JavaScript高级 - 纯js实现文件上传大文件切片上传断点续传（客户端）

西瓜watermelon
2021年08月07日 21:57 · 阅读 2909

这是我参与8月更文挑战的第7天，活动详情查看：8月更文挑战

前言

> - 在另一篇文章JavaScript高级 - nodejs+koa2实现文件上传大文件切片上传断点续传(服务器端)中已经介绍了文件上传的服务器端实现，并实现了几个不同上传形式的服务器端接口
> - 接下来我们来一起来实现一步步实现前端（客户端）的上传功能，关于html和css不再过多讲述，这里主要讲一下js代码实现。
> - 下面先来看一下效果图

一、环境准备及依赖库

- axios v0.21.1：用于调用服务端接口发送服务端请求
- spark-md5 v3.0.1：用于根据文件内容生成hash码
- qs v6.9.6：用于将application/x-www-form-urlencoded格式从参数解析为arxx&bcvy的格式

二、项目结构

web 项目根目录

- scripts 存放js脚本目录
  - axios.min.js axios库（第三方）
  - qs.js qs库（第三方）
  - spark-md5.min.js spark-md5库（第三方）
  - axios2.js axios二次封装库（自定义）。
  - upload.js 上传文件功能代码（自定义）
- css 样式文件目录
  - upload.css 页面样式
- index.html 文件上传html页面

三、功能实现

> - 结合上面的截图，将分为5个模块进行讲解，所有模块用到的上传控件都是html原生的类型为file的input控件。
> - 为了页面的美观，我们将input隐藏起来，并用普通按钮替代，当点击按钮时触发input的click事件。
> - 另外可以再额外加一些进度显示，图片缩略图显示，文件名称显示等。
> - 关于HTML和Css部分不再过多说明，下面将分模块进行js部分重点讲解，每个模块都用闭包函数包裹，避免变量污染

1、axios二次封装

> 在每个功能模块中，我们都将通过使用axios拦截器端发送请求，这时我们就需要对axios做一些特殊处理，也就是二次封装
>
> - 创建axios对象，避免不同场景配置冲突
> - 设置baseURL
> - 设置默认Content-Type 为 multipart/form-data
> - 在transformRequest中判断Content-Type，如果是application/x-www-form-urlencoded 则利用qs库对参数进行格式化

```
//axios.js axios二次封装
let request = axios.create();
request.defaults.baseURL = 'http://127.0.0.1:3000';
request.defaults.headers['Content-Type'] = 'multipart/form-data';
request.defaults.transformRequest = (data, headers) => {
    let contentType = headers['Content-Type'];
    if (contentType === 'application/x-www-form-urlencoded') return Qs.stringify(data);
    return data;
}

request.interceptors.response.use(response => {
    return response.data;
});
```

2、单文件上传FROM-DATA，先选文件再上传

- 简单步骤分析：
  - 首先应该先获取到需要用到的页面元素：上传控件input，选择按钮，上传按钮，缩略图展示，文件展示，进度条展示
  - 绑定选择按钮的click事件并在click事件中触发上传控件input的click事件
  - 绑定上传控件input的change事件，在该事件中获取已选择的文件
  - 绑定上传按钮的click事件，在该事件中组合参数并发送post请求调用服务端API实现文件上传
  - 文件上传的关键代码就是发送请求前的参数拼接部分
    - 这里我们利用js内置的FormData类将文件作为参数传输
    - new FormData().append("file", file);
- 代码实现

```
//upload.js 单文件上传FORM-data
(function () {
    let upload1 = document.querySelector("#upload1"),
        upload_inp = upload1.querySelector('.upload-inp'),
        upload_select = upload1.querySelector('.upload-btn.select'),
        upload_upload = upload1.querySelector('.upload-btn.upload'),
        sel_files = upload1.querySelector('.files'),
        file1 = upload1.querySelector('.abbr'),
        cur_pro = upload1.querySelector('.cur-pro'),
        pro_val = upload1.querySelector('.pro-val'),
        progress = upload1.querySelector('.progress'),
        _file;

    upload_select.addEventListener('click', () => {
        upload_inp.click();
    });
    upload_inp.addEventListener('change', function () {
        let file = this.files[0];
        _file = file;
        sel_files.innerHTML = file.name;
        progress.style.display = 'inline-block';
        pro_val.innerHTML = '';
    })

    upload_upload.addEventListener('click', function () {
        let formData = new FormData();
        formData.append('file', _file);
        formData.append('filename', _file.name);
        request.post('/upload_single_file', formData, {
            onUploadProgress: function (ev) {
                let pro = (ev.loaded / ev.total) * 100).toFixed(0) + '%';
                cur_pro.style.width = pro;
                pro_val.innerHTML = pro;
            }
        }).then(res => {
            console.log(res);
            file1.src = 'http://$res.serverPath';
            file1.style.display = 'block';
        }).catch(err => {
            console.log(err);
        });
    });
})();
```

