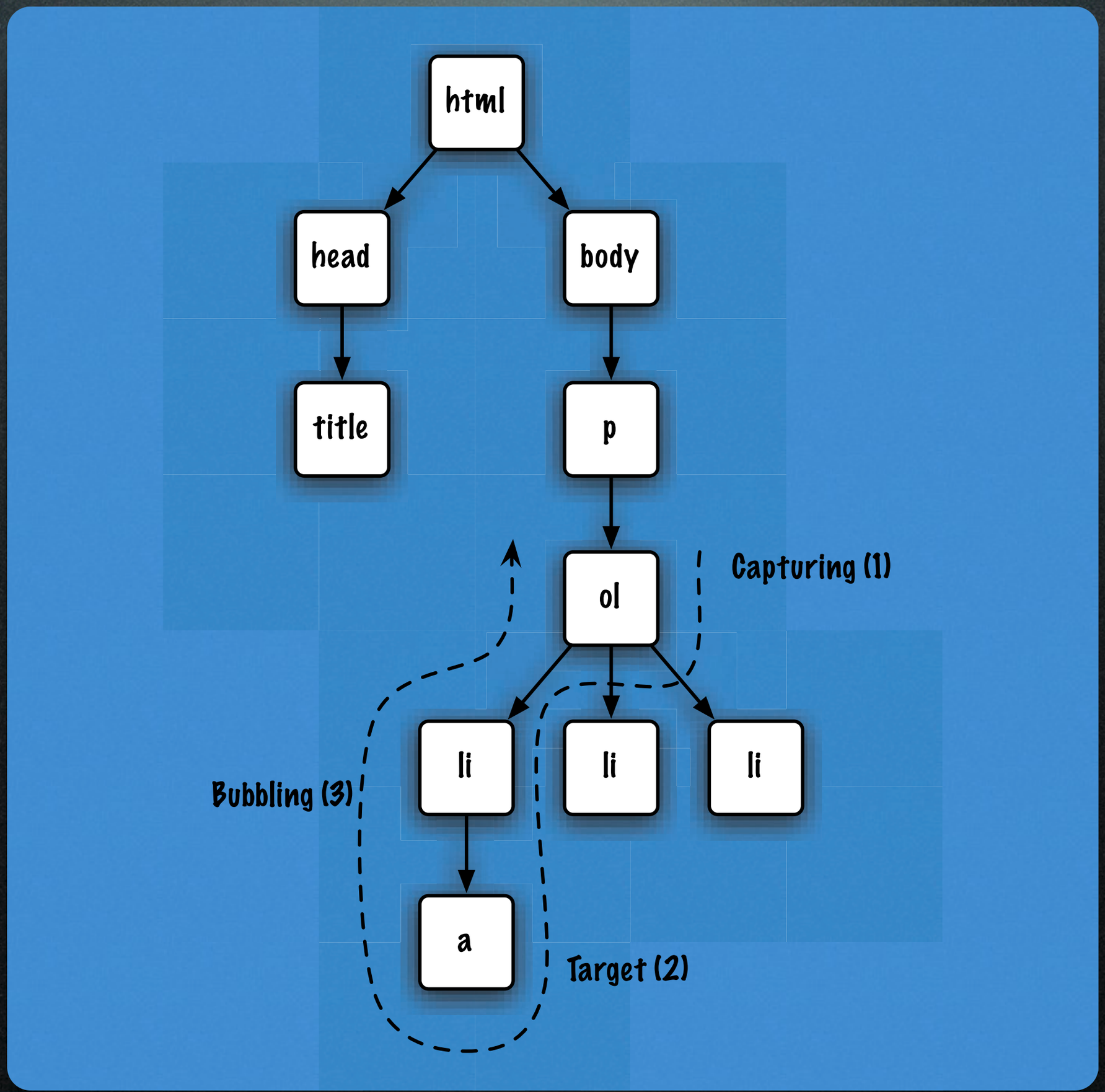
Propagation des événements

- **Objectif** : offrir des mécanismes pour la gestion des flux d'événements à travers une structure d'arbre

Propagation des événements

- Il y a **3 types de gestionnaires** : **capturing event listener**, **target listener** et **bubbling listener** (bouillonnement)

```
<script type="text/javascript">
  function init() {
      var elt;

      elt = window.document.getElementById( "elt1.1" );
      elt.addEventListener( "click", fCapturing, true );
      elt.addEventListener( "click", fBubbling, false );

      elt = window.document.getElementById( "elt1.1.1" );
      elt.addEventListener( "click", fCapturing, true );
      elt.addEventListener( "click", fBubbling, false );

      elt = window.document.getElementById( "elt1.1.1.1" );
      elt.addEventListener( "click", fTarget, false );
  }
  function fCapturing ( event ) {
      event.stopPropagation();
      alert( "Caputuring event listener:: current: " + event.currentTarget.id );
  }
  function fTarget ( event ) {
      alert( "Target event listener:: target: " + event.target.id );
  }
  function fBubbling ( event ) {
      alert( "Bubbling event listener:: current: " + event.currentTarget.id );
  }

  window.onload = init();
</script>
```


html
head
body
title
p
ol
Capturing (1)
li
li
li
Bubbling (3)
a
Target (2)

