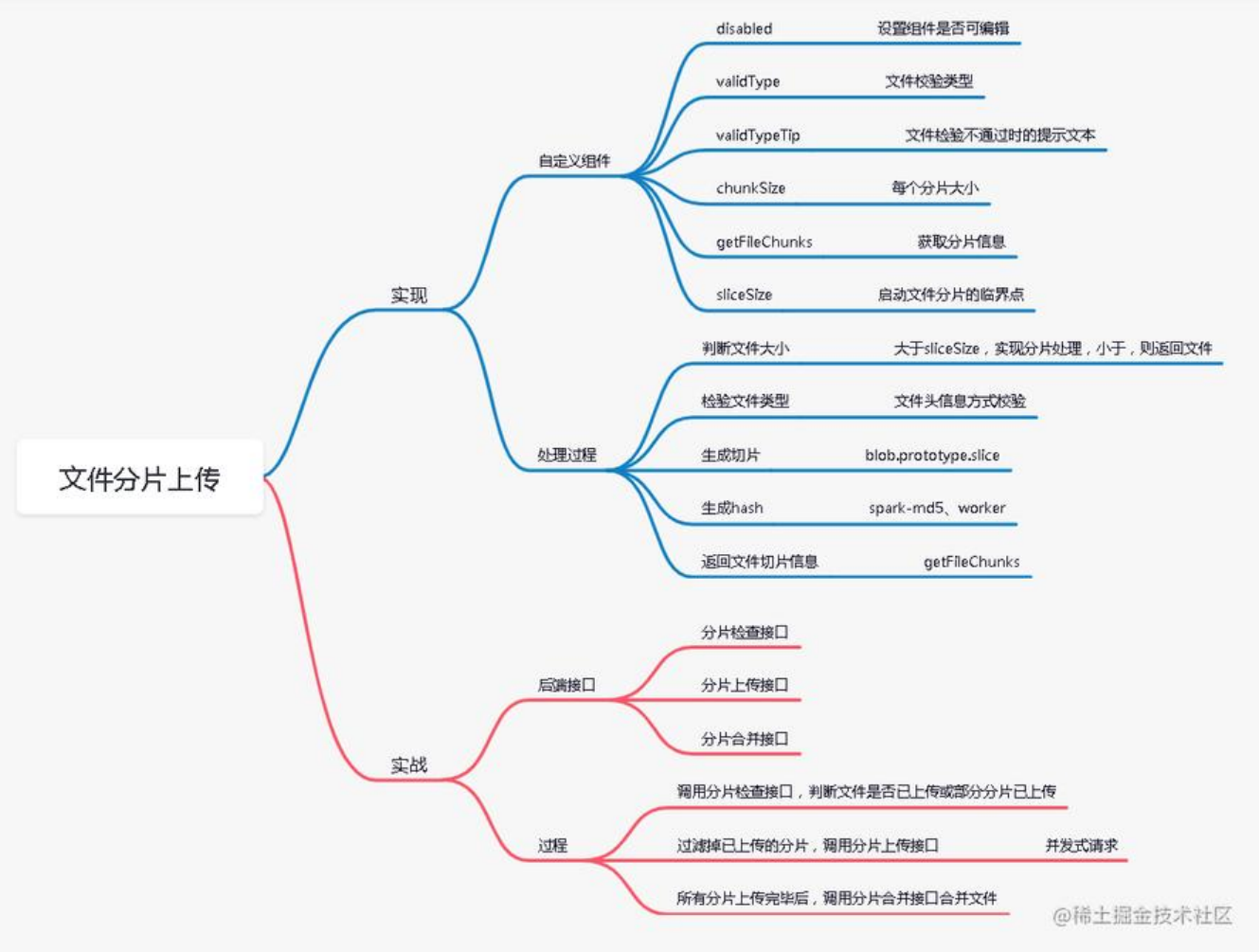
链滴

基于 elementUI 的文件分片上传封装组件

作者：Cosolar

原文链接：https://ld246.com/article/1636087050661

来源网站：链滴



1. 概念

文件分片的核心是利用Blod.prototype.slice方法，将文件切分为一个个的切片，借助 http 的可并发，同时上传多个切片

2. 实现

分片组件是基于elementUI的el-upload组件上的二次封装

```
<template>
  <div>
    <el-upload class="uploadComponent" ref="uploadFile" :drag="!disabled"
      v-bind="$attrs" :on-change="handleUpload" :disabled="disabled"
    >
      <i class="el-icon-upload" v-show="!disabled"></i>
    </el-upload>
  </div>
</template>
```

