

18 | 如何使用Canvas绘制统计图表（下）？

2019-08-24 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 15:49 大小 10.87M



Google 的 Chrome 浏览器已经默认支持 WebRTC 库了，因此 Chrome 浏览器之间已经可以进行音视频实时通信了。更让人欣喜的是 Google 还开源了 WebRTC 源码，此举不仅惊艳，而且非常伟大。WebRTC 源码的开放，为音视频实时通信领域从业者、爱好者提供了非常好的研究和学习的机会。

虽然“浏览器 + WebRTC”为广大用户提供了诸多便利，但当你开发产品时会发现，在浏览器上调试**媒体流**还是非常困难的。因为媒体通信涉及到了多个层面的知识，而浏览器更擅长的是处理 HTML 页面和 JavaScript 脚本，所以如果用它来分析媒体流的收发情况或者网络情况，就显得很困难了。

为了解决这个问题，Google 在它的 Chrome 浏览器中支持了 WebRTC 的统计分析功能，只要在 **Chrome 浏览器的地址栏**输入 “chrome://webrtc-internals/”，你就可以看到

浏览器中正在使用的 WebRTC 的各种统计分析数据了，而且这些数据都是以可视化统计图表的方式展现在你面前的，从而大大方便了你分析媒体流的效率。

实际上，关于 WebRTC 统计方面的内容我在前面《WebRTC 中的数据统计原来这么强大》的两篇文章中已经做了详细的介绍。而今天我们要讲的主要内容是**如何使用 Canvas 进行图表的绘制**。

浏览器中的 WebRTC 统计图表

下面我们先通过一个实际的例子，感受一下在 Chrome 浏览器中是如何通过统计图表来展现 WebRTC 的统计信息的。要想看到这个图表，你需按以下步骤操作：

在 Chrome 浏览器中同时打开两个 tab 页面；

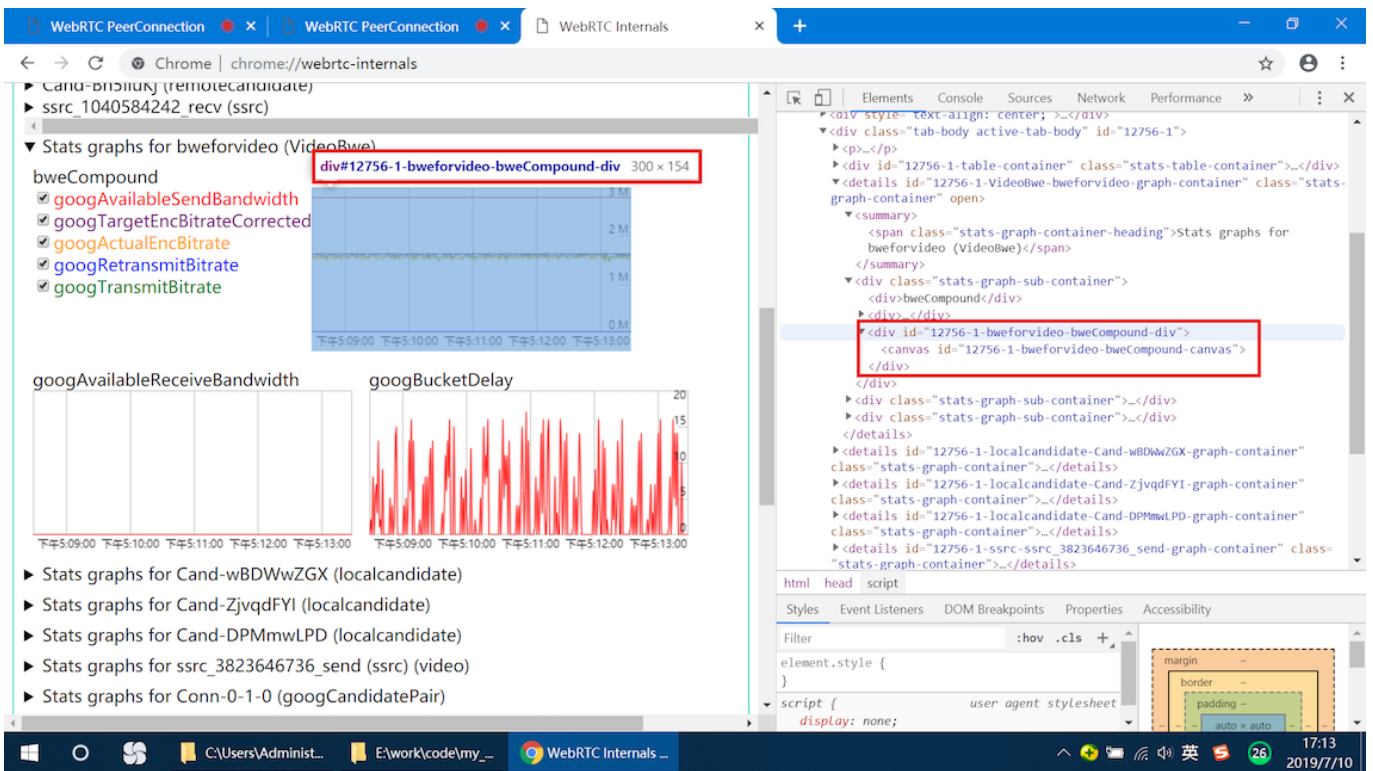
在两个 tab 页面中输入 <https://learningrtc.cn/getstats/index.html> 地址，这是一个用于测试 WebRTC 的 URL 地址；