Propagation

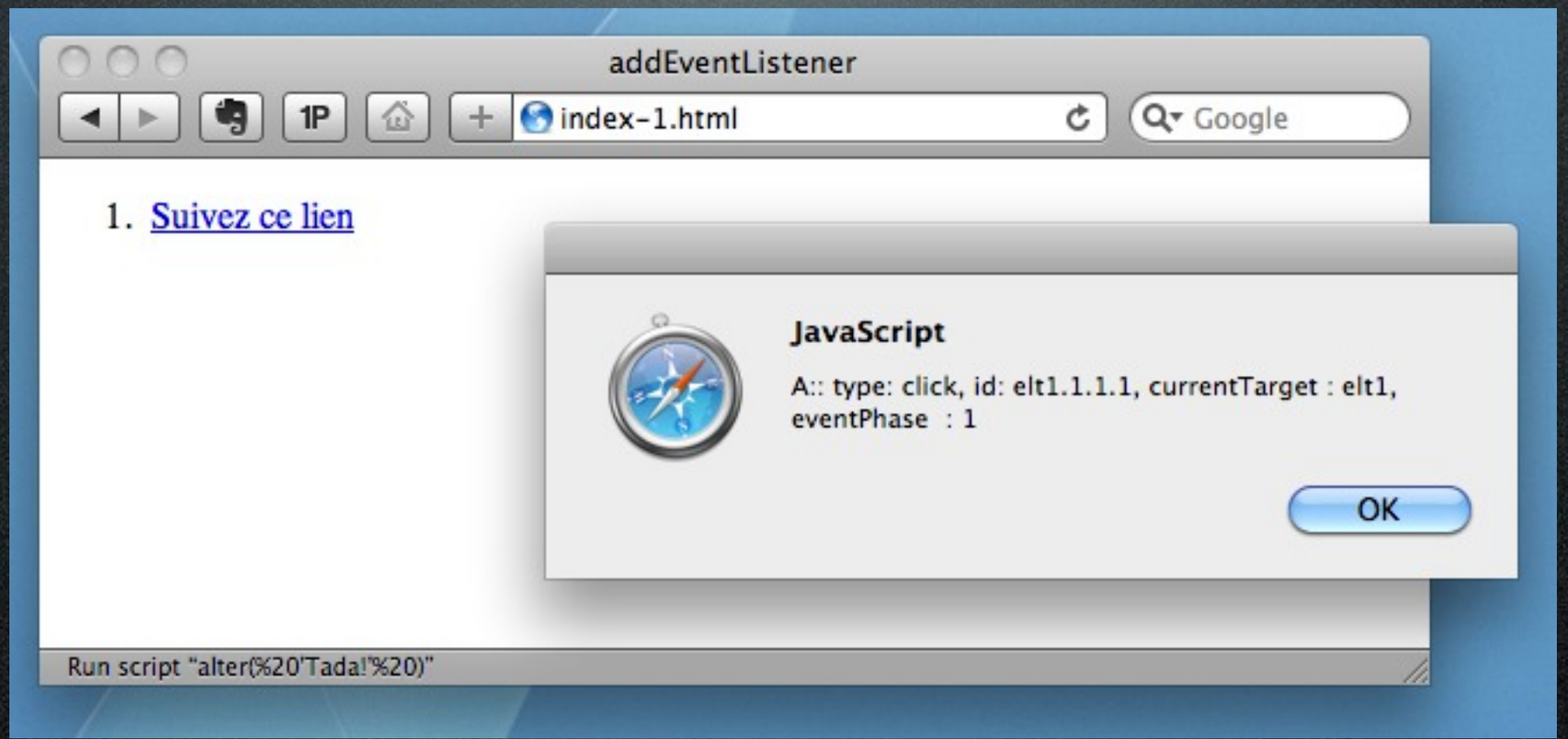
- La chaîne de gestionnaires est déterminée avant le déclenchement des gestionnaires
- Ainsi, si la structure de l'arbre change, l'ordre des appels de gestionnaires demeure inchangé

Propagation des événements

- capturing : l'élément est un ancêtre, le booléen **capture** est vrai
- target : l'élément est la cible, le booléen **capture** est faux
- bubbling : l'élément est un ancêtre, le booléen **capture** est faux
- Les gestionnaires sont appelés dans cet ordre

```
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>addEventListener</title>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
  </head>
  <body>
    <div id="elt1">
      <ol id="elt1.1">
       <li id="elt1.1.1">
         <a id="elt1.1.1.1" href="javascript:alert( 'Tada!' )">
            Suivez ce lien
         </a>
       </li>
      </ol>
    </div>
  </body>
</html>
```

```
function init() {
    var elt = window.document.getElementById( "elt1" );
    elt.addEventListener( "click", f, true );
    elt.addEventListener( "click", f, false );
    var elt = window.document.getElementById( "elt1.1" );
    elt.addEventListener( "click", f, true );
    elt.addEventListener( "click", f, false );
    var elt = window.document.getElementById( "elt1.1.1" );
    elt.addEventListener( "click", f, true );
    elt.addEventListener( "click", f, false );
    var elt = window.document.getElementById( "elt1.1.1.1" );
    elt.addEventListener( "click", f, true );
    elt.addEventListener( "click", f, false );
}
function f ( event ) {
    var msg;
    msg = event.target.nodeName + ":: type: ";
    msg = msg + event.type + ", id: ";
    msg = msg + event.target.id;
    msg = msg + ", currentTarget : " + event.currentTarget.id;
    msg = msg + ", eventPhase  : " + event.eventPhase;
    alert( msg );
}
```


addEventListener
index-1.html
Google
1. Suivez ce lien
JavaScript
A:: type: click, id: elt1.1.1.1, currentTarget : elt1, eventPhase : 1
OK
Run script "alter(%20'Tada!'%20)"

Propriétés de **Event** (suite)

- La propriété **eventPhase** a pour valeur **1** si capturing, **2** si target et **3** si bubbling

- **currentTarget** est une référence vers l'élément auquel le gestionnaire a été associé

- La méthode **stopPropagation()** élimine tout autres appels aux gestionnaires pour cet événement

```
<script type="text/javascript">
  function init() {
      var elt;

      elt = window.document.getElementById( "elt1.1" );
      elt.addEventListener( "click", fCapturing, true );
      elt.addEventListener( "click", fBubbling, false );

      elt = window.document.getElementById( "elt1.1.1" );
      elt.addEventListener( "click", fCapturing, true );
      elt.addEventListener( "click", fBubbling, false );

      elt = window.document.getElementById( "elt1.1.1.1" );
      elt.addEventListener( "click", fTarget, false );
  }
  function fCapturing ( event ) {
      event.stopPropagation();
      alert( "Caputuring event listener:: current: " + event.currentTarget.id );
  }
  function fTarget ( event ) {
      alert( "Target event listener:: target: " + event.target.id );
  }
  function fBubbling ( event ) {
      alert( "Bubbling event listener:: current: " + event.currentTarget.id );
  }

  window.onload = init();
</script>
```

Propagation des événements

- bubbling : traitement par défaut
- capturing : traitement obligatoire ; s'il stoppe la propagation, le traitement par défaut n'aura pas lieu

Propagation des événements

- capturing : si **capturing** est **true** le gestionnaire sera exécuté durant la phase **capture**
- capturing : si **capturing** est **false** le gestionnaire sera exécuté durant les phases **target** et **bubbling**

Implémenter un menu à l'aide de JavaScript/DOM

- Composé d'un menu principal et de menus déroulants
- Lorsque la souris passe au dessus d'un élément du menu principale, le menu déroulant associé est affiché
- Lorsque la souris passe au dessus d'un élément d'un menu déroulant, ce dernier est mis en évidence