通过属性透传v-bind=“$attrs”实现绑定在组件上的属性传递给el-upload组件，因此可以在组件上定el-upload组件已实现的属性

原文链接：基于 elementUI 的文件分片上传封装组件

```
<file-chunks-upload-component
    :disabled="!isEdit" ref="videoUpload" :file-list="videoFileList"
    action="" :auto-upload="false" list-type="fileList"
    accept=".mp4, .avi, .wmv, .rmvb, .rm" validType="video"
    validTypeTip="上传文件只能是mp4, avi, wmv, rmvb格式"
    :getFileChunks="handleUpload" :sliceSize="100 * 1024 * 1024"
    :on-preview="handleVideoPreview" :before-remove="handleVideoRemove"
/>
```

组件除了接收el-upload组件的属性，还额外接收几个属性：

- **disabled**：设置组件是否可编辑
- **validType**：文件校验类型，单个检验类型的话可传入字符串，如’ video’ 、’ image’ ，多个验类型的话可传入数组，如[‘video’ ， ‘image’]
- **validTypeTip**：文件检验不通过时的提示文本
- **chunkSize**：每个分片大小
- **getFileChunks**：获取分片信息
- **sliceSize**：启动文件分片的临界点

在el-upload组件上绑定on-change属性监听文件变化，拿到文件信息，进行一系列的处理

处理过程：

1. 判断文件大小是否大于sliceSize，是，则继续实现分片处理，否，则返回文件

```
// 文件大小小于sliceSize时，不进行分片，直接返回文件
if(this.sliceSize && file.size <= this.sliceSize){
    this.getFileChunks && this.getFileChunks({file})
    return
}
```

2. 检验文件类型

通过文件头信息方式检验文件类型，封装成一个工具方法

```
/***
 * 通过文件头信息方式检验文件类型
 * https://blog.csdn.net/tjcwt2011/article/details/120333846?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-3.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-3.no_search_lin

 */

const fileType = Object.assign(Object.create(null), {
    'isMP4': str => ['00 00 00 14', '00 00 00 18', '00 00 00 1c', '00 00 00 20'].some(s => str.indeOf(s) === 0),
    'isAVI': str => str.indexOf('52 49 46 46') === 0,
    'isWMV': str => str.indexOf('30 26 b2 75') === 0,
    'isRMVB': str => str.indexOf('2e 52 4d 46') === 0,
    'isJPG': str => str.indexOf('ff d8 ff') === 0,
```

```
    'isPNG': str => str.indexOf('89 50 4e 47') === 0,
    'isGIF': str => str.indexOf('47 49 46 38') === 0
})

const validFileType = Object.assign(Object.create(null), {
    'video': str => fileType['isMP4'](str) || fileType['isAVI'](str) ||  fileType['isWMV'](str) || fileType[
isRMVB'](str),
    'image': str => fileType['isJPG'](str) || fileType['isPNG'](str) || fileType['isGIF'](str)
})

const bufferToString = async buffer => {
    let str = [...new Uint8Array(buffer)].map(b => b.toString(16).toLowerCase().padStart(2, '0')).j
in(' ')
    return str
}

const readBuffer = (file, start = 0, end = 4) => {
    return new Promise((resolve, reject) => {
        const reader = new FileReader()
        reader.onload = () => {
            resolve(reader.result)
        }
        reader.onerror = reject
        reader.readAsArrayBuffer(file.slice(start, end))
    })
}

const validator = async (type, file) => {
    const buffer = await readBuffer(file)
    const str = await bufferToString(buffer)

    switch(Object.prototype.toString.call(type)){
        case '[object String]':
            return validFileType[type](str)
        case '[object Array]':
            return [...type].some(t => validFileType[t](str))
    }
}

export default validator
```

注：各种文件类型的文件头信息可从以下链接查询 文件信息头对照表

3. 生成切片

利用blob.prototype.slice将大文件切分成一个一个的小片段

```
createFileChunk(file){
    let chunks = []
    let cur = 0
    if(file.size > this.chunkSize){
        while(cur < file.size){
            chunks.push({ file: file.slice(cur, cur + this.chunkSize) })
```

```
            cur += this.chunkSize
        }
    }else{
        chunks.push({ file: file })
    }
    return chunks
},
```

4. 生成hash

为了实现断点续传、秒传的效果，需要前端生成一个唯一标识提供给后端，由于唯一标识必须保持不，所以正确的做法是根据文件内容生成hash。这里需要用到一个库spark-md5，它可以根据文件内计算出文件的hash值。

