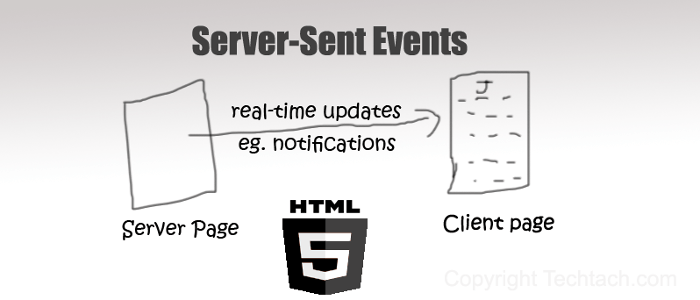
**Server-Sent Events和WebSocket服务器向浏览器推送信息**

服务器向浏览器推送信息，除了 [WebSocket](http://www.ruanyifeng.com/blog/2017/05/websocket.html)，还有一种方法：Server-Sent Events（以下简称 SSE）。本文介绍它的用法。



## 一、SSE 的本质

严格地说，[HTTP 协议](http://www.ruanyifeng.com/blog/2016/08/http.html)无法做到服务器主动推送信息。但是，有一种变通方法，就是服务器向客户端声明，接下来要发送的是流信息（streaming）。

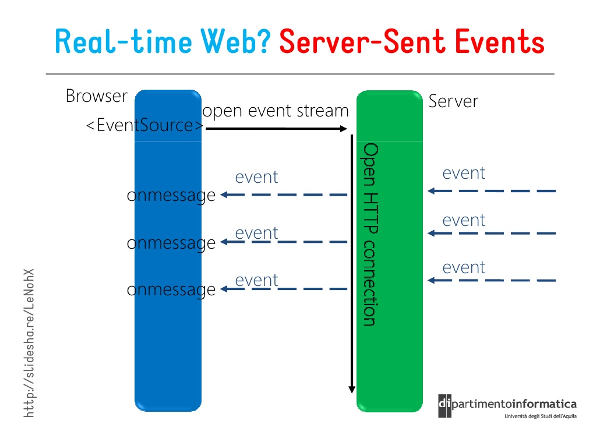
也就是说，发送的不是一次性的数据包，而是一个数据流，会连续不断地发送过来。这时，客户端不会关闭连接，会一直等着服务器发过来的新的数据流，视频播放就是这样的例子。本质上，这种通信就是以流信息的方式，完成一次用时很长的下载。

SSE 就是利用这种机制，使用流信息向浏览器推送信息。它基于 HTTP 协议，目前除了 IE/Edge，其他浏览器都支持。

## 二、SSE 的特点

SSE 与 WebSocket 作用相似，都是建立浏览器与服务器之间的通信渠道，然后服务器向浏览器推送信息。

总体来说，WebSocket 更强大和灵活。因为它是全双工通道，可以双向通信；SSE 是单向通道，只能服务器向浏览器发送，因为流信息本质上就是下载。如果浏览器向服务器发送信息，就变成了另一次 HTTP 请求。



但是，SSE 也有自己的优点。

* SSE 使用 HTTP 协议，现有的服务器软件都支持。WebSocket 是一个独立协议。
* SSE 属于轻量级，使用简单；WebSocket 协议相对复杂。
* SSE 默认支持断线重连，WebSocket 需要自己实现。
* SSE 一般只用来传送文本，二进制数据需要编码后传送，WebSocket 默认支持传送二进制数据。
* SSE 支持自定义发送的消息类型。

因此，两者各有特点，适合不同的场合。

## 三、客户端 API

### 3.1 EventSource 对象

SSE 的客户端 API 部署在EventSource对象上。下面的代码可以检测浏览器是否支持 SSE。

1. if ('EventSource' in window) {
2. *// ...*
3. }

使用 SSE 时，浏览器首先生成一个EventSource实例，向服务器发起连接。

1. var source = new EventSource(url);