在每一个打开的页面中，点击“Connect Sig Server”按钮，此时每个页面都会出现两个视频窗口，这说明 WebRTC 视频通信已经建立成功了；

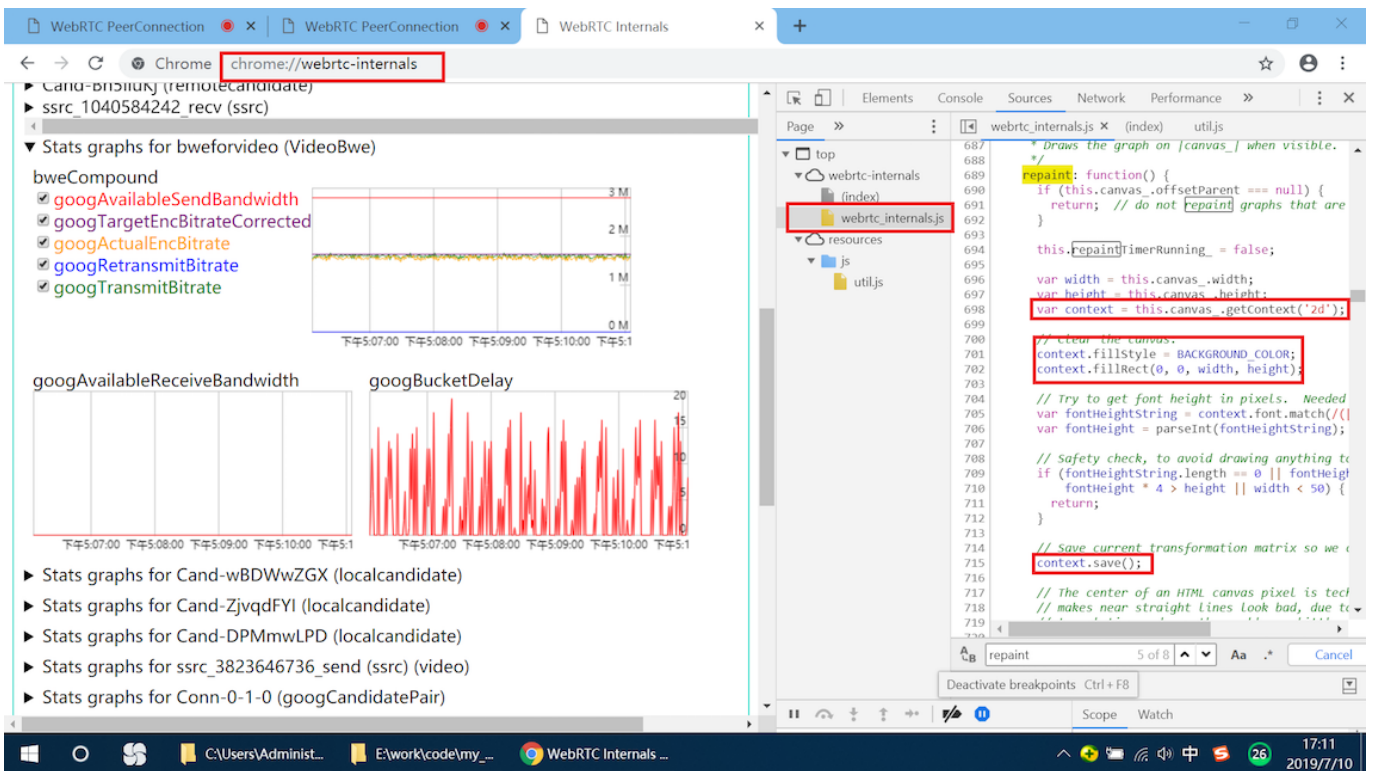
在 Chrome 浏览器中再打开一个 tab 页面（也就是第三个页面），输入 `chrome://webrtc-internals/` 地址，此时，在这个页面中就会展示正在运行的 WebRTC 的各种统计信息，如媒体流统计信息、网络统计信息等；