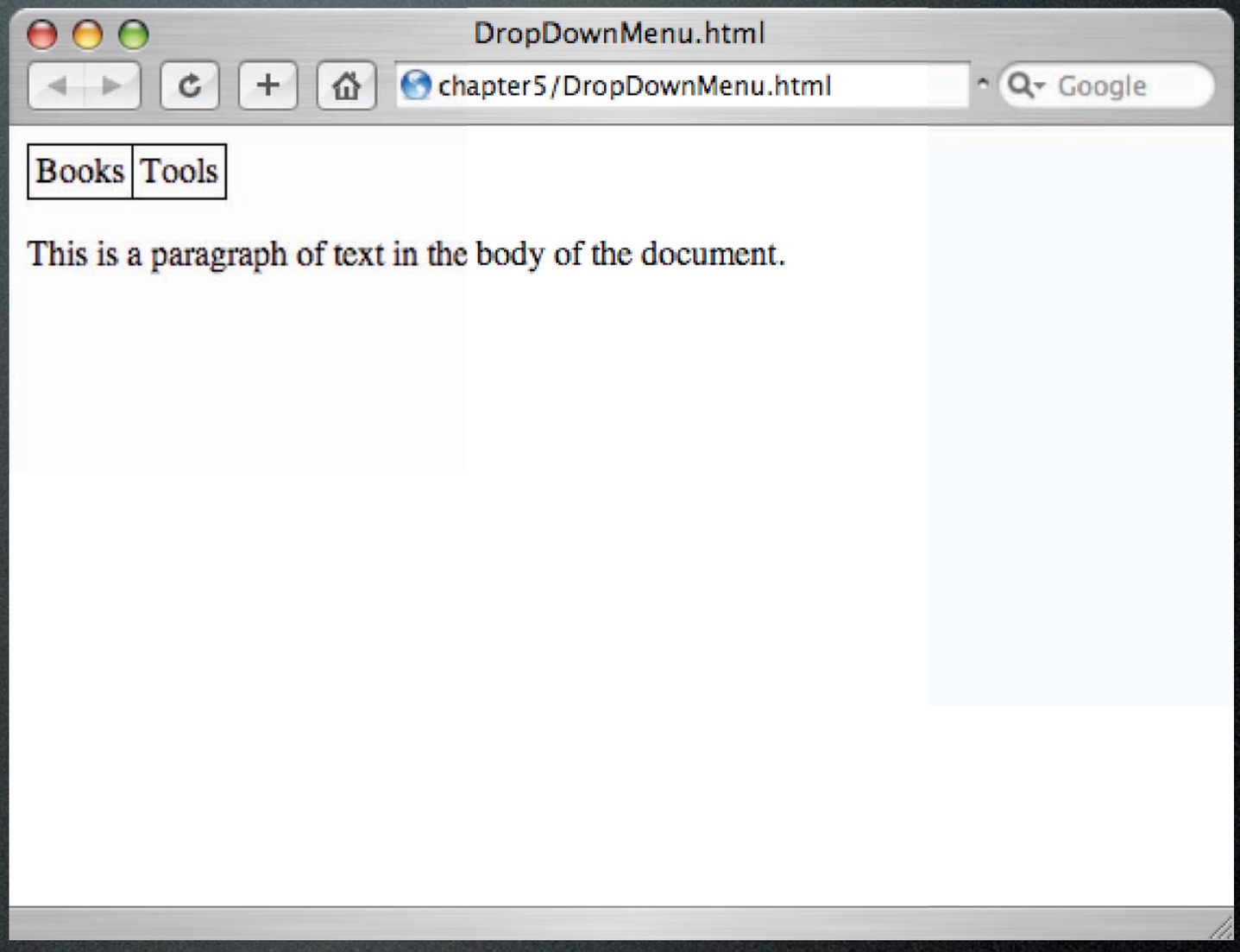
Implémenter un menu à l'aide de JavaScript/DOM

- Quelle est la contribution du modèle de propagation à la conception du menu déroulant ?

- Le menu disparaît lorsque le curseur ne survole ni l'entête, ni le menu déroulant (la gestion repose sur l'arborescence du document et non pas sur le rendu visuel)