3、单文件上传BASE64，只能上传小于100K的png或jpg图片文件

- 简单步骤分析：
  - 首先应该先获取到需要用到的页面元素：上传控件input，选择上传按钮，缩略图展示，文件名展示，进度条展示
  - 绑定选择并上传按钮的click事件并在click事件中触发上传控件input的click事件
  - 这里由于选择和上传按钮合并为一个，所以向服务端发送请求的步骤我们将放在input的change事件中
  - 绑定input的change事件，在该事件中组合参数并发送post请求调用服务端API实现文件上传
  - 在文件域的change事件中，我们需要
    - 根据容错的文件信息校验文件的大小不能超过100k（base64格式上传文件不能太大）**file.size属性**
    - 校验文件格式只能是png或jpg格式，**file.type属性**
    - 利用js内置的FileReader类将文件转换为BASE64格式
    - 并在FileReader的onload函数中拿到转换为BASE64格式的内容并发送post请求实现文件上传
    - 另外在因为BASE64格式的内容涉及到的字符比较多，为了避免一些特殊字符问题，在参数传递前需要用encodeURIComponent进行编码

- 代码实现

```
//upload.js 单文件Base64上传
(function () {
    let upload2 = document.querySelector("#upload2"),
        upload_inp = upload2.querySelector('.upload-inp'),
        upload_upload = upload2.querySelector('.upload-btn.upload'),
        sel_files = upload2.querySelector('.files'),
        file2 = upload2.querySelector('.abbr'),
        progress = upload2.querySelector('.progress'),
        cur_pro = upload2.querySelector('.cur-pro'),
        pro_val = upload2.querySelector('.pro-val'),
        _file;

    upload_upload.addEventListener('click', () => {
        upload_inp.click();
    });
    upload_inp.addEventListener('change', function () {
        progress.style.display = 'inline-block';
        pro_val.innerHTML = '';
        let file = this.files[0];
        _file = file;
        if (file.size > 100 * 1024) {
            alert('图片最小于100k');
            return;
        }

        if (!/(jpg|jpeg|png)/.test(file.type)) {
            alert('只能上传jpg或jpeg格式的图片');
            return;
        }
        sel_files.innerHTML = file.name;
        let fr = new FileReader();
        fr.readAsDataURL(file);
        fr.onload = ev => {
            file2.src = ev.target.result;
            file2.style.display = 'block';
            console.log(file.name);
            request.post('/upload_base64', {
                file: encodeURIComponent(ev.target.result),
                filename: file.name
            }, {
                headers: {
                    "Content-Type": "application/x-www-form-urlencoded"
                },
                onUploadProgress: function (ev) {
                    let pro = ((ev.loaded / ev.total) * 100) + '%';
                    pro_val.innerHTML = pro;
                    cur_pro.style.width = pro;
                }
            }).then(res => {
                console.log(res);
                alert('上传成功了');
                return;
            }).catch(err => {
                console.log(err);
                alert('失败了？');
            });
        };

    })
})();
```

