**二、Blob 怎么用？**

Blob 由一个可选字符串 type 和 blobParts 组成，其中， type 通常为 MIME 类型。

MIME（Multipurpose Internet Mail Extensions）多用途互联网邮件扩展类型，常见有：超文本标记语言文本 .html text/html 、PNG图像 .png image/png 、普通文本 .txt text/plain 等。

**1. 构造函数**

Blob 构造函数语法为：

const myBlob = new Blob(blobParts[, options])

**入参：**

* blobParts：它是一个由 ArrayBuffer，ArrayBufferView，Blob，DOMString 等对象构成的数组。DOMStrings 会被编码为 UTF-8。
* options ：一个可选的对象，包含以下两个属性：
  + type ：默认值为 "" ，表示将会被放入到 blob 中的数组内容的 MIME 类型。
  + endings ：默认值为 "transparent"，用于指定包含行结束符 \n 的字符串如何被写入。它是以下两个值中的一个："native"，代表行结束符会被更改为适合宿主操作系统文件系统的换行符，或者 "transparent"，代表会保持 blob 中保存的结束符不变。

**出参：** 返回一个新创建的 Blob 对象，其内容由参数中给定的数组串联组成。

**2. 属性和方法**

**2.1 属性介绍**

Blob 对象拥有 2 个属性：

* size ：只读，表示 Blob 对象中所包含的数据大小（以字节为单位）；
* type ：只读，值为字符串，表示该 Blob 对象所包含数据的 MIME 类型。若类型未知，则该属性值为空字符串。

**2.2 方法介绍**

* slice([start[, end[, contentType]]]) ：返回一个新的 Blob 对象，包含了源 Blob 对象中指定范围内的数据。
* stream()：返回一个能读取 Blob 内容的 ReadableStream 。
* text()：返回一个 Promise 对象且包含 Blob 所有内容的 UTF-8 格式的 USVString 。
* arrayBuffer()：返回一个 Promise 对象且包含 Blob 所有内容的二进制格式的 ArrayBuffer 。

注意：\*\* Blob 对象是不可改变的\*\*，但是可以进行分割，并创建出新的 Blob 对象，将它们混合到一个新的 Blob 中。类似于 JavaScript 字符串：我们无法更改字符串中的字符，但可以创建新的更正后的字符串。

**3. 简单上手**

**3.1 示例1：从字符串创建 Blob**

let myBlobParts = ['<html><h2>Hello Leo</h2></html>']; // 一个包含DOMString的数组

let myBlob = new Blob(myBlobParts, {type : 'text/html', endings: "transparent"}); // 得到 blob

console.log(myBlob.size + " bytes size");

// Output: 31 bytes size

console.log(myBlob.type + " is the type");

// Output: text/html is the type

**3.2 示例2：从类型化数组和字符串创建 Blob**

JavaScript类型化数组是一种类似数组的对象，并提供了一种用于 **访问原始二进制数据的机制** 。并且在类型数组上调用 Array.isArray() 会返回 false 。 详细可参考MDN《JavaScript 类型化数组》章节。

let hello = new Uint8Array([72, 101, 108, 108, 111]); // 二进制格式的 "hello"

let blob = new Blob([hello, ' ', 'leo'], {type: 'text/plain'});

// Output: "Hello leo"

**3.3 示例3：组装新的 Blob**

由于 Blob 对象是不可改变的，但我们可以进行分割，并组装成一个新的 Blob 对象：

let blob1 = new Blob(['<html><h2>Hello Leo</h2></html>'],

{type : 'text/html', endings: "transparent"});

let blob2 = new Blob(['<html><h2>Happy Boy!</h2></html>'],

{type : 'text/html', endings: "transparent"});

let slice1 = blob1.slice(16);

let slice2 = blob2.slice(0, 16);

await slice1.text();

// currtent slice1 value: "Leo</h2></html>"

await slice2.text();

// currtent slice2 value: "<html><h2>Happy "

let newBlob = new Blob([slice2, slice1],

{type : 'text/html', endings: "transparent"});

await newBlob.text();

// currtent newBlob value: "<html><h2>Happy Leo</h2></html>"

**三、Blob 有哪些使用场景？**

**1. 图片本地预览**

这里整理 2 种图片本地预览的方式：

1. 使用 DataURL 方式；
2. 使用 Blob URL/Object URL 方式；

<body>

<h1>1.DataURL方式：</h1>

<input type="file" accept="image/\*" onchange="selectFileForDataURL(event)">

<img id="output1">

<h1>2.Blob方式：</h1>

<input type="file" accept="image/\*" onchange="selectFileForBlob(event)">

<img id="output2">

<script>

// 1.DataURL方式：

async function selectFileForDataURL() {

const reader = new FileReader();

reader.onload = function () {

const output = document.querySelector("#output1")

output.src = reader.result;