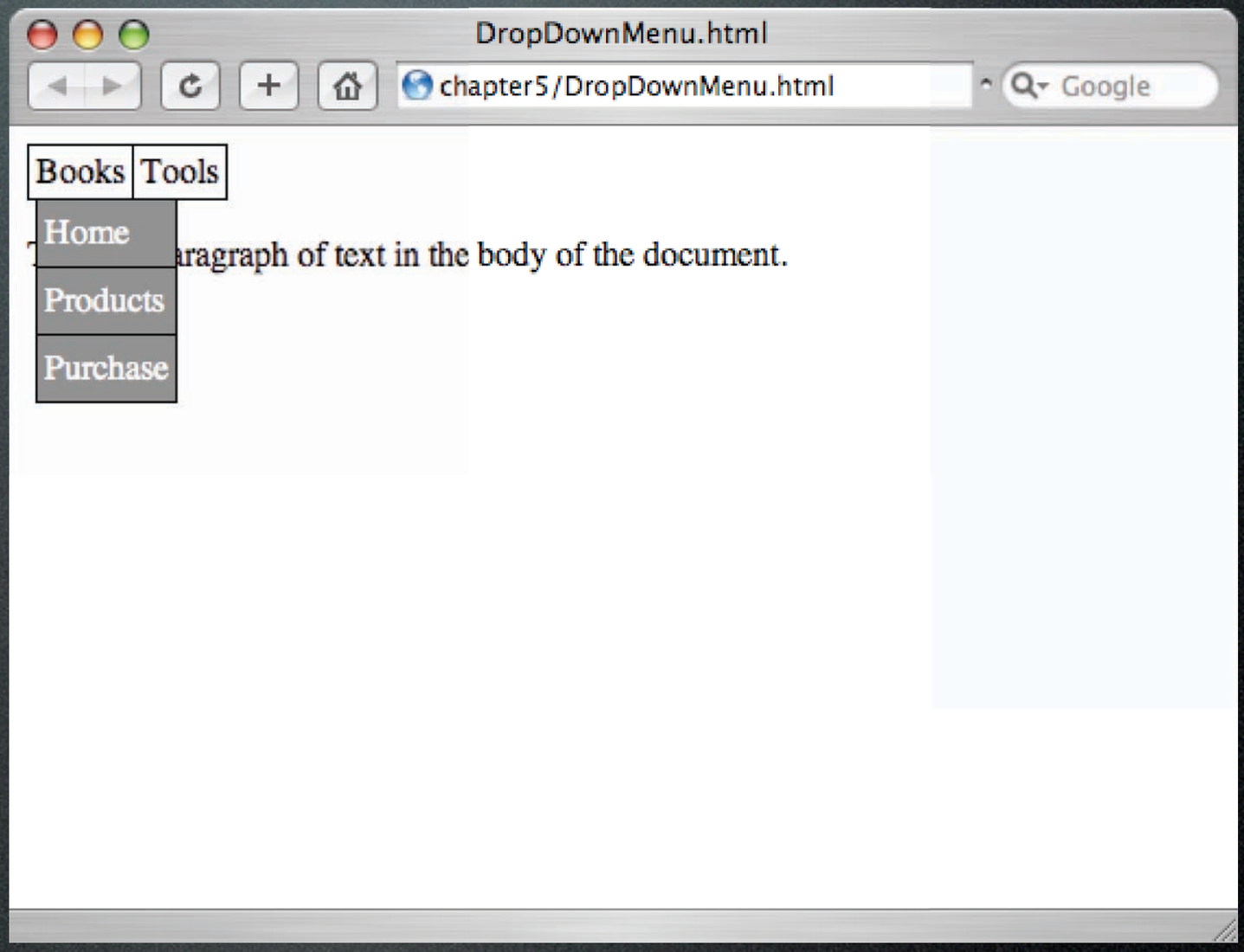
Menu déroulant

menu principal



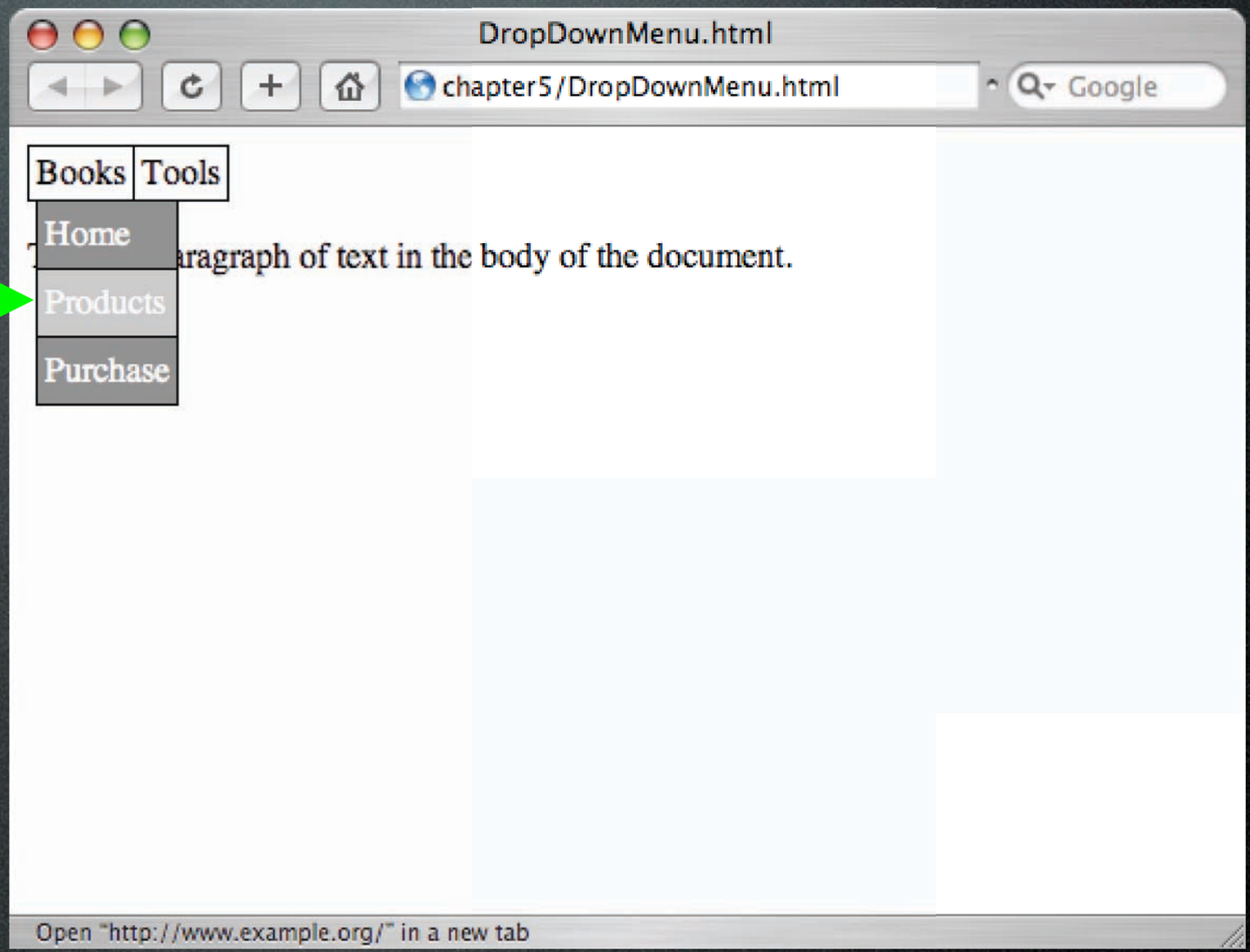
Menu déroulant

menu déroulant



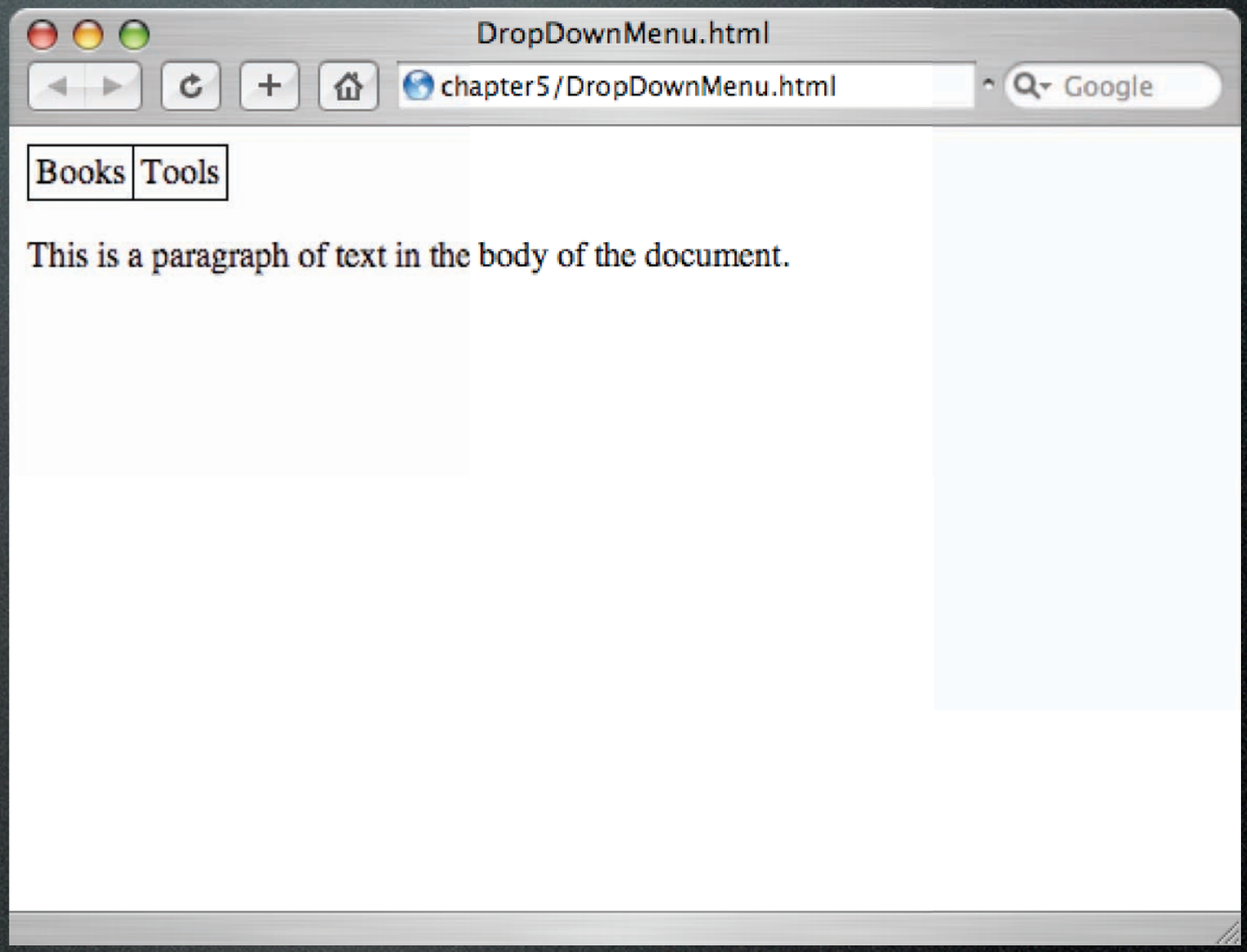
Menu déroulant

mise en évidence



Menu déroulant

Le menu déroulant disparaît lorsque la souris quitte le menu principal et le menu déroulant



```
<body onload="addEventHandlers();">
 <table cellpadding="0" cellspacing="0" class="menubar">
   <tbody>
    <tr>
     <td>
      <div id="MenuBar1">
        Books<div id="DropDown1">
        <table cellpadding="3" cellspacing="0" class="navbar">
          <tbody>
            <tr>
              <td id="DropDown1_1"><a
                 href="http://www.example.com"
                 >Home</a>
              </td>
            </tr>
            <tr>