4、多文件上传FORM-DATA

- 简单步骤分析：
  - 多文件上传的步骤跟单文件FORM-DATA的步骤类似
  - 首先也是先获取到需要用到的页面元素：上传控件input，选择上传按钮
  - 绑定选择按钮的click事件并在click事件中触发上传控件input的click事件
  - 绑定上传控件input的change事件，在该事件中获取已选择的文件，并遍历所选的文件组合参数发送post请求调用服务端API实现文件上传
  - 与单文件FORM-DATA上传不同的是，我们获取到的是一个文件列表
    - 在change事件中需要遍历所有所选的文件，组合参数然后循环发送请求调用服务器API接口实现多文件上传
    - 因为是不固定个数文件上传，所以在遍历所选文件后需要动态将文件名和上传进度拼接为HTML元素展示在页面上
- 代码实现

```
// upload.js 多文件上传FORM-data
(function () {
    let upload3 = document.querySelector("#upload3"),
```

```
    upload_inp = upload3.querySelector('.upload-inp');
    upload_upload = upload3.querySelector('.upload-btn');
    set_files = upload3.querySelector('.list');


    upload_upload.addEventListener('click', () => {
        upload_inp.click();
    });
    upload_inp.addEventListener('change', function () {
        let files = this.files;
        set_files.innerHTML = '';
        [].forEach.call(files, (file, index) => {
            set_files.innerHTML += `<div><span class="files" style="margin-right:8px;font-size:12px">${file.name}</span><span class="pro-val" id="myfiles${index}">%</span></div>`
            let formData = new FormData();
            formData.append('file', file);
            formData.append('filename', file.name);
            request.post('/upload_single_file', formData, {
                onUploadProgress: function (ev) {
                    let pro = (ev.loaded / ev.total) * 100).toFixed(0) + '%';
                    document.querySelector(`#myfiles${index}`).innerHTML = pro;
                    // set_files.innerHTML += `<span class="file">${file.name}</span> <span class="pro-val" id=pro${index}></span>`
                }
            }).then(res => {
                console.log(res);
                // alert('上传成功?');
            }).catch(err => {
                console.log(err);
            });
        });
    });
});();
```

5、多文件拖拽上传FORM-DATA

- **简单步骤分析：**

> 在该模块中，实现拖拽上传的重点就是在**`==drapleave==`**上，而其它上传步骤跟上面的多文件上传差一样的。
> 除了要实现拖拽上传，原来的点击上传按钮也需要保留
> 所以首先我们还是要获取到上传控件input，div块作为拖拽区域和触发点击事件的区域
> 然后把上面的上传逻辑封装为一个单独的方法uploadFiles，接收一个数组或类数组类型的参数，以方便点击或拖拽时调用
> 点击上传：
> - 这里我们要绑定拖拽域（div）绑定一个click事件，并在该事件中触发input的click事件
> - 绑定input的change事件，在该事件中调用获取到选择的文件列表，并调用上传方法uploadFiles实现文件上传
>
> 拖拽上传：
> - 要实现拖拽上传需要借助两个事件，分别是dragover和drop，就是将文件拖拽到拖拽域（div）上或是落在拖拽域（div）上时要触发的事件
> - 需要注意的是：如果将一个文件拖拽到网页上时，有些格式的文件（如：txt、jpg、png等）会被默认打开直接显示在网页上，所以我们需要在这两个事件中阻止浏览器的默认行为：ev.preventDefault()
> - 然后在drop事件中也就是当我们将文件拖拽到div上并松开鼠标时，调用uploadFiles将文件上传
>
> 在drop事件中有个dataTransfer属性的files属性可以获取到拖拽进来的文件列表，这样就可以通过ev.dataTransfer.files获取到所有的多文件了

- **代码实现**

```
//upload.js 拖拽上传form-data
(function () {
    let upload5 = document.querySelector("#upload5"),
        upload_inp = upload5.querySelector('.upload-inp'),
        upload_upload = upload5.querySelector('.upload-btn'),
        set_files = upload5.querySelector('.list');

    const uploadFiles = function uploadFiles(files) {
        set_files.innerHTML = '';
        [].forEach.call(files, (file, index) => {
            set_files.innerHTML += `<div><span class="files" style="margin-right:8px;font-size:12px">${file.name}</span><span class="pro-val" id="myfiles${index}">%</span></div>`
            let formData = new FormData();
            formData.append('file', file);
            formData.append('filename', file.name);
            request.post('/upload_single_file', formData, {
                onUploadProgress: function (ev) {
                    let pro = (ev.loaded / ev.total) * 100).toFixed(0) + '%';
                    document.querySelector(`#myfiles${index}`).innerHTML = pro;
                }
            }).then(res => {
                console.log(res);
                // alert('上传成功?');
            }).catch(err => {
                console.log(err);
            });
        });
    }
    upload5.addEventListener('dragover', function (ev) {
        ev.preventDefault();
    });
    upload5.addEventListener('drop', (ev) => {
        ev.preventDefault();
        uploadFiles(ev.dataTransfer.files);
    });

    upload_inp.addEventListener('change', function () {
        uploadFiles(this.files);
    });
    upload5.addEventListener('click', (ev) => {
        upload_inp.click();
    });
});();
```