你可以继续点开任意一个带有“**Stats graphs**”字样的选项，这样相关的统计图表就会展示出来了。

在这些统计图表中，你可以看到每一路音视频流的收发包、传输码率、带宽等信息。下面两张图展示的就是视频相关的统计信息。




WebRTC 统计信息图 (一)



WebRTC 统计信息图 (二)

在统计信息页面中，你可以点击鼠标右键，在弹出的菜单中选择“检查”，就会显示出该页面的 HTML 源码，也就是上面两张图中右半部分的内容。下面我们就对这个源码做一下简单的分析。

在 `chrome://webrtc-internals/` 地址源码的 `<head>` 标签中，引用了 `webrtc_internals.js` 和 `util.js` 两个 JavaScript 脚本，代码如下所示：

 复制代码

```
1 ...
2 <head>
3     ...
4     <script src="chrome://resources/js/util.js"></script>
5     <script src="webrtc_internals.js"></script>
6     ...
7 </head>
8 ...
```

在这两个脚本中，最关键的是 `webrtc_internals.js` 脚本，因为**所有统计信息的绘制都在 `webrtc_internals.js` 中完成的。**

那这些图表是怎么绘制出来的呢？为了解开这个迷团，我们来观察一下“WebRTC 统计信息图（一）”这张图。在这张图中，左侧红框框中的信息表示的是 id 为“**12756-1-bweforvideo-bweCompound-div**”的 DIV，在这个 DIV 中绘制了一张**发送视频带宽**的图表。然后，我们再来看一下这张图表所对应的代码，也就是图中右侧红框框中的 HTML5 代码，从中我们可以知道，**左侧的图表是由右侧的 HTML5 代码中的 `<canvas>` 标签绘制而成的。**

在“WebRTC 统计信息图（二）”中，我在图的右侧用红框选中了 `webrtc_internals.js`，在该脚本的源码中，我们能够看到在 `webrtc_internals.js` 脚本中调用了 `getContext('2d')` API，代码如下：

 复制代码

```
1 ...
2 var context = this.canvas_.getContext('2d'); // 获得 canvas 上下文
3 context.fillStyle = BACKGROUND_COLOR; // 设置填充颜色
4 context.fillRect(0, 0, width, height); // 设置填充区域
5 ...
```

上面的这段代码，首先通过 Canvas 获得它的上下文，然后通过 Canvas 上下文来设置 Canvas 的背景颜色和区域大小。通过这段代码，我们还可以得出一个结论：**WebRTC 中**

