

Раздел

Бинарные данные и файлы

Навигация по уроку

Blob как URL

Blob to base64

Изображение в Blob

Из Blob в ArrayBuffer



Итого

Комментарии

Поделиться



Редактировать на GitHub

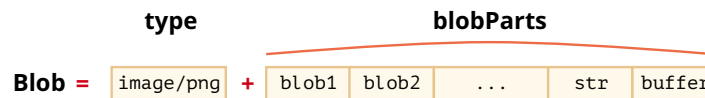
 → Бинарные данные и файлы 19-го июля 2020

Blob

`ArrayBuffer` и бинарные массивы являются частью ECMA-стандарта и, соответственно, частью JavaScript.

Кроме того, в браузере имеются дополнительные высокоуровневые объекты, описанные в [File API](#).

Объект `Blob` состоит из необязательной строки `type` (обычно MIME-тип) и `blobParts` – последовательности других объектов `Blob`, строк и `BufferSource`.



Благодаря `type` мы можем загружать и скачивать `Blob`-объекты, где `type` естественно становится `Content-Type` в сетевых запросах.

Конструктор имеет следующий синтаксис:

```
1 new Blob(blobParts, options);
```

- **blobParts** – массив значений `Blob` / `BufferSource` / `String`.
- **options** – необязательный объект с дополнительными настройками:
 - **type** – тип объекта, обычно MIME-тип, например. `image/png`,
 - **endings** – если указан, то окончания строк создаваемого `Blob` будут изменены в соответствии с текущей операционной системой (`\r\n` или `\n`). По умолчанию `"transparent"` (ничего не делать), но также может быть `"native"` (изменять).

Например:

```
1 // создадим Blob из строки
2 let blob = new Blob(["<html>...</html>"], {type: 'text/html'});
3 // обратите внимание: первый аргумент должен быть массивом
```

```
1 // создадим Blob из типизированного массива и строк
2 let hello = new Uint8Array([72, 101, 108, 108, 111]);
3
4 let blob = new Blob([hello, ' ', 'world'], {type: 'text/html'})
```

Мы можем получить срез `Blob`, используя:

```
1 blob.slice([byteStart], [byteEnd], [contentType]);
```

- **byteStart** – стартовая позиция байта, по умолчанию 0.
- **byteEnd** – последний байт, по умолчанию до конца.
- **contentType** – тип `type` создаваемого `Blob`-объекта, по умолчанию такой же, как и исходный.

Аргументы – как в `array.slice`, отрицательные числа также разрешены.

Раздел

Бинарные данные и файлы

Навигация по уроку

Blob как URL

Blob to base64

Изображение в Blob

Из Blob в ArrayBuffer

Итого

Комментарии

Поделиться



Редактировать на GitHub



Blob не изменяем (immutable)

Мы не можем изменять данные напрямую в Blob, но мы можем делать срезы и создавать новый Blob на их основе, объединять несколько объектов в новый и так далее.

Это поведение аналогично JavaScript-строке: мы не можем изменить символы в строке, но мы можем создать новую исправленную строку на базе имеющейся.

Blob как URL

Blob может быть использован как URL для `<a>`, `` или других тегов, для показа содержимого.

Давайте начнём с простого примера. При клике на ссылку мы загружаем динамически генерируемый Blob с `hello world` содержимым как файл:

```
1 <!-- download атрибут указывает браузеру делать загрузку
2 <a download="hello.txt" href="#" id="link">Загрузить</a>
3
4 <script>
5   let blob = new Blob(["Hello, world!"], {type: 'text/pla
6
7   link.href = URL.createObjectURL(blob);
8 </script>
```

Мы также можем создать ссылку динамически, используя только JavaScript, и эмулировать на ней клик, используя `link.click()`, тогда загрузка начнётся автоматически.

Далее простой пример создания «на лету» и загрузки Blob-объекта, без использования HTML:

```
1 let link = document.createElement('a');
2 link.download = 'hello.txt';
3
4 let blob = new Blob(['Hello, world!'], {type: 'text/pla
5
6 link.href = URL.createObjectURL(blob);
7
8 link.click();
9
10 URL.revokeObjectURL(link.href);
```

URL.createObjectURL берёт Blob и создаёт уникальный URL для него в формате `blob:<origin>/<uuid>`.

Вот как выглядит сгенерированный URL:

```
1 blob:https://javascript.info/1e67e00e-860d-40a5-89ae-6a
```

Браузер для каждого URL, сгенерированного через `URL.createObjectURL`, сохраняет внутреннее соответствие URL → Blob. Таким образом, такие URL короткие, но дают доступ к большому объекту Blob.

Сгенерированный url действителен, только пока текущий документ открыт. Это позволяет ссылаться на сгенерированный в нём Blob в ``, `<a>` или в любом другом объекте, где ожидается url в качестве одного из параметров.

В данном случае возможен побочный эффект. Пока в карте соответствия существует ссылка на Blob, он находится в памяти. Браузер не может освободить память, занятую Blob-объектом.