上面的url可以与当前网址同域，也可以跨域。跨域时，可以指定第二个参数，打开withCredentials属性，表示是否一起发送 Cookie。

1. var source = new EventSource(url, { withCredentials: true });

EventSource实例的readyState属性，表明连接的当前状态。该属性只读，可以取以下值。

* 0：相当于常量EventSource.CONNECTING，表示连接还未建立，或者断线正在重连。
* 1：相当于常量EventSource.OPEN，表示连接已经建立，可以接受数据。
* 2：相当于常量EventSource.CLOSED，表示连接已断，且不会重连。

### 

### 3.2 基本用法

连接一旦建立，就会触发open事件，可以在onopen属性定义回调函数。

1. source.onopen = function (event) {
2. *// ...*
3. };
4. *// 另一种写法*
5. source.addEventListener('open', function (event) {
6. *// ...*
7. }, false);

客户端收到服务器发来的数据，就会触发message事件，可以在onmessage属性的回调函数。

1. source.onmessage = function (event) {
2. var data = event.data;
3. *// handle message*
4. };
5. *// 另一种写法*
6. source.addEventListener('message', function (event) {
7. var data = event.data;
8. *// handle message*
9. }, false);

上面代码中，事件对象的data属性就是服务器端传回的数据（**文本格式**）。

如果发生通信错误（比如连接中断），就会触发error事件，可以在onerror属性定义回调函数。

1. source.onerror = function (event) {
2. *// handle error event*
3. };
4. *// 另一种写法*
5. source.addEventListener('error', function (event) {
6. *// handle error event*
7. }, false);

close方法用于关闭 SSE 连接。

1. source.close();

### 3.3 自定义事件

默认情况下，服务器发来的数据，总是触发浏览器EventSource实例的message事件。开发者还可以自定义 SSE 事件，这种情况下，发送回来的数据不会触发message事件。

1. source.addEventListener('foo', function (event) {
2. var data = event.data;
3. *// handle message*
4. }, false);

上面代码中，浏览器对 SSE 的foo事件进行监听。如何实现服务器发送foo事件，请看下文。

## 四、服务器实现

### 4.1 数据格式

服务器向浏览器发送的 SSE 数据，必须是 UTF-8 编码的文本，具有如下的 HTTP 头信息。

Content-Type: text/event-stream

Cache-Control: no-cache

Connection: keep-alive

上面三行之中，第一行的Content-Type必须指定 MIME 类型为event-steam。

每一次发送的信息，由若干个message组成，每个**message之间**用**\n\n**分隔。每个message内部由若干行组成，每一行都是如下格式。

**[field]: value\n\n**

**请看下面实例:**

router.get('/ServerSent', function(req, res, next) {

  res.setHeader(**'Content-Type'**, **'text/event-stream'**);

  res.setHeader(**'Cache-Control'**, **'no-cache'**);

  res.end("data: The server time is"+new Date()+"\n\n");

});

**上面的field可以取四个值。**

* **data**
* **event**
* **id**
* **retry**

此外，还可以有冒号开头的行，表示注释。通常，服务器每隔一段时间就会向浏览器发送一个注释，保持连接不中断。

: This is a comment

下面是一个例子。

: this is a test stream\n\n

data: some text\n\n

data: another message\n

data: with two lines \n\n

### 

### 4.2 data 字段

数据内容用data字段表示。

data: message\n\n

如果数据很长，可以分成多行，最后一行用\n\n结尾，前面行都用\n结尾。

data: begin message\n

data: continue message\n\n

下面是一个发送 JSON 数据的例子。

data: {\n

data: "foo": "bar",\n

data: "baz", 555\n

data: }\n\n

### 

### 4.3 id 字段

数据标识符用id字段表示，相当于每一条数据的编号。

**id: msg1\n**

**data: message\n\n**

浏览器用lastEventId属性读取这个值。一旦连接断线，浏览器会发送一个 HTTP 头，里面包含一个特殊的Last-Event-ID头信息，将这个值发送回来，用来帮助服务器端重建连接。因此，这个头信息可以被视为一种同步机制。