的各种统计图表都是通过 Canvas 来绘制的。

了解了上面的这些信息后，下面我们就来分析一下 WebRTC 是如何通过 Canvas 来绘制图表的。

使用 Canvas 绘制图形

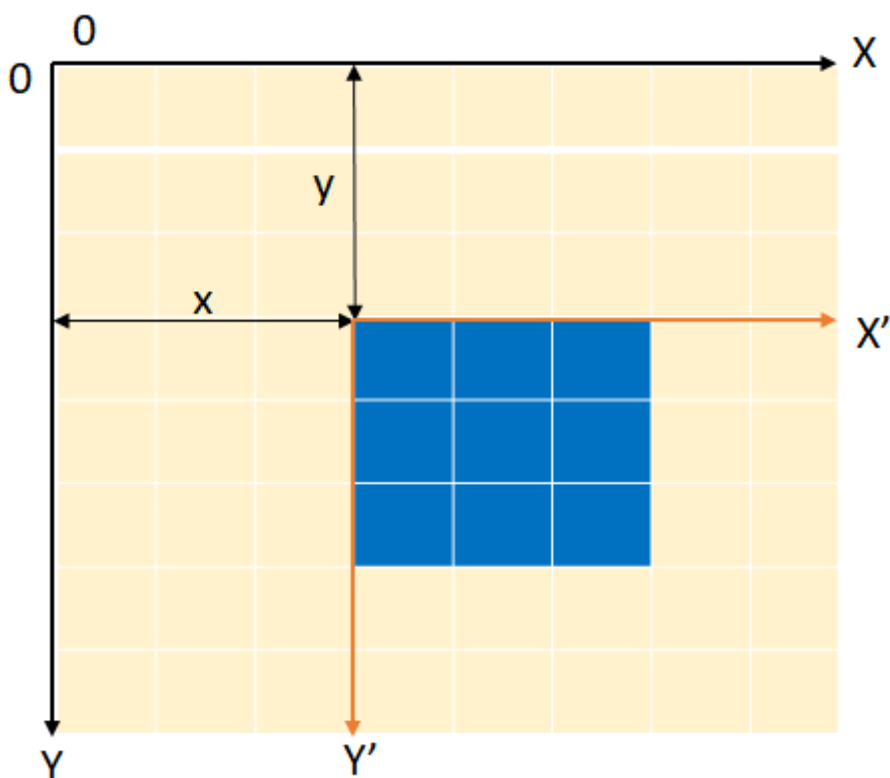
Canvas 可以绘制矩形、路径、圆弧等基本几何图形，通过这些基本图形的组合，可以绘制出其他更加复杂的图形。

除了绘制各种图形外，Canvas 还可以对图形进行颜色填充和边框涂色。而对图形的操作，如旋转、伸缩、位置变换等也是 Canvas 必备的功能。

下面我们就来学习一下如何通过 Canvas 来绘制统计图表。

1. 坐标系

通过上面的描述，我想你应该已经知道了 Canvas 的功能还是非常强大的，通过它可以绘制各种复杂的图形。不过在使用 Canvas 绘制各种复杂的图形之前，你必须要先了解 Canvas 的坐标系。只有对 Canvas 的坐标系了解清楚了，你才能做后面的事儿。




Canvas 坐标系图

Canvas 坐标系的原点在画布的左上角，X 坐标从左向右增长，Y 坐标是从上到下增长。因为 Canvas 绘制的是像素图，所以你可以把上图坐标系中的小方格理解为一个像素。

另外，Canvas 坐标系是可以改变的。你既可以从坐标 (0,0) 点来绘图，也可以改成相对 (x,y) 点来绘图，这是通过接口 `translate(x,y)` 来实现的。也就是说，当你想改变坐标系的原点时，可以通过调用 `translate(x,y)` 这个 API 将原点设置为 (x,y) 点，这样你后面所有图形的绘制都是基于 (x,y) 这个坐标点的相对坐标了。坐标原点变换，主要是为了处理上的方便，它往往需要配合 `save()/restore()` 等 API 来完成。