Ссылка в карте соответствия автоматически удаляется при выгрузке документа, после этого также освобождается память. Но если приложение

Раздел

Бинарные данные и файлы

Навигация по уроку

Blob как URL

Blob to base64

Изображение в Blob

Из Blob в ArrayBuffer

Итого

Комментарии

Поделиться



Редактировать на GitHub



имеет длительный жизненный цикл, это может произойти не скоро. Таким образом, если мы создадим URL для Blob, он будет висеть в памяти, даже если в нём нет больше необходимости.

`URL.revokeObjectURL(url)` удаляет внутреннюю ссылку на объект, что позволяет удалить его (если нет другой ссылки) сборщику мусора, и память будет освобождена.

В последнем примере мы использовали Blob только единожды, для мгновенной загрузки, после мы сразу же вызвали `URL.revokeObjectURL(link.href)`.

В предыдущем примере с кликабельной HTML-ссылкой мы не вызывали `URL.revokeObjectURL(link.href)`, потому что это сделало бы ссылку недействительной. После удаления внутренней ссылки на Blob, URL больше не будет работать.

Blob to base64

Альтернатива `URL.createObjectURL` – конвертация Blob-объекта в строку с кодировкой base64.

Эта кодировка представляет двоичные данные в виде строки с безопасными для чтения символами в ASCII-кодах от 0 до 64. И что более важно – мы можем использовать эту кодировку для «data-urls».

`data url` имеет форму `data:[<mediatype>][;base64],<data>`. Мы можем использовать такой url где угодно наряду с «обычным» url.

Например, смайлик:

```
1 `:

1. Для отрисовки изображения (или его части) на холсте (canvas) используется `canvas.drawImage`.
2. Вызов canvas-метода `.toBlob(callback, format, quality)` создаёт Blob и вызывает функцию `callback` при завершении.

В примере ниже изображение просто копируется, но мы можем взять его часть или трансформировать его на canvas перед созданием Blob:

```
1 // берём любое изображение
2 let img = document.querySelector('img');
3
4 // создаём <canvas> того же размера
5 let canvas = document.createElement('canvas');
6 canvas.width = img.clientWidth;
7 canvas.height = img.clientHeight;
8
9 let context = canvas.getContext('2d');
10
11 // копируем изображение в canvas (метод позволяет выре
12 context.drawImage(img, 0, 0);
13 // мы можем вращать изображение при помощи context.rota
14
15 // toBlob является асинхронной операцией, для которой с
16 canvas.toBlob(function(blob) {
17 // после того, как Blob создан, загружаем его
18 let link = document.createElement('a');
19 link.download = 'example.png';
20
21 link.href = URL.createObjectURL(blob);
22 link.click();
23
24 // удаляем внутреннюю ссылку на Blob, что позволит бр
25 URL.revokeObjectURL(link.href);
26 }, 'image/png');
```

Или если вы предпочитаете `async/await` вместо колбэка:

```
1 let blob = await new Promise(resolve => canvasElem.toBl
```

Для создания скриншота страницы мы можем использовать такую библиотеку, как <https://github.com/niklasvh/html2canvas>. Всё, что она делает, это просто проходит страницу и отрисовывает её в `<canvas>`. После этого мы можем получить Blob одним из вышеуказанных способов.

## Из Blob в ArrayBuffer

Конструктор `Blob` позволяет создать Blob-объект практически из чего угодно, включая `BufferSource`.

Но если нам нужна производительная низкоуровневая обработка, мы можем использовать `ArrayBuffer` из `FileReader`:

```
1 // получаем arrayBuffer из Blob
2 let fileReader = new FileReader();
3
4 fileReader.readAsArrayBuffer(blob);
5
6 fileReader.onload = function(event) {
7 let arrayBuffer = fileReader.result;
8 };
```

Раздел

[Бинарные данные и файлы](#)

Навигация по уроку

[Blob как URL](#)

[Blob to base64](#)

[Изображение в Blob](#)

[Из Blob в ArrayBuffer](#)

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



## Итого

В то время как `ArrayBuffer`, `Uint8Array` и другие `BufferSource` являются «бинарными данными», `Blob` представляет собой «бинарные данные с типом».

Это делает `Blob` удобным для операций загрузки/выгрузки данных, которые так часто используются в браузере.

Методы, которые выполняют сетевые запросы, такие как `XMLHttpRequest`, `fetch` и подобные, могут изначально работать с `Blob` так же, как и с другими объектами, представляющими двоичные данные.

Мы можем легко конвертировать `Blob` в низкоуровневые бинарные типы данных и обратно:

- Мы можем создать `Blob` из типизированного массива, используя конструктор `new Blob(...)`.
- Мы можем обратно создать `ArrayBuffer` из `Blob`, используя `FileReader`, а затем создать его представление для низкоуровневых операций.

Проводим [курсы по JavaScript и фреймворкам](#). ✕

## Комментарии

перед тем как писать...