электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 03, март 2016

УДК 004.424.7

Сравнительный анализ способов сетевой передачи изображений в HTML5 Canvas

***Дунин И.В.**, бакалавр*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю.Е., к.т.н., доцент,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»
gapyu@bmstu.ru

Технология Canvas - одна из составных частей стандарта HTML5. Она предназначена для создания двумерных изображений средствами скриптовых языков браузеров, в первую очередь, средствами Javascript. На основе этой технологии может создаваться разнообразный мультимедийный контент для веб-приложений: графики, анимации, видео. Также Canvas широко применяется в браузерных играх. Предоставляемое API открывает широкие возможности в работе с геометрическими объектами, поэтому одним из возможных применений Canvas являются графические редакторы, в том числе многопользовательские. Для приложений такого рода особую роль играет высокая степень синхронизации, которая напрямую зависит от способа обработки изображения для передачи по сети. Ниже будут рассмотрены некоторые методы передачи в контексте задачи графического редактора.

С точки зрения разметки страницы, Canvas-элемент представляет собой тег, который отображается браузером как прозрачное поле фиксированных размеров. Стандартными значениями являются 300px и 150px в ширину и высоту соответственно, но API предоставляет возможность установки любых значений средствами Javascript. В работе будут использоваться значения 500px на 400px. Пример установки значений:

HTML:

```
<canvas class='js-canvas'></canvas>
```

Javascript (jQuery):

```
var canvas = this.$('.js-canvas');
```

```
canvas.get(0).width = 500;
canvas.get(0).height = 400;
```

В рассматриваемом примере два пользователя с запущенными на связанных по сети клиентских машинах приложениями должны видеть максимально синхронное изменение изображения на Canvas-элементах, в случае, если один из них совершает на нем какое-либо изменение. Рассмотрим следующую ситуацию, продемонстрированную на рис.1: пользователь на чистом поле редактора рисует вертикальную линию толщиной 25px.

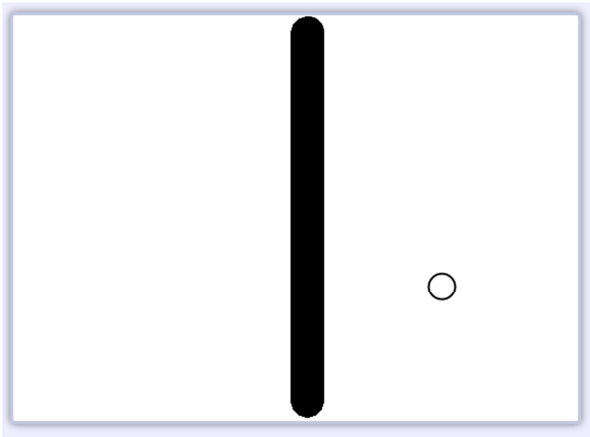

Рис. 1. Поле редактора с нарисованной линией (вид в браузере)

При этом процесс рисования линии должен наблюдаться вторым пользователем на его Canvas-элементе. Очевидно, что при рисовании такой линии на поле появилось около 25*400 = 10 000 точек (за вычетом точек скруглениях линии).

Метод getImageData()

Наиболее очевидным способом обмена является пересылка изображения целиком после каждого совершенного изменения. Для этого потребуется провести сериализацию изображения в массив. Для этой цели используется метод getImageData(), входящий в состав API. Он сохраняет данные с поля в формате по 4 значения на 1 пиксель (цветовая модель RGB с альфа-каналом). Его результатом будет Javascript-объект с полем data, которое является массивом, в который записываются поочередно 4 значения для каждого пикселя поля. На принимающей стороне изображение можно восстановить с помощью обратного метода putImageData().

```
var canvas = this.$('.js-canvas');
var context = canvas.get(0).getContext('2d');
```

```
var imgData = context.getImageData();
```

Таким образом, в imgData[0], imgData[1], imgData[2], ImgData[3] записаны соответственно три значения для схемы RGB и одно для альфа-канала крайнего пикселя изображения. Значения находятся в диапазонах 0 … 255, т. е. элемент массива занимает 1 байт. Для рассматриваемой задачи с полем 500 на 400, размер получаемого массива данных равен 80 Кбайт. Если мышь совершит перемещение на 400 пикселей, потребуется переслать 80 * 400 = 32 000 Кбайт = 32 Мбайт. Следует отметить, что этот размер является верхней границей. Пересылка должна осуществляться при изменении положения мыши в коде обработчика соответствующего события.

```
 this.$('.js-canvas').on('mousemove', function (event) {
     var context = event.target.getContext('2d');
     var imgData = context.getImageData();
     // ...
     // действия по пересылке
});
```

Вызов обработчика не гарантируется при каждом перемещении, так как мышь передает процессору свое новое местоположение с некоторой ограниченной частотой, зависящей от аппаратной реализации (как правило это число не превышает 100 раз в секунду). Если мышь пересечет за секунду более 100 пикселей, не все ее положения будут зафиксированы. Соответственно, чем медленнее перемещается мышь при рисовании линии, тем более гладкой получится линия, тем чаще будут отсылаться данные и, соответственно, тем больше будет трафик.