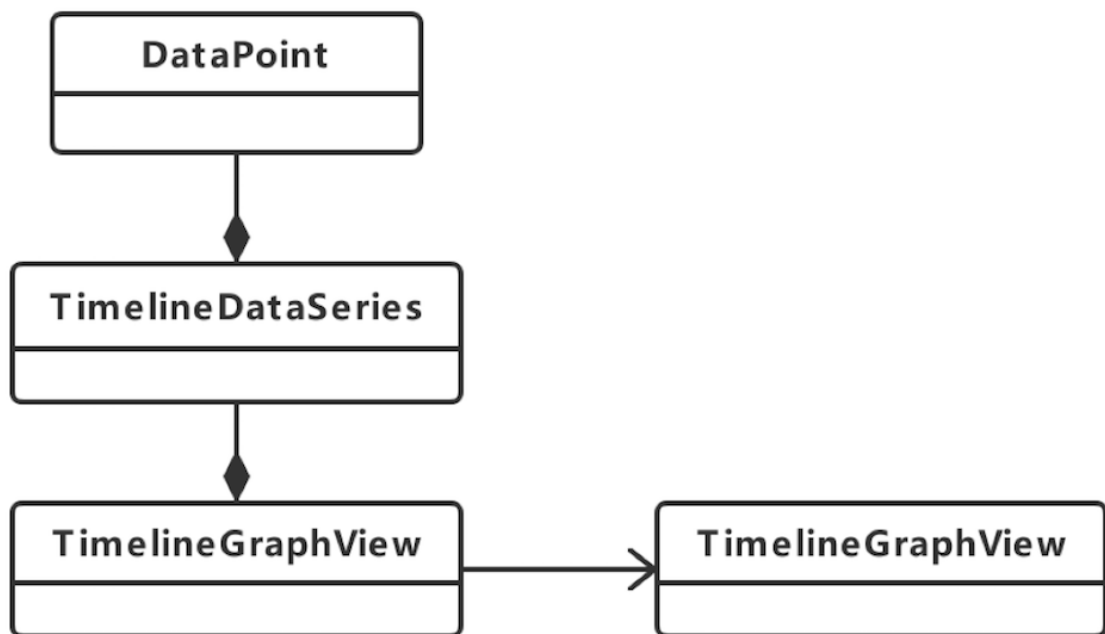
2. 绘制 WebRTC 图表

关于绘制 WebRTC 统计图表的实现，可以参考在 GitHub 上的[Chrome 源码](#)的镜像，在这个源码中，与绘图相关的代码主要是以下三个文件：

 复制代码

```
1 content/browser/resources/media/timeline_graph_view.js
2 content/browser/resources/media/data_series.js
3 content/browser/resources/media/stats_graph_helper.js
```

接下来，我们就对这个实现做一下简单的分析，由于功能不是很复杂，所以涉及的类也不多，通过下面这张类图就可以表示得很清楚：



绘制 WebRTC 统计图表的类图

类名称	功能说明
DataPoint	代表图形中的一个点，包含数值和时间。时间是 Unix epoch 时间，单位是毫秒。
TimelineDataSeries	存储统计数据，如时间、颜色信息等，最终形成一个数据集。
TimelineGraphView	代表一个统计图，在一张统计图可以画多条统计线。
Graph	它是一个绘图功能类，做具体的绘制工作。

下面我们就对上面这几个类之间的逻辑关系做一下梳理：

当我们在浏览器的 tab 页中输入 `chrome://webrtc-internals/` 打开 WebRTC 的统计页面后，该页面首先会通过 WebRTC 的 `RTCPeerConnection` 对象的 `getStats` API 获取 WebRTC 的各种统计数据，然后按照不同的分类，将数据转成一个的 **DataPoint**。

`chrome://webrtc-internals/` 会启动一个定时器，每秒触发一次，这样就可以每秒调用一次 `getStats` 方法获取统计数据，然后生成各种 **DataPoint** 对象。随着时间的推移，这些 **DataPoint** 就可以连成一条线了。

每种 **DataPoint** 都被保存到 **TimelineDataSeries** 对象中。该对象是按时间排序的数据集。当然，在该数据集中所有的 **DataPoint** 都属于一类，举个例子，每秒钟发送端带宽属于一类，每秒钟发送的数据包个数属于另一类。

TimelineGraphView对象表示的是一个以时间为主线的图表，在这个图表中可以绘制多条曲线。实际上我们可以将每秒钟发送端的带宽与每秒钟发送的数据包数放在同一个 **TimelineGraphView** 中显示。但在我们的例子中，每个图表只绘制了一种曲线。

数据准备好后，最终图形的绘制是通过 **Graph** 对象完成的，它才是在 **Canvas** 中绘制图形真正的“负责人”。

下面咱们再从代码的角度来看一下它们具体是如何实现的吧！


首先是 DataPoint 类，它非常简单，定义如下：

 复制代码

```
1 function DataPoint(time, value) {  
2     this.time = time; // 数据产生的时间，以毫秒为单位  
3     this.value = value; // 数值  
4 }
```

这个类没有什么需要特别讲的，就是记录了一个**数据产生的时间和数值**。

其次是 TimelineDataSeries 类，你可以结合下面的示例代码来看看它都包括哪些内容。

 复制代码

```
1 ...  
2 // 按时间顺序的列表  
3 this.dataPoints_ = [];  
4 // 画点的默认颜色  
5 this.color_ = 'red';  
6 // 是否真正的绘制图形  
7 this.isVisible_ = true;  
8 ...
```

在该类中主要包括上面三个属性：

`dataPoints_`：用于存放 `DataPoint` 类型数据的数组，在实际场景中就是每秒生成的 `DataPoint` 的数组。

`color_`：表示在绘制时以什么颜色画点。

`isVisible_`：指示数据是否真的要在图表中展示出来。

通过以上三个属性，你就可以非常清楚地知道这个类的作用是存放 `DataPoint` 类型数据的集合，并指明了画点时的颜色。

再次是 `TimelineGraphView` 类，它的重要属性如下所示：

 复制代码

```
1 ...
2 // 获得 canvas
3 this.canvas_ = document.getElementById(canvasId);
4
5 // 开始时间
6 this.startTime_ = 0;
7 // 终止时间
8 this.endTime_ = 1;
9 //graph 用于绘图
10 this.graph_ = null;
11
12 // 横向刻度，每毫秒一个像素，默认刻度是一秒
13 this.scale_ = 1000;
14
15 // 初始化开启滚动条
16 this.updateScrollbarRange_(true);
17 ...
```