由于计算hash是非常耗费时间的，并且会引起 UI 的阻塞，所有这里使用web-worker在worker线程算hash。由于实例化 web-worker 时，函数参数 URL 为指定的脚本且不能跨域，所以我们单独创建个 hash.js 文件放在 public 目录下，然后通过importScripts 函数用于导入 spark-md5

hash.js

```
// 引入spark-md5
self.importScripts('spark-md5.min.js')

self.onmessage = e=>{
  // 接受主线程传递的数据
  let { chunks } = e.data
  const spark = new self.SparkMD5.ArrayBuffer()

  let count = 0

  const loadNext = index=>{
    const reader = new FileReader()
    reader.readAsArrayBuffer(chunks[index].file)
    reader.onload = e=>{
      count ++
      spark.append(e.target.result)

      if(count==chunks.length){
        self.postMessage({
          hash:spark.end()
        })
      }else{
        loadNext(count)
      }
    }
  }
  loadNext(0)
}
```

分片越多，分片越大，计算hash所耗费的时间就越久，为优化计算速度，在损失一定精度的情况下，用抽样的方式计算hash

这里分了两种情况

1.文件大小小于50M时，对所有的分片计算hash

2.文件大小大于50M时，以偶数位的方式抽取分片，并提取每个分片10M的内容去计算hash

计算hash方法

```
 async calculateHashWorker(chunks, size){
    var _this = this;
    // 文件大小超过50M时，使用抽样方式生成hash
    const MAX_FILE_SIZE = 50 * 1024 * 1024
    const SAMPLE_SIZE = 10 * 1024 * 1024
    if(size >= MAX_FILE_SIZE){
        chunks = chunks.map((item, index) => {
            if(index % 2 == 0){
                return { file: item.file.slice(0, SAMPLE_SIZE)}
            }
        }).filter(item => item)
    }

    return new Promise(resolve => {
        _this.worker = new Worker('/hash.js')
        _this.worker.postMessage({ chunks })
        _this.worker.onmessage = e => {
            const { hash } = e.data
            if (hash) {
                resolve(hash)
            }
        }
    })
},
```

5. 返回文件切片信息

通过在组件上绑定getFileChunks属性获取到切片集合、hash值以及文件信息

```
async handleUpload(params){
    if(!params.raw) return
    let file = params.raw
    // 文件大小小于sliceSize时，不进行分片，直接返回文件
    if(this.sliceSize && file.size <= this.sliceSize){
        this.getFileChunks && this.getFileChunks({file})
        return
    }
    // 校验文件类型
    if(this.validType && !await validator(this.validType, file)){
        this.$message({
            type: 'warning',
            message: this.validTypeTip,
            duration: 3000,
        })
        this.clearUploadFiles()
        return
    }
```

```
    const loading  = this.$loading({
        lock: true,
        text: '生成切片中，请耐心等候'
    })

    let chunks = this.createFileChunk(file)
    let hash = await this.calculateHashWorker(chunks, file.size)

    this.getFileChunks && this.getFileChunks({chunks, hash, file})

    loading.close()
},
```