Метод getDataUrl()

Следующим возможным способом пересылки является использование метода toDataUrl(), который в соответствии с названием возвращает данные изображения в схеме data: URL. Метод позволяет сохранять содержимое Canvas-элемента как файл в формате png или jpeg с возможным выбором качества изображения, кодируя его в Base64 с разрешением 96dpi. Параметрами метода являются формат изображения («image/jpeg» или «image/png», причем второе является значением по умолчанию) и числовое значение от 0 до 1, указывающее качество изображения. Второй параметр учитывается только при сохранении в формате jpeg. Примеры получения данных в формате data: URL:

```
var jpegUrl = canvas.toDataURL("image/jpeg");
```

```
var pngUrl = canvas.toDataURL();
```

Полученные данные отправляются на сервер, который может сохранить их в виде файла или без сохранения отправить другому клиенту. Восстановление изображения на Canvas-элементе может выглядеть следующим образом.

```
var canvas = this.$('.js-canvas');
var context = canvas.get(0).getContext('2d');
var image = new Image();
image.onload = function() {
    context.drawImage(image, 0, 0);
};
image.src = "...";
// изображение, заданное через data: URL
```

Размеры изображений, получаемых путем применения метода toDataUrl значительно меньше, чем у массива, генерируемого методом toImageData. В случае применения формата jpeg выигрыш обеспечивается встроенными технологиями сжатия, свойственными этому формату. Сохраненный файл в формате png может иметь меньший размер за счет использования альфа-канала, что особо актуально для такого типа приложений, как графические редакторы. Так, файл, представляющий собой вертикальную линию на пустом поле, показанный на рис. 2, занимает около 2 Кбайт (в сравнении с 80 Кбайт у массива ImageData). Разумеется, размер файла будет расти по мере заполнения поля, но, тем не менее, этот метод очевидно более эффективен по затратам памяти и трафика.

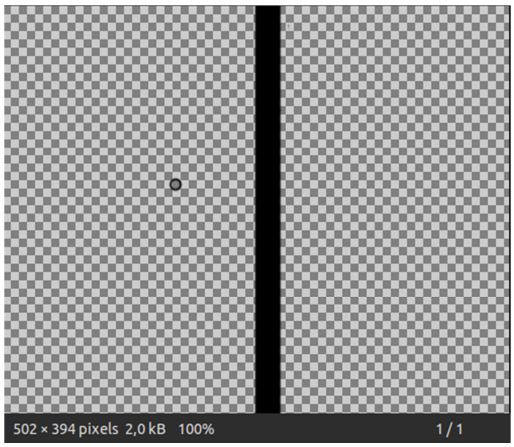

Рис. 2. Поле редактора, сохраненное как PNG-изображение

Обмен координатами

В обоих описанных выше методах для обмена информацией между участниками использовалась передача картинки целиком. Объемы передаваемой информации можно значительно уменьшить, если отказаться от восстановления состояния поля по картинке в пользу применения методов API, предназначенных для рисования объектов. Для отображения лини на Canvas-элементе достаточно использовать следующую комбинацию методов: beginPath() и closePath() для начала и завершения изображения линии, moveTo(x,y) для перемещения курсора, lineTo() и stroke() для отрисовки линии. Таким образом, для изменения изображения на Canvas-элементах клиента достаточно передавать 6 параметров: координаты (две пары чисел), толщину и цвет рисуемой линии. Используем для передачи формат JSON. Javascript-объект с необходимыми данными имеет следующий вид:

```
{
x0: 100,
y0: 200,
x: 300,
y: 400,
width: 25,
color: '#FFFFFF'
}
```

При сериализации мы получим JSON-строку вида:

"{"x0":100,"y0":200,"x":300,"y":400,"width":500,"color":"#FFFFFF"}"

При кодировании в UTF-8 каждый символ потребует 1 байт. В данной строке 67 символов. Таким образом, на каждый запрос, отправляемый при изменении изображения на Canvas-элементе будет пересылаться 67 байт, вместо нескольких килобайт при использовании сериализации методом toDataUrl() и нескольких десятков килобайт при сериализации методом getImageData().

Ниже представлен пример метода, принимающего такую строку как параметр и отображающего на Canvas-элементе линию от (x0, y0) до (x,y), цвета color и толщины width.

```
onNewCurve: function (data) {
     var curve = JSON.parse(data); // десериализация JSON-строки
     this.context.beginPath();
```

```
    this.context.moveTo(curve.x0, curve.y0);
    this.context.strokeStyle = curve.color;
    this.context.lineWidth = curve.width;
    this.context.lineTo(curve.x, curve.y);
    this.context.stroke();
    this.context.closePath();
}
```

При необходимости этот метод можно дополнительно оптимизровать, к примеру, не включая в отправляемый запрос цвет и толщину линий. Так как цвет и толщина меняются со значительно меньшей частотой, чем происходит перемещение мыши, их изменения можно передавать в отдельных запросах.

В случае если не требуется производить изменения в реальном времени, можно передавать координаты не отдельными запросами, а группами, что позволит сэкономить на передаче системной информации, в частности, на передаче HTTP заголовков.

Заключение

Проведенный сравнительный анализ методов сетевой передачи изображений между объектами HTML5 Canvas позволил определить наиболее эффективный среди них. Метод передачи координат новых точек и восстановление изображения с помощью геометрических методов API технологии Canvas позволяет получить многократный выигрыш по сравнению с передачей изображений в других форматах.

Список литературы

[1]. Fulton S., Fulton J. HTML5 Canvas. Sebastopol: O'Reilly Media, Inc., 2011, 321 p.

[2]. Таненбаум Э., Остин Т., Архитектура компьютера. М.: Питер, 2015. 816 с.

[3]. Официальный портал консорциума W3C. Режим доступа: http://www.w3.org/ (дата обращения 13.09.2015).

[4]. Gapanyuk Yu., Lakomkin E., Ionkin S., Davtyan M. MVC WEB Framework based on eXist application server and XRX architecture // 7th Spring Researchers Colloquium on Databases and Information Systems: proceedings. Available at: http://ceur-ws.org/Vol-735/paper4.pdf, accessed 13.09.2015.