6、大文件切片上传，断点续传FORM-DATA

> 接下来就是本文的最后一个模块，也是最复杂和最重点的模块：大文件切片上传和断点续传，要实现切片上传和断点续传逻辑较前面几个模块都稍有些复杂。下面我们依然来一步步分析一下：

- **简单逻辑分析**

> - 切片上传顾名思义就是将一个大文件分割成多个小文件进行分别上传，待所有切片上传完成后再将它们合并成一个文件，这样就实现了一个大文件的切片上传，同时如果上传过程出现问题，下次继续上传时还能实现断点续传
> - 切片上传的关键在于上传前要将所有切片文件合并，那么合并时就需要考虑一下问题了：
> - 需要将哪些文件进行合并？
> - 找到需要合并后的文件，要放置么样的顺序进行合并？ >>首先第一问题，为了能够快速方便的找到哪些文件是需要合并的，在得切片文件上传到服务器时，我们需要在服务器端建立一个单独的临时文件夹用于保存所有的切片 >>第二个问题，为了保证合并后的文件与原文件保持一致，在切片时需要给每个切片高加一个索引，这样就能在合并时按照索引进行按顺序合并了。
> - 如果切片在上传过程中出现了问题，导致上传中断，那么下次上传时为了实现不重复上传，也就是所说的断点续传，就需要进行判断，如果文件存在则直接跳过，那么如何去判断一个文件（或切片）是否存在了呢？
> - 这时就需要用到我们前面提到的==spark-md5==库了，该库可根据文件内容生成一串hash值，只要文件内容不变那么生成出来的hash值也永远都是一样的，所以我们可以利用hash值加索引的形式进行文件切片的命名
> - 切片思路分析：
> - 要将一个文件进行切片，需要借助文件的**`==size==`属性**和==slice==**`方法`**
> - 方法一（固定个数）：将一个文件切成固定个数，比如20个，然后用size/20计算出每个切片文件的大小，再利用slice进行截取
> - 方法二（固定大小）：固定每个切片文件的大小，比如100k，然后用size/100计算需要分成几个切片，同样也是再用slice截取
> - 在本案例中，我们将采取方法一和方法二合并的方式进行切片：我们为根据方法二固定每个切片的大小，计算出切片的个数，然后再额外规定一个最大个数，如果计算出的个数超过了最大个数，就需要根据方法一进行重新切片，如果没有超出个数，则按固定大小切片。

- **简单步骤分析**

> 在根据出切片逻辑后，下面就是要一步步实现了：
> - 首先我们先来封装一个返回promise实例的方法retrieveHash，该方法主要用于根据文件内容生成一个hash值，需要借助spark-md5和FileReader
> - 封装一个所有切片上传完成后发送合并请求方法uploadComplete
> - 在该方法外面需要定义一个计数器，每上传完成一个切片需要调用一次该方法，计数器就需要累加1
> - 当计数器的值等于切片的个数时，则说明所有切片已经上传完成，这时就可以发送合并请求进行切片合并了
> - 这里有一点需要注意：就是在发送合并请求前，最好是延迟几秒再发送，以避免一些不必要的错误
> - 在文件域的change事件中得到要上传的文件
> - 调用上面封装好的获取hash值的方法retrieveHash，然后根据hash值向服务端发送一个请求来获取已经上传过的切片列表fileList（用于断点续传判断）
> - 根据上面分析的切片逻辑进行切片，并将切片文件信息保存在数组中
> - 遍历切片数组，首先判断该切片是否已经上传，也就是看该切片文件是否已经存在于上面所获取到的文件列表fileList中
> - 如果存在则调用uploadComplete，让计数器累加
> - 如果不存在，则调用服务端切片上传接口进行文件上传，同时在上传完成后，仍需调用uploadComplete方法进行计数器累加，一旦计数器的值跟切片个数相等，则会自动调用合并接口进行文件合并
> - 至此大文件的切片上传和断点续传就实现了。