从上面的代码中，你可以看到 `TimelineGraphView` 类可以获得 `Canvas` 元素，并保存了要绘制图形的开始时间和结束时间等信息。

其中还有一个特别关键的属性，即 `graph_` 属性，`TimelineGraphView` 对象最终会使用 `Graph` 对象绘制图形。


最后一个是 `Graph` 类，它的重要属性定义如下：

 复制代码

```
1 ...
2 //TimelineDataSeries 对象数组
3 this.dataSeries_ = [];
4
5 // Cached properties of the graph, set in layout.
6 this.width_ = 0; // 图表的宽度
7 this.height_ = 0; // 图表的高度
8 ...
```

通过上面的属性你可以知道，Graph 对象里存放了 TimelineDataSeries 对象数组，也就是说可以在一张图表中绘制多条不同的曲线。除此之外，Graph 中还定义了图表的宽度和高度。

那 Graph 是如何通过 Canvas 绘制图形的呢？我们来看一下它的具体实现，代码如下：

 复制代码

```
1 repaint: function () {
2
3   ...
4
5   let width = this.canvas_.width;
6   let height = this.canvas_.height;
7   let context = this.canvas_.getContext('2d');
8
9   // 清空 Canvas
10  context.fillStyle = BACKGROUND_COLOR;
11  context.fillRect(0, 0, width, height);
12
13  ...
14
15  if (this.graph_) {
16    ...
17
18    // 画线
19    this.graph_.drawLines(context);
20    ...
21  }
22  ...
23 }
24
25 drawLines: function (context) {
26   ...
27   // 遍历每个 TimelineDataSeries 对象，实际只有一个
28   for (let i = this.dataSeries_.length - 1; i >= 0; --i) {
29     // 从 TimelineDataSeries 对象中取出值
```

```
30     let values = this.getValues(this.dataSeries_[i]);
31     if (!values) {
32         continue;
33     }
34     // 取出画线的颜色
35     context.strokeStyle = this.dataSeries_[i].getColor();
36     context.beginPath();
37     for (let x = 0; x < values.length; ++x) {
38         // 连线
39         context.lineTo(
40             x, bottom - Math.round((values[x] - this.min_) * scale));
41     }
42     context.stroke();
43 }
44 }
```

在 Graph 中首先调用 repaint 函数进行绘图，在该函数中通过 Canvas 元素获取到了 Canvas 的上下文，然后通过该上下文将 Canvas 清空，最后调用 drawLines 函数进行连线。

在 drawLines 函数中，它从 dataSeries 数组中取 DataPoint 数据，之后调用 context.lineTo 方法将所有的点进行连接，这样就可以将 WebRTC 的统计信息通过图表的方式展示出来了。

通过上面的分析，你应该可以看出使用 HTML5 中的 Canvas 标签来绘制图表还是蛮简单的！

总结

本文通过一个例子，向你讲述了 WebRTC 的统计信息图表是如何通过 Canvas 来实现的。还分析了在 Chrome 浏览器中的 <chrome://webrtc-internals> 统计页面中各种统计信息图表实现的基本逻辑。

另外，你还需要知道，Chrome 浏览器中提供的统计信息的功能非常强大，除了本文介绍的可视化图表外，更重要的是你需要理解统计数据含义，这样才更有利于分析产品中遇到的问题。对于这些数据的含义我在之前的文章中已经向你做了介绍，不清楚的同学可以再回顾一下之前的内容。

所以，你如果有时间也有兴趣的话，可以对 WebRTC 做进一步了解，你也可参考本文提供的 Chrome 源码部分以及相关资料，做进一步学习。

思考时间

我们通过 Canvas 绘制曲线的时候，经常会出现锯齿形，这种锯齿非常难看，有什么办法可以让曲线更平滑一些吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。



从 0 打造音视频直播系统

手把手教你打造实时互动音视频直播系统

李超

新东方音视频直播技术专家
前沪江音视频架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | 如何使用Canvas绘制统计图表（上）？

精选留言 (1)

写留言



许童童

2019-08-24

可以手动开启抗锯齿：

```
canvas.getContext('2d').imageSmoothingEnabled = true;
```

展开 ∨