### 

### 4.4 event 字段

event字段表示自定义的事件类型，默认是message事件。浏览器可以用addEventListener()监听该事件。

event: foo\n

data: a foo event\n\n

data: an unnamed event\n\n

event: bar\n

data: a bar event\n\n

上面的代码创造了三条信息。第一条的名字是foo，触发浏览器的foo事件；第二条未取名，表示默认类型，触发浏览器的message事件；第三条是bar，触发浏览器的bar事件。

下面是另一个例子。

event: userconnect

data: {"username": "bobby", "time": "02:33:48"}

event: usermessage

data: {"username": "bobby", "time": "02:34:11", "text": "Hi everyone."}

event: userdisconnect

data: {"username": "bobby", "time": "02:34:23"}

event: usermessage

data: {"username": "sean", "time": "02:34:36", "text": "Bye, bobby."}

### 

### 4.5 retry 字段

服务器可以用retry字段，指定浏览器重新发起连接的时间间隔。

retry: 10000\n

两种情况会导致浏览器重新发起连接：一种是时间间隔到期，二是由于网络错误等原因，导致连接出错。

## 五、Node 服务器实例

SSE 要求服务器与浏览器保持连接。对于不同的服务器软件来说，所消耗的资源是不一样的。Apache 服务器，每个连接就是一个线程，如果要维持大量连接，势必要消耗大量资源。Node 则是所有连接都使用同一个线程，因此消耗的资源会小得多，但是这要求每个连接不能包含很耗时的操作，比如磁盘的 IO 读写。

下面是 Node 的 SSE 服务器[实例](http://cjihrig.com/blog/server-sent-events-in-node-js/)。

1. var http = require("http");
2. http.createServer(function (req, res) {
3. var fileName = "." + req.url;
4. if (fileName === "./stream") {
5. res.writeHead(200, {
6. "Content-Type":"text/event-stream",
7. "Cache-Control":"no-cache",
8. "Connection":"keep-alive",
9. "Access-Control-Allow-Origin": '\*',
10. });
11. res.write("retry: 10000\n");
12. res.write("event: connecttime\n");
13. res.write("data: " + (new Date()) + "\n\n");
14. res.write("data: " + (new Date()) + "\n\n");
16. interval = setInterval(function () {
17. res.write("data: " + (new Date()) + "\n\n");
18. }, 1000);
20. req.connection.addListener("close", function () {
21. clearInterval(interval);
22. }, false);
23. }
24. }).listen(8844, "127.0.0.1");

请将上面的代码保存为server.js，然后执行下面的命令。

1. $ node server.js

上面的命令会在本机的8844端口，打开一个 HTTP 服务。

然后，打开这个[网页](http://jsbin.com/vuziboduwa/edit?html,output)，查看客户端代码并运行。

## 

## 六、参考链接

* Colin Ihrig, [Implementing Push Technology Using Server-Sent Events](http://jspro.com/apis/implementing-push-technology-using-server-sent-events/)
* Colin Ihrig，[The Server Side of Server-Sent Events](http://cjihrig.com/blog/the-server-side-of-server-sent-events/)
* Eric Bidelman, [Stream Updates with Server-Sent Events](http://www.html5rocks.com/en/tutorials/eventsource/basics/)
* MDN，[Using server-sent events](https://developer.mozilla.org/en-US/docs/Server-sent_events/Using_server-sent_events)
* Segment.io, [Server-Sent Events: The simplest realtime browser spec](https://segment.io/blog/2014-04-03-server-sent-events-the-simplest-realtime-browser-spec/)

（完）

来源地址：<https://blog.csdn.net/qq_25072517/article/details/78479777>

<https://blog.csdn.net/w470543882/article/details/51459171>