              <td id="DropDown1_2"><a
                 href="http://www.example.org"
                 >Products</a>
              </td>
            </tr>
            ...
```

```
.menubar div { position:relative;
               line-h eight:1.5em;
               padding:0 0.5ex;
               margin:0
             }

.menubar div div { position:absolute;
                   top:1.5em; left:0;
                   z-index:1;
                   display:none
                 }
```

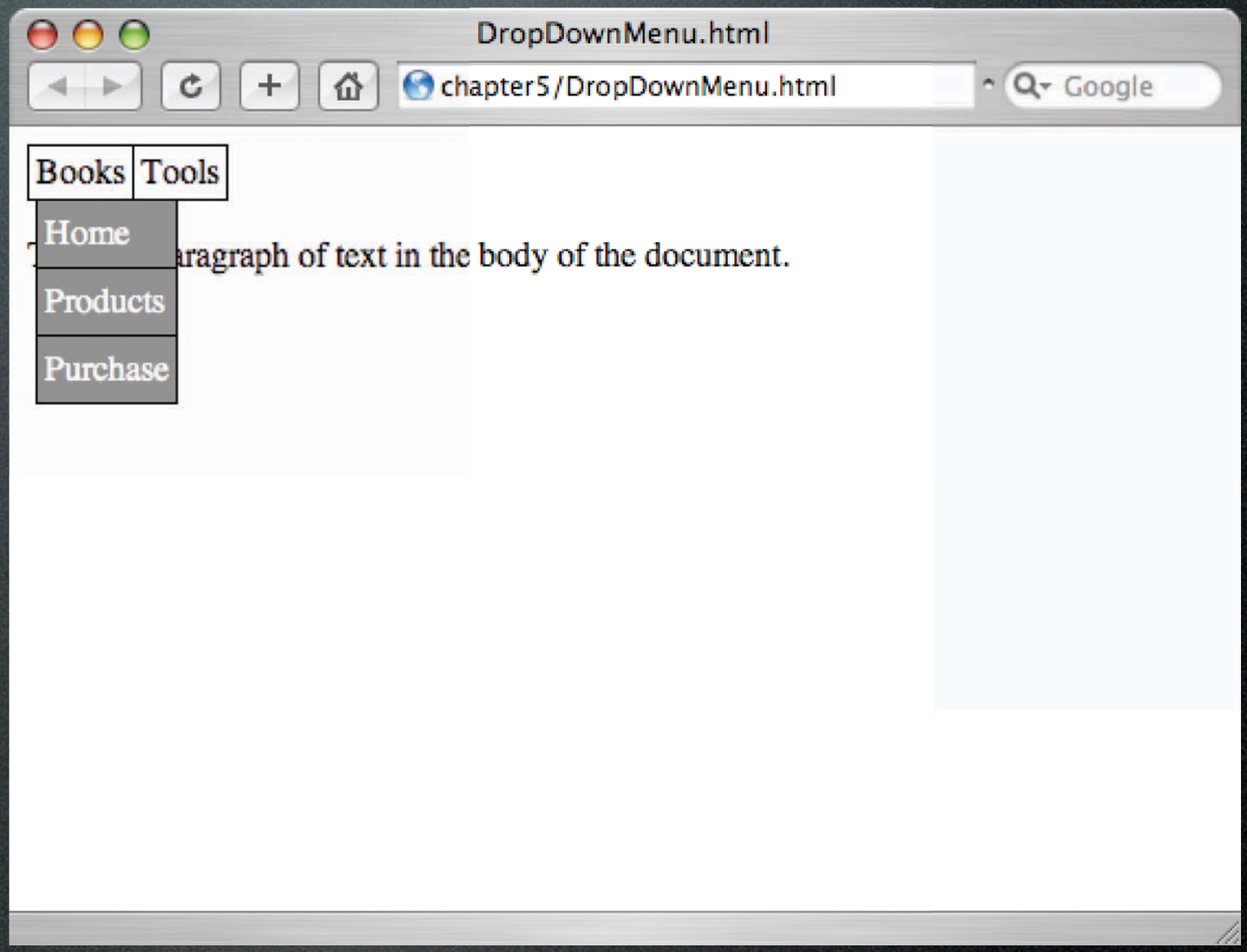
position:absolute, retire l'élément du flux de calcul normal

display:none, donc l'élément n'est pas visible

z-index:1, dessiné par dessus de tous les éléments

Menu déroulant

Évitez les espaces entre l'entête et le menu, sinon, un événement mouseout sera possiblement généré