- **代码实现**

```
//upload.js 大文件切片上传，断点续传
(function () {
    let upload6 = document.querySelector("#upload6"),
        upload_inp = upload6.querySelector('.upload-inp'),
        upload_upload = upload6.querySelector('.upload-btn'),
        set_files = upload6.querySelector('.files'),
        cur_pro = upload6.querySelector('.cur-pro'),
        pro_val = upload6.querySelector('.pro-val'),
        progress = upload6.querySelector('.progress');

    const retrieveHash = function retrieveHash(file) {
        return new Promise(resolve, reject) => {
            let spark = new SparkMD5.ArrayBuffer();
            let fr = new FileReader();
            fr.readAsArrayBuffer(file);
            fr.onload = (ev) => {
                spark.append(ev.target.result);
                let hash = spark.end();
                let suffix = /\.([0-9a-zA-Z+]{,4},exec(file.name)[1];
                resolve({
                    hash,
                    suffix
                });
            };
        });

    }

    let complete = 0;
    const uploadComplete = function uploadComplete(hash, count) {
        complete++;
        let progress = (complete / count * 100).toFixed(2) + '%';
        cur_pro.style.width = progress;
        pro_val.innerHTML = progress;
        if (complete == count) return;
        cur_pro.style.width = '100%';
        pro_val.innerHTML = '100%';
        setTimeout(() => {
            request.post('/upload_merge', {
                hash,
                count
            }, {
                headers: {
                    'content-Type': 'application/x-www-form-urlencoded'
                }
            }).then(res => {
                console.log(res);
                // alert('上传成功?');
            }).catch(err => {
                console.log(err);
            });
        }, 3000);
    }
    upload_upload.addEventListener('click', function () {
        upload_inp.click();
    });

    upload_inp.addEventListener('change', async function () {
        let file = this.files[0],
        progress.style.display = 'inline-block';
        cur_pro.style.width = '0%';
        pro_val.innerHTML = '0%';
        let chunks = [];
        let {
            hash,
            suffix
        } = await retrieveHash(file);
        set_files.innerHTML = `${hash}.${suffix}` ;
        let {
            fileList
        } = await request.get('/uploaded', {
            params: {
                hash
            }
        });

        let maxSize = 100 * 1024; //100k
        let count = Math.ceil(file.size / maxSize);
        //最多切片的数量不能超过这个，并重新计算每个切片的大小
        if (count > 20) {
            maxSize = file.size / 20;
            count = 20;
        }

        let index = 0;
        while (index < count) {
            chunks.push({
                file: file.slice(index * maxSize, (index + 1) * maxSize),
                filename: `${hash}_${index+1}.${suffix}`
            });
            index++;
        }

        chunks.forEach((item, index) => {
            // 检测切片上传过程不终止了
            if (fileList && fileList.length > 0 && fileList.includes(item.filename)) {
                uploadComplete(hash, count);
                return;
            }
            let formData = new FormData();
            formData.append('file', item.file);
            formData.append('filename', item.filename);
            request.post('/upload_chunk', formData,).then(res => {
                uploadComplete(hash, count);
                // console.log(res);
                // alert('上传成功?');
            }).catch(err => {
                console.log(err);
            });
        });
    });

});()
```

总结

本文中我们进行了分模块分场景实现了关于强文件上传的几个功能点。并重点分析了大文件切片上传和断点续传的功能点。每个模块对应的功能代码都已经提供。

分类：前端　标签：JavaScript　前端

文章被收录于专栏：

JavaScript
专门记录JavaScript的所学所想与感悟

关注专栏

安装掘金浏览器插件

条内容更合浏览、多引擎快捷搜索、多工具便捷提效、多模式随心畅享。你想要的，这里都有！

前往安装

相关小册

VIP 可视化入门：从 0 到 1 开发一个图表库

数据可视化 CSP 190/周末

¥9.95 ¥19.9 直单单价价

VIP 微信小程序底层框架实现原理

夜还不够黑 、 CSP 172/周末

¥14.95 ¥29.9 直单单价价

评论

看完就，登录 分享一下感受吧~

全部评论 6

最新 最热

大椰椰 CSP 芜心麻天狂 @ 五吴 1年前

大佬，什么时候有这个的完整代码啊 😭

点赞 3

西瓜watermelon （作者） 1年前

客户端的基本就是这样了，没有html和css部门，服务器端是由完整代码的

点赞 回复

西瓜watermelon （作者） 1年前

客户端的就是这样了，没有HTML和css部分，服务器端是有完整代码的

点赞 回复

大椰椰 回复 西瓜watermelon 1年前

好的，谢谢

> 客户端的就是这样了，没有HTML和css部分，服务器端是有完整代码的

点赞 回复

JS CSP 切图仔 @ 自由人 1年前

好啊 多好的文章居然没人点赞！老子点了！！！！ 😍 😍

点赞 1

西瓜watermelon （作者） 1年前

谢谢支持 ❤

点赞 回复

复制
取消
发布