}

reader.readAsDataURL(event.target.files[0]);

}

//2.Blob方式：

async function selectFileForBlob(){

const reader = new FileReader();

const output = document.querySelector("#output2");

const imgUrl = window.URL.createObjectURL(event.target.files[0]);

output.src = imgUrl;

reader.onload = function(event){

window.URL.revokeObjectURL(imgUrl);

}

}

</script>

</body>

上面主要介绍 Blob URL 和 Data URL 两种方式实现图片本地预览，这两个类型的区别在\*\*《五、拓展》\*\*中介绍。

**2. 图片本地预览 + 分片上传**

**实现本地预览：** 将 input 获取到的 file 对象，通过实例化 FileReader ，赋值给变量 reader ，调用reader 的 readAsDataURL 方法，将 file 对象转换为 dataURL ，然后监听 reader 的 onload 属性，获取到读取结果 result ，然后设置为图片的 src 值。 **实现分片上传：** 由于 File 是特殊类型的 Blob，可用于任意 Blob 类型的上下文，所以针对大文件传输，我们可以使用 slice 方法进行文件切割，分片上传。

<body>

<input type="file" accept="image/\*" onchange="selectFile(event)">

<button onclick="upload()">上传</button>

<img id="output">

<script>

const chunkSize = 10000;

const url = "https://httpbin.org/post";

async function selectFile(){

// 本地预览

const reader = new FileReader();

reader.onload = function(){

const output = document.querySelector("#output")

output.src = reader.result;

}

reader.readAsDataURL(event.target.files[0]);

// 分片上传

await upload(event.target.files[0]);

}

async function upload(files){

const file = files;

for(let start = 0; start < file.size; start += chunkSize){

const chunk = file.slice(start, start + chunkSize + 1);

const fd = new FormData();

fd.append("data", chunk);

await fetch(url, { method: "post", body: fd }).then((res) =>{

console.log(res)

res.text();

});

}

}

</script>

</body>

**3. 图片本地预览 + 分片上传 + 暂停 + 续传**

<body>

<input type="file" accept="image/\*" onchange="selectFile(event)">

<button onclick="upload()">上传</button>

<button onclick="pause()">暂停</button>

<button onclick="continues()">继续</button>

<img id="output" src="" alt="">

<script>

const chunkSize = 30000;

let start = 0, curFile, isPause = false;

const url = "https://httpbin.org/post";

async function selectFile(){

const reader = new FileReader();

reader.onload = function(){

const output = document.querySelector("#output")

output.src = reader.result;

}

reader.readAsDataURL(event.target.files[0]);

curFile = event.target.files[0];

}

async function upload(){

const file = curFile;

for(start; start < file.size; start += chunkSize){

if(isPause) return;

const chunk = file.slice(start, start + chunkSize + 1);

const fd = new FormData();

fd.append("data", chunk);

await fetch(url, { method: "post", body: fd }).then((res) =>{

res.text()

}

);

if(chunk.size < chunkSize){

uploadSuccess();

return;

}

}

}

function pause(){

isPause = true;

}

function continues(){

isPause = false;

upload();

}

function uploadSuccess(){

isPause = false;

start = 0;

}

</script>

</body>

**4. 从互联网下载数据**

在实现“从互联网下载数据”方法时，我们使用 createObjectURL 显示图片，在请求互联网图片时，我们有两种方式：

* 使用 XMLHttpRequest ；
* 使用 fetch ；

<body>

<button onclick="download1()">使用 XMLHttpRequest 下载</button>

<button onclick="download2()">使用 fetch 下载</button>

<img id="pingan">

<script>

const url = "http://images.pingan8787.com/TinyCompiler/111.png";

const pingan = document.querySelector('#pingan');

function download1 (){

const xhr = new XMLHttpRequest();

xhr.open('GET', url);

xhr.responseType = 'blob';

xhr.onload = () => {

renderImage(xhr.response);

}

xhr.send(null);

}

function download2 (){

fetch(url).then(res => {

return res.blob();

}).then(myBlob => {

renderImage(myBlob);

})

}

function renderImage (data){

let objectURL = URL.createObjectURL(data);

pingan.src = objectURL;

// 根据业务需要手动调用 URL.revokeObjectURL(imgUrl)