Gestion des événements

```
var menuBar1  = window.document.getElementById("MenuBar1");
menuBar1.addEventListener("mouseover", showDropDown, false);
menuBar1.addEventListener("mouseout", hideDropDown, false);

menuBar1.dropDown = window.document.getElementById("DropDown1");

var dropDown1_1  = window.document.getElementById("DropDown1_1");
dropDown1_1.addEventListener("mouseover", highlight, false);
dropDown1_1.addEventListener("mouseout", lowlight, false);
var dropDown1_2 = window.document.getElementById("DropDown1_2");
dropDown1_2.addEventListener("mouseover", highlight, false);
dropDown1_2.addEventListener("mouseout", lowlight, false);
var dropDown1_3 = window.document.getElementById("DropDown1_3");
dropDown1_3.addEventListener("mouseover", highlight, false);
dropDown1_3.addEventListener("mouseout", lowlight, false);
```

Gestion des événements

```
function showDropDown(event) {

  if (event.target == event.currentTarget) {
   var dropDown = event.currentTarget.dropDown;
   dropDown.style.display = "block";
  }
  return;
}
```

- Appelé lors d'événements mouseover
- Ignore les événements «bubbling»
- Rend le menu visible (display = “block”)

Gestion des événements

```
function hideDropDown(event) {

  if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "none";
  }
  return;
}
```

- Appelé lors d'événements mouseout
- Ignore les événements si le curseur est toujours à l'intérieur du menu principale, du menu déroulant, ou un descendant
- Rend le menu invisible (display = "none")

Gestion des événements

```
function ancestorOf( ancestorElt, descendElt ) {
  var found;

  if ( ! descendElt ) {
    found = false;
  } else if ( descendElt == ancestorElt ) {
    found = true;
  } else {
    found = ancestorOf(ancestorElt, descendElt.parentNode);
  }
  return found;
}
```

Gestion des événements

```
function highlight(event) {

  if ( event.currentTarget.style.backgroundColor != "silver" ) {
    event.currentTarget.style.backgroundColor = "silver";
  }

  event.stopPropagation();
  return;
}
```

Gestion des événements

```
function lowlight(event) {
  if ( ! ancestorOf( event.currentTarget, event.relatedTarget ) ) {
    event.currentTarget.style.backgroundColor = "gray";
  }
  return;
}
```

Gestionnaires implicites

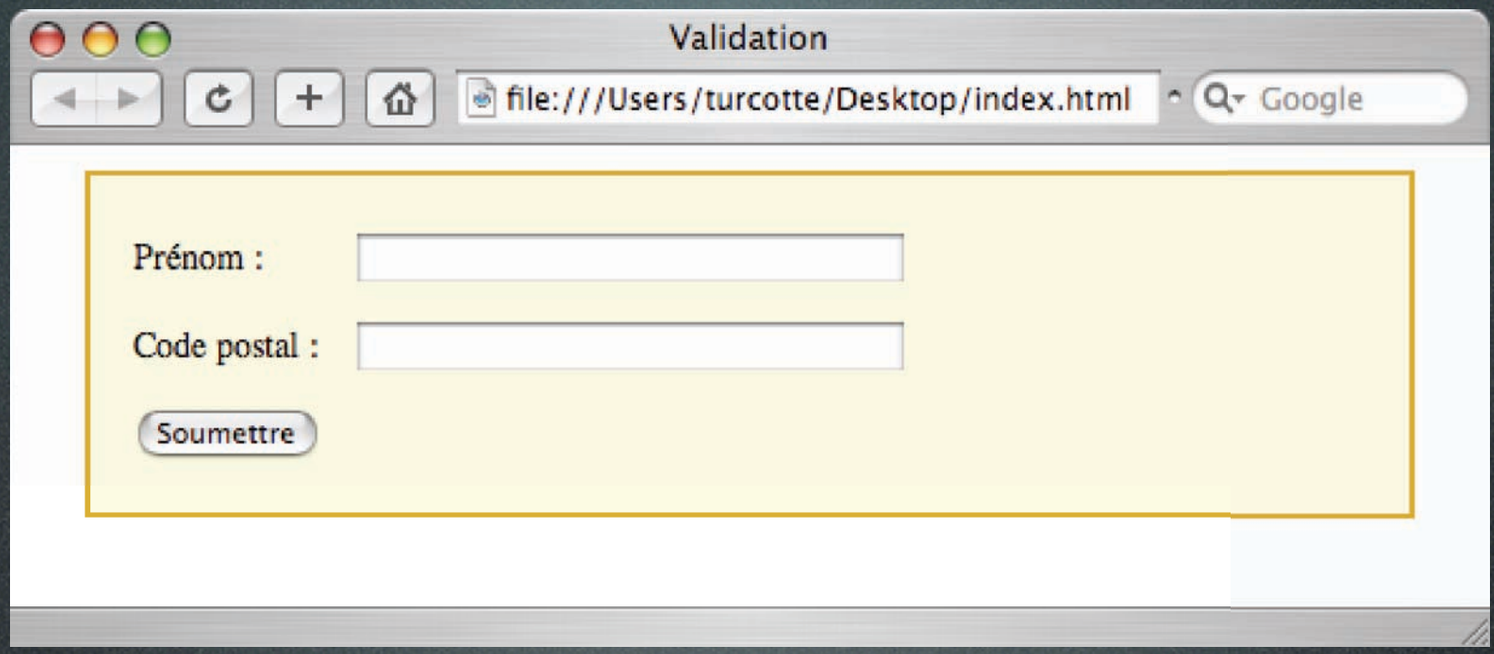
- En plus des mécanismes de gestion présentés jusqu'à maintenant
- Certains éléments ont un gestionnaire implicite qui est appelé à la suite de ceux-ci
- Par exemple, un gestionnaire pour les éléments **a** pour charger une nouvelle page, un gestionnaire pour l'élément **input**...

