HTML5 avancé

Communication - Web Messaging

Nouvelles méthodes de communication

- Possibilités de partager des données entre documents web de différents contextes de navigation
  - Environnement du document : onglets, fenêtres, iframes, ...

- Modèle de communication commun
  - Web messaging
    - *Cross-document messaging*
    - *Channel messaging*
  - Server sent events
  - Web Sockets

- Spécification : http://www.w3.org/TR/html5/comms.html

Nouvelles méthodes de communication → Pourquoi ?

- HTTP : protocole déconnecté
  - Pour faire du temps réel, on utilise jusque là des solutions de polling ou long polling
  - Pas de solution de réel « push » entre serveur et client
    - *HTML5 : websocket + Server sent events*

- Cross-site scripting : sécurité des navigateurs
  - Les documents de différents domaines ou contextes ne peuvent pas communiquer ou s'affecter
    - *HTML5 : Web messaging*

L'API de communication

- Les messages sont transmis à travers un MessageEvent

```
interface MessageEvent : Event {
  readonly attribute any data;
  readonly attribute DOMString origin;
  readonly attribute DOMString lastEventId;
  readonly attribute WindowProxy source; //windowProxy
  readonly attribute MessagePortArray ports;

  void initMessageEvent(in DOMString typeArg, in boolean
canBubbleArg, in boolean cancelableArg, in any dataArg, in
DOMString originArg, in DOMString lastEventIdArg, in
WindowProxy sourceArg, in MessagePortArray portsArg);
};
```

- L'attribut data contient le corps du message
- Des méthodes sont à disposition pour gérer les messages
  - postMessage ou send pour envoi
  - onmessage pour écouter les réceptions

Web messaging

- Anciennement PostMessage API http://dev.w3.org/html5/postmsg/
- But : communication inter contexte et origines
- Deux méthodes :
  - Cross-document messaging
  - Channel messaging

Cross document messaging

- Principe :
  - Permettre aux documents de contextes différents de communiquer quelque soit leurs origines
  - Exemple : communication entre une iframe et sa page parente

- Origine : domaine + protocole + port
  - https://www.site.com, http://www.site.com, http://site.com, http://www.site.com:81, ont tous une origine différente

Cross-document messaging

- Envoi de message à window avec postMessage

```
otherWindow.postMessage(message, targetOrigin [, transfer ]) ;
```

  - Les messages peuvent être des objets ou tableaux de strings, entiers, Date, File, Blob, Filelist (File API), …
  - Les objets listé dans `transfer` sont transférés (sans copie)
  - `TargetOrigin` : « origin » qui reçoit le message
    - *"*" pour des origines différentes, "\" pour restreindre à la même origine que `window`, ou une url (même origine)*

- Réception de messages

```
window.onmessage = function(event) {
    if(event.origin == 'http://www.zenika.com'){
        //récupérer le message : event.data
    }
}
```

  - Ou écouter l'événement "message" avec addEventListener

Cross-document messaging

- Exemple : envoi de message entre une page et une iframe

```
//document A at http://domainA.com
<iframe id="iframeB" src="http://domainB.com/B.html"
onload="sendCommand();"></iframe>

<script>
    function sendCommand() {
        var iframeB = document.getElementById("iframeB");
        iframeB.contentWindow.postMessage("Hello iframeB",
"http://domainB.com");
    }
</script>
```

```
//document B at http://domainB.com
<script>
    window.onmessage = function(event) {
    if(event.origin == 'http://domainA.com'){
        alert(event.data); //="Hello iframeB"
    }
}
</script>
```

Cross-document messaging Support

- Navigateurs :
  - Chrome 4.0+
  - Firefox 3.0+
  - Safari 4.0+
  - Opera 9.5+

- Tester l'implémentation

```
if (!!window.postMessage) {

} else {
   console.log("Votre navigateur ne supporte pas PostMessage");
}
```

Cross-document messaging Sécurité

- Toujours vérifier l'origine du message
  - Ne pas accepter des messages d'origines inconnues
- Vérifier le contenu et format des données transmises
  - Ne pas utiliser le message directement (dans un formulaire par exemple).
- Attention au "*"
  - La destination du message n'est pas assurée
  - Ne pas transmettre de données confidentielles
- Possibilités d'attaques de type DoS
  - Limiter le taux de réception de messages
  - Attention au volume des messages reçus

Channel messaging

- Communication bi-directionnelle, asynchrone, entre documents de contextes de navigation différents
  - Ouvre un canal de communication
  - Messages envoyés d'un port à l'autre sous forme d'événements (MessageEvent)
- Intérêt par rapport à cross-document messaging
  - Communication inter-origines précise
  - Contrôle qui envoi et reçoit les messages
- Cas d'utilisation
  - Sites sociaux, applications « mashup »

Channel messaging
Principe d'utilisation

- Créer le canal de communication

```
var channel = new MessageChannel();
```

- Envoyer le port avec lequel communiquer

```
window.postMessage('port', 'http://othersite.com', [channel.port2]);
```

- L'autre page récupère ce port et peut envoyer des messages

```
window.onmessage = function(event) {
    //check origin
    var port = evt.ports[0];
    port.postMessage('port received');
}
```

- Les deux ports sont à présent « impliqués »

Channel messaging API

- Deux interfaces :
  - MessageChannel

```
interface MessageChannel {
  readonly attribute MessagePort port1;
  readonly attribute MessagePort port2;
};
```

  - MessagePort

```
interface MessagePort : EventTarget {
  void postMessage(any message, optional sequence<Transferable>
transfer);
  void start();
  void close();

  // event handlers
          attribute EventHandler onmessage;
};
MessagePort implements Transferable;
```

Channel messaging

- Lorsqu'on crée le « MessageChannel », il en résulte deux ports non reliés.
  - Un reste local, l'autre est envoyé au contexte destinataire
  - On appel start() pour ouvrir le port et activer la réception de messages
  - La méthode close() ferme le port et coupe le canal de communication
  - Les deux ports communiquent avec postMessage

```
channel.port1.postMessage('msg from port1');
```

  - On utilise la méthode onMessage() pour écouter

```
channel.port1.onmessage = function(event) {
  // le message est dans event.data
  // ...
}
```

Channel messaging
Support et état de spécification

- Spécification en cours de rédaction
- Non présente sur caniuse et html5 test
  - Seul Chrome semble la supporter correctement, Opera et Safari ont un support partiel.
- Tester le support :
  - Définition de `window.MessageChannel`
  - L'existence des ports suite à la création du channel ; l'envoi d'un des ports et sa bonne réception

```
if (window.MessageChannel === undefined){
    //alert
} else {
    //create channel and send port
}
```

```
window.onmessage = function(event) {
    var port = evt.ports[0];
    if(port === undefined){ //alert }
}
```

TP