}

</script>

</body>

**5. 下载文件**

通过调用 Blob 的构造函数来创建类型为 "text/plain" 的 Blob 对象，然后通过动态创建 a 标签来实现文件的下载。

<body>

<button onclick="download()">Blob 文件下载</button>

<script>

function download(){

const fileName= "Blob文件.txt";

const myBlob = new Blob(["一文彻底掌握 Blob Web API"], { type: "text/plain" });

downloadFun(fileName, myBlob);

}

function downloadFun(fileName, blob){

const link = document.createElement("a");

link.href = URL.createObjectURL(blob);

link.download = fileName;

link.click();

link.remove();

URL.revokeObjectURL(link.href);

}

</script>

</body>

**6. 图片压缩**

当我们希望本地图片在上传之前，先进行一定压缩，再提交，从而减少传输的数据量。 在前端我们可以使用 Canvas 提供的 toDataURL() 方法来实现，该方法接收 type 和 encoderOptions 两个可选参数：

* type 表示**图片格式**，默认为 image/png ；
* encoderOptions 表示**图片质量**，在指定图片格式为 image/jpeg 或 image/webp 的情况下，可以从 0 到 1 区间内选择图片质量。如果超出取值范围，将会使用默认值 0.92，其他参数会被忽略。

<body>

<input type="file" accept="image/\*" onchange="loadFile(event)" />

<script>

// compress.js

const MAX\_WIDTH = 800; // 图片最大宽度

// 图片压缩方法

function compress(base64, quality, mimeType) {

let canvas = document.createElement("canvas");

let img = document.createElement("img");

img.crossOrigin = "anonymous";

return new Promise((resolve, reject) => {

img.src = base64;

img.onload = () => {

let targetWidth, targetHeight;

if (img.width > MAX\_WIDTH) {

targetWidth = MAX\_WIDTH;

targetHeight = (img.height \* MAX\_WIDTH) / img.width;

} else {

targetWidth = img.width;

targetHeight = img.height;

}

canvas.width = targetWidth;

canvas.height = targetHeight;

let ctx = canvas.getContext("2d");

ctx.clearRect(0, 0, targetWidth, targetHeight); // 清除画布

ctx.drawImage(img, 0, 0, canvas.width, canvas.height);

let imageData = canvas.toDataURL(mimeType, quality / 100); // 设置图片质量

resolve(imageData);

};

});

}

// 为了进一步减少传输的数据量，我们可以把它转换为 Blob 对象

function dataUrlToBlob(base64, mimeType) {

let bytes = window.atob(base64.split(",")[1]);

let ab = new ArrayBuffer(bytes.length);

let ia = new Uint8Array(ab);

for (let i = 0; i < bytes.length; i++) {

ia[i] = bytes.charCodeAt(i);

}

return new Blob([ab], { type: mimeType });

}

// 通过 AJAX 提交到服务器

function uploadFile(url, blob) {

let formData = new FormData();

let request = new XMLHttpRequest();

formData.append("image", blob);

request.open("POST", url, true);

request.send(formData);

}

function loadFile(event) {

const reader = new FileReader();

reader.onload = async function () {

let compressedDataURL = await compress(

reader.result,

90,

"image/jpeg"

);

let compressedImageBlob = dataUrlToBlob(compressedDataURL);

uploadFile("https://httpbin.org/post", compressedImageBlob);

};

reader.readAsDataURL(event.target.files[0]);

};

</script>

</body>

其实 Canvas 对象除了提供 toDataURL() 方法之外，它还提供了一个 toBlob() 方法，该方法的语法如下：

canvas.toBlob(callback, mimeType, qualityArgument)

和 toDataURL() 方法相比，toBlob() 方法是异步的，因此多了个 callback 参数，这个 callback 回调方法默认的第一个参数就是转换好的 blob文件信息。

**7. 生成 PDF 文档**

在浏览器端，利用一些现成的开源库，比如 jsPDF，我们也可以方便地生成 PDF 文档。

<body>

<h3>客户端生成 PDF 示例</h3>

<script src="https://unpkg.com/jspdf@latest/dist/jspdf.min.js"></script>

<script>

(function generatePdf() {

const doc = new jsPDF();

doc.text("Hello semlinker!", 66, 88);

const blob = new Blob([doc.output()], { type: "application/pdf" });

blob.text().then((blobAsText) => {

console.log(blobAsText);

});

})();

</script>

</body>

其实 jsPDF 除了支持纯文本之外，它也可以生成带图片的 PDF 文档，比如：

let imgData = '...'

let doc = new jsPDF();

doc.setFontSize(40);

doc.text(35, 25, 'Paranyan loves jsPDF');

doc.addImage(imgData, 'JPEG', 15, 40, 180, 160);