Gestionnaires implicites

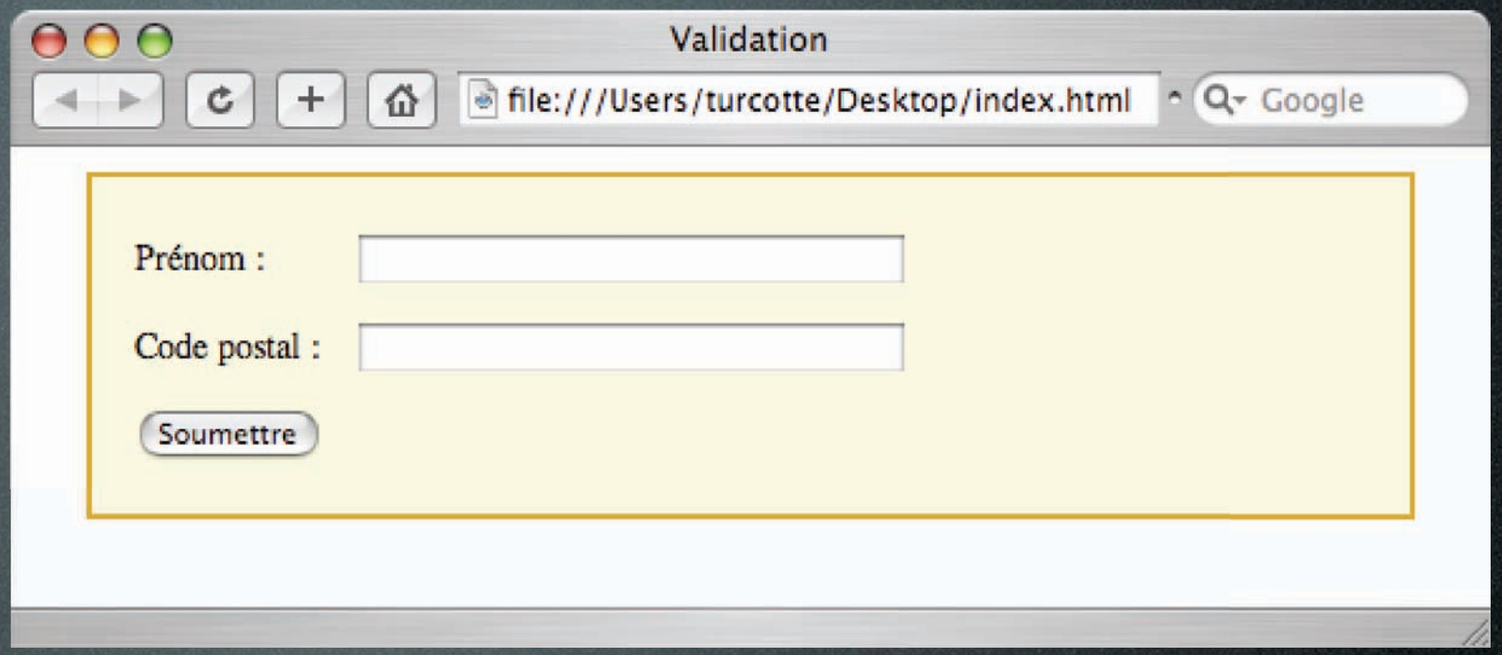
- Ils ne sont pas affectés par la méthode **stopPropagation()**
- Il existe cependant un mécanisme pour les mettre hors circuit, la méthode **preventDefault()** de l'objet **Event**

Exemple détaillé : validation de données

- La **validation des données** est un excellent exemple d'utilisation de **JavaScript** et du **DOM**
- Évite les aller-retour entre le serveur et le client
- Les traitements sont simples et ne dépendent pas de données se trouvant sur le serveur




```
<form action="http://localhost:8080/services/subscribe" method="post">
 <table border="0" cellpadding="5">
  <tr>
   <td><label for="prénom">Prénom :</label></td>
   <td><input type="text" size="30" name="prénom" /></td>
  </tr>
  <tr>
   <td><label for="codepostal">Code postal :</label></td>
   <td><input type="text" size="30" name="codepostal" /></td>
  </tr>
  <tr>
   <td colspan="2" align="left"><input type="submit" value="Soumettre" /></td>
  </tr>
 </table>
</form>
```



JavaScript :

```
function validerCodePostal( elem ) {
    var cp = elem.value;
    var valide = true;
    if ( /^\s*$/.test( cp ) ) {
        alert( "Veuillez entrer votre code postal" );
        valide = false;
    } else if ( ! /^[a-zA-Z]\d[a-zA-Z]\s*\d[a-zA-Z]\d$/.test( cp ) ) {
        alert( "Ce code postal n'est pas valide" );
        valide = false;
    }
    return valide;
}
```

XHTML :

```
<input onchange="validaterCodePostal( this );" ... />
```

Remarques

- Cette solution est insuffisante :
  1. Soumission sans code postale
  2. Soumission erronée

Si on utilise des événements intrinsèques comme attributs de l'élément INPUT/SUBMIT, l'agent utilisateur fera quand même la soumission des informations, après avoir soumis les messages d'alerte

Voir liste d'événements Section 5.64

Solution :

- Stopper le traitement des événements :

```
function init() {
   var formulaire = document.getElementById( "fsaisi" );
   formulaire.addEventListener( "submit", valider, false );
}

function valider( event ) {
   ...
   if ( ! valide ) {
      event.preventDefault();
   }
   valide;
}
```

XHTML :

```
<body onload="init();">
```

```
function valider( event ) {
    var champs = document.getElementsByTagName( "input" );
    var valide = true;
    for ( var i=0; valide && i<champs.length; i++ ) {
        var champ = champs[ i ];
        var nom = champ.getAttribute( "name" );
        var valeur = champ.value;

        if ( nom != null && /^\s*$/.test( valeur ) ) {
            alert( "Veuillez entrer une valeur pour le champ " + nom );
            valide = false;
        } else if ( nom == "codepostal" ) {
            if ( ! validerCodePostal( champ ) ) {
                valide = false;
            }
        }
        if ( ! valide ) {
            event.preventDefault();  // Ne pas soumettre les données au serveur
        }
    }
    valide;
}
```

Générer un événement click lorsque l'usager sélectionne un élément d'un menu

preventDefault (suite)

