zenika
ARCHITECTURE INFORMATIQUE

HTML5 avancé

Communication – Server Sent Events

Server Sent Events

- Spécification connue aussi sous le nom de EventSource
  - http://www.w3.org/TR/eventsource/
- Principe :
  - L'application reçoit automatiquement des messages envoyés par le serveur sous forme d'événement DOM
  - Communication dans un seul sens en HTTP
- Exemples de cas d'utilisation :
  - Notifications temps réel, actualités
  - Réseaux sociaux : fil twitter/facebook temps réel
  - Cours des actions en temps réel
  - Monitoring de serveur

L'API de communication
Rappel

- Interface MessageEvent utilisée pour envoyer les messages

```
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);
};
```

EventSource

- L'API utilise l'interface EventSource

```
[Constructor(DOMString url, optional EventSourceInit
eventSourceInitDict)]
interface EventSource : EventTarget {
  readonly attribute DOMString url;
  readonly attribute boolean withCredentials;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute unsigned short readyState;

  // networking
  [TreatNonCallableAsNull] attribute Function? onopen;
  [TreatNonCallableAsNull] attribute Function? onmessage;
  [TreatNonCallableAsNull] attribute Function? onerror;
  void close();
};

dictionary EventSourceInit {
  boolean withCredentials = false;
};
```

EventSource
Coté client

- Connexion au serveur : création d'un objet EventSource

```
var source = new EventSource('/source'); //url à laquelle on se
                                         //connecte
source.onmessage = function (event) {
  alert(event.data); //récupérer le message
};

source.onopen = function(e) {
   console.log("connexion établie");
});

source.onerror = function(e) {
   console.log("eventsource error : " + e.data);
});
```

- On vérifie l'état de la connexion avec l'attribut `readyState`
- Requête HTTP : contenu demandé event-stream

```
Accept:text/event-stream
```

EventSource
Coté serveur

- Format des données envoyées
  - Type MIME : text/event-stream
  - Encodage : UTF-8
  - Une ligne entre chaque message à envoyer

```
data: message 1

data: message 2 ligne 1
data: message 2 ligne 2
```

- En-tête HTTP

```
Content-Type: text/event-stream
Cache-Control: no-cache
```

- Il existe 4 champs interprétés pour un event stream :
  - `data, event, id, retry`

EventSource
Event stream

- Définir le nom du message à envoyer avec `event`

```
event: time
data: 16:21

event: tweet
data: tweet from ...
```

  - Permet de spécifier des types de messages
    - *Par défaut « message » : géré par onmessage()*

```
source.addEventListener('time', function(e){
    //e.data
}, false);

source.addEventListener('tweet', function(e){
    //e.data
}, false);
```

EventSource
Event stream

- Utiliser les id
  - On peut associer un identifiant à un message

```
id: 13
data: hello world

id: 14
event: news
data: {"title": "last new", "article": "..."}
```

  - L'attribut `lastEventId` du MessageEvent contient le dernier id de la source
  - En cas de déconnexion, une requête avec le header `last-event-id` est envoyée
  - Le navigateur pourra ainsi déterminer le dernier message envoyé

EventSource
Event stream

- Le champ `retry` sert à contrôler le timeout de reconnexion (par défaut ~3s)

```
retry: 10000
```

- Envoyer des données json

```
data: {
data: "msg": "hello world",
data: "time": "19:05"
data: }

data: ...
```

  - Le client peut ensuite parser les données reçues

```
source.onmessage = function(event) {
  var data = JSON.parse(event.data);
});
```

EventSource Exemple PHP

```php
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache'); // recommended to prevent caching of event data.

/**
 * Constructs the SSE data format and flushes that data to the client.
 *
 * @param string $id Timestamp/id of this connection.
 * @param string $msg Line of text that should be transmitted.
 */
function sendMsg($id, $msg) {
  echo "id: $id" . PHP_EOL;
  echo "data: $msg" . PHP_EOL;
  echo PHP_EOL;
  ob_flush();
  flush();
}

$serverTime = time();

sendMsg($serverTime, 'server time: ' . date("h:i:s", time()));
```

EventSource
Exemple NodeJS

```
serv.get('/stream', function(req, resp){
    resp.writeHead(200, {
        'Content-Type': 'text/event-stream',
        'Cache-Control': 'no-cache',
        'Connection': 'keep-alive'
    });

    var idCounter = 0;

    setInterval(function(){
        resp.write("event: tweet\n");
        resp.write("id: "+idCounter+"\n");
        resp.write("data:{'user':'toto','tweet':'RT @ZenikaIT
Awesome HTML5 training !'}\n\n");
        idCounter++;
    },5000);

    req.on("close", function() {
        // Connexion fermée!
    });
});
```

SSE et CORS

- Le CORS n'est pas supporté pour le SSE d'après les standards
- Cependant Firefox > 11 le supporte
- Cela devrait devenir un standard prochainement

Support coté client

- Navigateurs :
  - Chrome 6.0+
  - Firefox 6.0+
  - Safari 5.0+
  - Opera 11.0+

- Tester l'implémentation

```
if (!!window.EventSource) {
    //connexion et écoute des événements
} else {
   console.log("Votre navigateur ne supporte pas EventSource");
}
```

TP