1. Web Workers

Với công nghệ phần cứng hiện nay, việc sử dụng đa luồng đã trở nên một phần không thể thiếu trong các phần mềm. Tuy nhiên, công nghệ thiết kế web vẫn chưa tận dụng được sức mạnh này. Với các công việc đòi hỏi một quá trình xử lý lâu, lập trình viên thường phải sử dụng những thủ thuật như setTimeout(), setInterval(),… để thực hiện từng phần công việc. Hiện nay, để giải quyết vấn đề này, một API mới đã xuất hiện với tên gọi Web Worker.

Đối tượng Web Worker được tạo ra sẽ thực thi trong một thread độc lập và chạy ở chế độ nền nên không ảnh hưởng đến giao diện tương tác của trang web với người dùng. Với đặc điểm này, chúng ta có thể sử dụng Web Worker trong các công việc đòi hỏi thời gian xử lý lâu nạp dữ liệu, tạo cache,…

Điểm hạn chế của Web Worker là không thể truy xuất được thành phần trên DOM, và cả các đối tượng **window**, **document** hay **parent**. Mã lệnh các công việc cần thực thi cũng phải được cách ly trong một tập tin script.

* Kiểm tra hỗ trợ

Trước khi tạo một web worker, kiểm tra xem trình duyệt của người dùng có hỗ trợ:

if(typeof(Worker)!=="undefined")

{

// Yes! Web worker support!

// Some code.....

}

else

{

// Sorry! No Web Worker support..

}

* Tạo một tệp Web Worker

Ví du, tạo một tập tin “demo\_workers.js”. Trong đó là đoạn mã để đếm tăng dần theo thời gian:

var i=0;

function timedCount()

{

i=i+1;

postMessage(i);

setTimeout("timedCount()",500);

}

timedCount();

Tất nhiên chúng ta không sử dụng web workers để làm chuyện đơn giản như vậy, thường nó được sử dụng cho các tác vụ chuyên sâu hơn.

* Tạo một đối tượng Web Worker

Khi đã tập tin web worker đã sẵn sàng, bây giờ ta gọi nó hoạt động từ trang web. Tạo một đối tượng web workers, kiểm tra xem nó tồn tại hay chưa, nếu chưa thì tạo ra và chạy đoạn mã trong tập tin web workers:

if(typeof(w)=="undefined")

{

w=new Worker("demo\_workers.js");

}

Khi tạo xong, bắt đầu nó bằng cách gọi phương thức postMessage():

worker.postMessage(); // Start the worker.

Sau đó chúng ta có thể nhận và gởi tin nhắn đến web workers, thêm sự kiện “onmessage” để lắng nghe web workers:

w.onmessage=function(event){

document.getElementById("result").innerHTML=event.data;

};

* Hủy Web Worker

Khi một đối tượng web workers được tạo ra, nó sẽ lắng nghe các tin nhắn cho tới khi nó được hủy. Để hủy một web workers, giải phóng tài nguyên, ta dùng phương thức terminate():

w.terminate();

1. Web Sockets

WebSocket là một API xây dựng kết nối "socket" giữa trình duyệt và máy chủ. Nói một cách đơn giản: có một kết nối liên tục giữa máy client và máy chủ vì vậy cả hai bên có thể bắt đầu gửi dữ liệu bất cứ lúc nào.

Sử dụng WebSocket có thể tạo ra độ trễ thấp hơn, kết nối thì gần như là kết nối thời gian thực. Nhờ vào các đặt điểm nổi bật đó mà WebSocket thường được sử dụng để:

* **Tạo các trò chơi trực tuyến nhiều người.**
* Tạo ứng dụng chat qua mạng.
* Đặt vé trực tuyến.
* Cập nhật tin tức mạng xã hội.
* Tạo kết nối

Để mở một kết nối websocket, chỉ đơn giản bằng cách gọi:

var connection = new WebSocket('ws://html5rocks.websocket.org/echo', ['soap', 'xmpp']);

* Thêm xử lý sự kiện

Sau khi kết nối vừa tạo xong, tiếp theo, thêm các xử lý sự kiện, các sự kiện như kết nối được mở, nhận tin nhắn đến, xảy ra lỗi, …

*// When the connection is open, send some data to the server*

connection.onopen = function () {

connection.send('Ping'); *// Send the message 'Ping' to the server*

};

*// Log errors*

connection.onerror = function (error) {

console.log('WebSocket Error ' + error);

};

*// Log messages from the server*

connection.onmessage = function (e) {

console.log('Server: ' + e.data);

};

* Giao tiếp với server

Ngay khi có một kết nối đến máy chủ, chúng ta có thể bắt đầu gửi dữ liệu đến máy chủ bằng cách sử dụng phương thức send(‘your message’) trên đối tượng kết nối. Nó hỗ trợ gởi các chuỗi duy nhất, nhưng trong phiên bản mới nhất, đã có thể gửi tin nhắn dạng nhị phân. Để gửi dữ liệu nhị phân, ta có thể sử dụng một trong hai đối tương Blob hoặc ArrayBuffer.

*// Sending String*

connection.send('your message');

*// Sending canvas ImageData as ArrayBuffer*

var img = canvas\_context.getImageData(0, 0, 400, 320);

var binary = new Uint8Array(img.data.length);

for (var i = 0; i < img.data.length; i++) {

binary[i] = img.data[i];

}

connection.send(binary.buffer);

*// Sending file as Blob*

var file = document.querySelector('input[type="file"]').files[0];

connection.send(file);

Tương tự như vậy, server cũng có thể gởi messages cho client bất cứ khi nào.

Để xác định các định dạng nhị phân nhận được, thiết lập thuộc tính binaryType của WebSocket là 'blob' hoặc 'arraybuffer'. Định dạng mặc định là 'blob':

*// Setting binaryType to accept received binary as either 'blob' or 'arraybuffer'*

connection.binaryType = 'arraybuffer';

connection.onmessage = function(e) {

console.log(e.data.byteLength); *// ArrayBuffer object if binary*

};