- L'usager clic sur un lien qui mène vers une page d'un autre domaine

Génération d'événements

- **select** : éléments **input** (**text**, **file** ou **passwd**) et **textarea**
- **focus** : **anchor**, **input**, **select** et **textarea**
- **click** : **input** (**button**, **checkbox**, **radio**, **reset** ou **submit**)
- **blur** : **anchor**, **input**, **select** ou **textarea**

```
function valider( event ) {
    var champs = document.getElementsByTagName( "input" );
    var valide = true;
    for ( var i=0; valide && i<champs.length; i++ ) {
        var champ = champs[ i ];
        var nom = champ.getAttribute( "name" );
        var valeur = champ.value;

        if ( nom != null && /^\s*$/.test( valeur ) ) {
            alert( "Veuillez entrer une valeur pour le champ " + nom );
            valide = false;
        } else if ( nom == "codepostal" ) {
            if ( ! validerCodePostal( champ ) ) {
                valide = false;
            }
        }
        if ( ! valide ) {
            event.preventDefault();  // Ne pas soumettre les données au serveur
            champ.select(); // Sélectionner le champ erroné
        }
    }
    valide;
}
```

Générer un événement click lorsque l'usager sélectionne un élément d'un menu

Génération d'événements

- Générer un événement **click** suite à la sélection de l'un des éléments d'un menu...
- L'usager sélectionne un pays, le script envoie l'information au serveur qui revient avec un nouveau formulaire ayant un menu «provinces»
- Suite à la sélection de la province, le script soumet l'information au serveur, et le nouveau formulaire contient maintenant un choix de villes...

Remarques

- J’ai remarqué certaines inconsistances lorsque j’ajoute un gestionnaire à l’aide d’**addEventListener** vs **événement intrinsèque**
- Qu’arrive-t-il si le document n’est pas valide?

Remarques

- La méthode **removeEventListener** existe
- Évidemment, pour IE, les principes sont les mêmes, mais la syntaxe est différente...

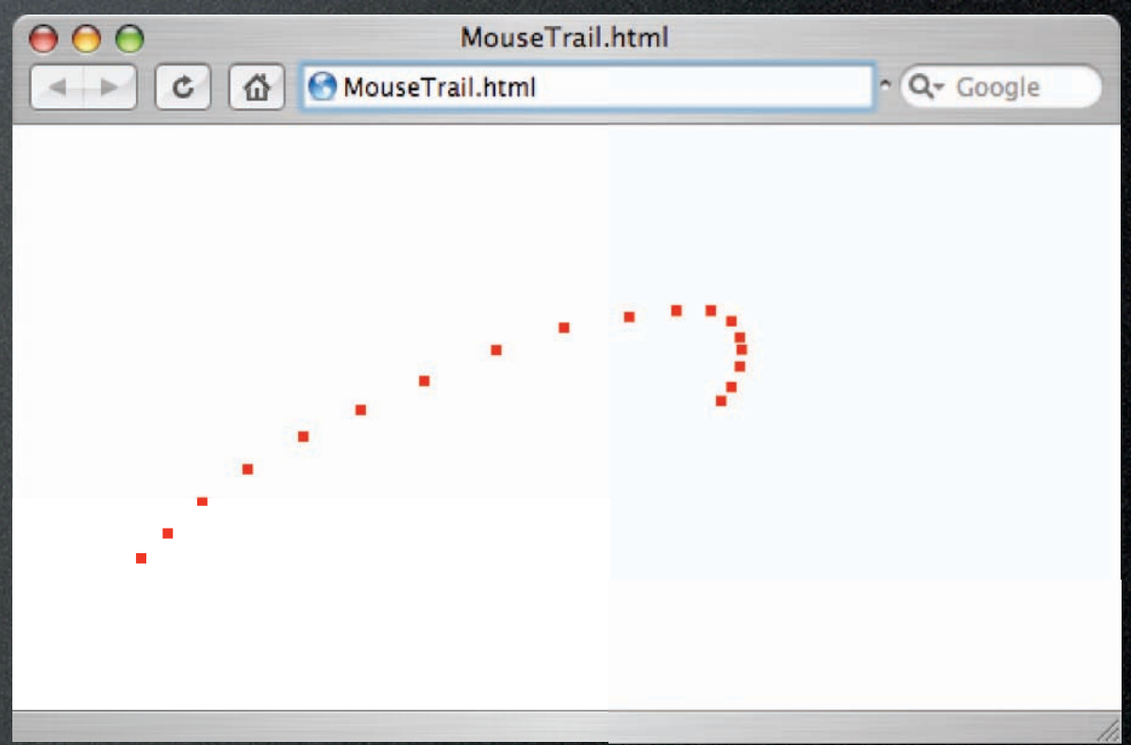
Environnement hôte (suite)

- En plus des objets définis par le DOM, les environnements hôtes offrent plus fonctionnalités :
  - open(URI, Target), resizeTo (Number, Number), print(), etc.

```
<body onload="open( 'http://www.uottawa.ca', '_blank' );
open( 'http://bio.site.uottawa.ca', '_blank' );
open( 'http://www.google.com', '_blank' );">
```

Autres applications DOM/ JavaScript (côté client)

- Jeux, jeux, jeux :
  - Solitaire et autres jeux de cartes, sudoku, mahjong
  - Calculette, générateur de mots de passe
- Effets visuels



Revue des concepts

- JavaScript + DOM permet la création de pages dynamiques HTML (on parle parfois de DHTML)
- Accès et manipulation du document HTML
- Associer des actions à des événement
- Contrôle assez sophistiquer du traitement et de la propagation des événements

Modules

- DOM Core
  - Accès, manipulation et créations d'objets associés au document
  - DOM Event
    - L'objet Event donne des informations sur le contexte
    - addEventListener associer des gestionnaires aux éléments
    - Gestion dans le contexte de l'arbre

Modules (suite)

- DOM HTML
  - Application de DOM Core à XHTML
    - Objets spécifiques telles que HTMLDocument, HTMLElement
- DOM Style
  - Ajout, retrait, définition de règles

Modules (suite)

- DOM Traversal and Range
  - NodeIterator, TreeWalker, NodeFilter

```
NodeIterator iter=
    ((DocumentTraversal)document).createNodeIterator(
        root, NodeFilter.SHOW_ELEMENT, null);

while (Node n = iter.nextNode())
    printMe(n);
```

Ressources

- La spécification du modèle objet de document (DOM) niveau 2 Core [http://www.yoyodesign.org/doc/w3c/dom2-core/Overview.html] 2007
- La spécification du modèle objet de document (DOM) niveau 2 HTML [http://www.yoyodesign.org/doc/w3c/dom2-html/Overview.html] 2007

Ressources

- http://www.w3.org/DOM/DOMTR