分片上传组件完整代码

```
<template>
    <div>
        <el-upload class="uploadComponent" ref="uploadFile" :drag="!disabled"
            v-bind="$attrs" :on-change="handleUpload" :disabled="disabled"
        >
            <i class="el-icon-upload" v-show="!disabled"></i>
        </el-upload>
    </div>
</template>

<script>
import validator from '@/utils/fileTypeValidate'

export default {
    name: 'FileChunksUploadComponent',
    props: {
        disabled: {  // 是否可编辑
            type: Boolean,
            default: false
        },
        validType: {  // 文件校验类型
            type: String | Array,
            default(){
                return ''
            }
        },
        validTypeTip: {  // 文件检验提示
            type: String,
            default: '上传文件格式不正确!'
        },
        chunkSize: {  // 分片大小
            type: Number,
            default: 50 * 1024 * 1024
        },
        getFileChunks: {  // 获取分片信息方法
            type: Function,
            default: null
```

```
    },
    sliceSize: {  // 文件实现分片的临界大小
        type: Number,
        default: 0
    }
},
methods: {
    async handleUpload(params){
        if(!params.raw) return
        let file = params.raw
        // 文件大小小于sliceSize时，不进行分片，直接返回文件
        if(this.sliceSize && file.size <= this.sliceSize){
            this.getFileChunks && this.getFileChunks({file})
            return
        }
        // 校验文件类型
        if(this.validType && !await validator(this.validType, file)){
            this.$message({
                type: 'warning',
                message: this.validTypeTip,
                duration: 3000,
            })
            this.clearUploadFiles()
            return
        }

        const loading  = this.$loading({
            lock: true,
            text: '生成切片中，请耐心等候'
        })

        let chunks = this.createFileChunk(file)
        let hash = await this.calculateHashWorker(chunks, file.size)

        this.getFileChunks && this.getFileChunks({chunks, hash, file})

        loading.close()
    },
    createFileChunk(file){
        let chunks = []
        let cur = 0
        if(file.size > this.chunkSize){
            while(cur < file.size){
                chunks.push({ file: file.slice(cur, cur + this.chunkSize) })
                cur += this.chunkSize
            }
        }else{
            chunks.push({ file: file })
        }
        return chunks
    },
    async calculateHashWorker(chunks, size){
        var _this = this;
        // 文件大小超过50M时，使用抽样方式生成hash
```

```
            const MAX_FILE_SIZE = 50 * 1024 * 1024
            const SAMPLE_SIZE = 10 * 1024 * 1024
            if(size >= MAX_FILE_SIZE){
                chunks = chunks.map((item, index) => {
                    if(index % 2 == 0){
                        return { file: item.file.slice(0, SAMPLE_SIZE)}
                    }
                }).filter(item => item)
            }

            return new Promise(resolve => {
                _this.worker = new Worker('/hash.js')
                _this.worker.postMessage({ chunks })
                _this.worker.onmessage = e => {
                    const { hash } = e.data
                    if (hash) {
                        resolve(hash)
                    }
                }
            })
        },
        clearUploadFiles(){
            this.$refs.uploadFile.clearFiles()
        }
    }
}
</script>
```

3. 实战篇

为了实现大文件分片上传功能，需后台提供三个接口，包括分片检查接口、分片上传接口、分片合并接口。

- **分片检查接口**：根据组件生成的hash值去检测是否有分片已上传过，文件是否已上传，实现中断续，秒传的效果，避免文件重复上传。
- **分片上传接口**：上传未上传过的分片
- **分片合并接口**：文件全部分片上传完成，告知后台将分片合并成文件

分片合并过程是需要耗费一定的时间的，对于小文件来说，分片上传反而会增加上传时间，因此这里了控制，在组件上绑定了sliceSize属性，控制文件小于100M时，组件只返回文件，直接调用文件上接口。

```
<file-chunks-upload-component
    :disabled="!isEdit" ref="videoUpload" :file-list="videoFileList"
    action="" :auto-upload="false" list-type="fileList"
    accept=".mp4, .avi, .wmv, .rmvb, .rm" validType="video"
    validTypeTip="上传文件只能是mp4, avi, wmv, rmvb格式"
    :getFileChunks="handleUpload" :sliceSize="100 * 1024 * 1024"
    :on-preview="handleVideoPreview" :before-remove="handleVideoRemove"
/>
```

3.1 分片上传流程

1. 调用分片检查接口，判断文件是否已上传或部分分片已上传

```
let {data} = await sewingCraftApi.checkChunkUpload({taskId: hash})
if(data && data.completeUpload){
    resolve(data.uploadUrl)
    return
}
```


2. 过滤掉已上传的分片，调用分片上传接口

```
if(data.map){
    chunks = chunks.filter(({file, name}) => {
        return !data.map[name] || data.map[name] != file.size
    })
}

if(chunks.length){
    let uploadRes = await httpUtil.multiRequest(chunks, uploadRequest)
    let flag = uploadRes.every(res => res.data)
    if(!flag){
        reject('上传失败！')
        return
    }
}
```


3. 所有分片上传完毕后，调用分片合并接口合并文件

```
let extName = file.name.substring(file.name.lastIndexOf('.') + 1)
sewingCraftApi.mergeChunk({taskId: hash, extName}).then(res => {
    resolve(res.data)
})
```


完整代码

```
<file-chunks-upload-component
   :disabled="!isEdit" ref="videoUpload" :file-list="videoFileList"
   action="" :auto-upload="false" list-type="fileList"
   accept=".mp4, .avi, .wmv, .rmvb, .rm" validType="video"
   validTypeTip="上传文件只能是mp4, avi, wmv, rmvb格式"
   :getFileChunks="handleUpload" :sliceSize="100 * 1024 * 1024"
   :on-preview="handleVideoPreview" :before-remove="handleVideoRemove"
/>


async handleUpload({chunks, hash, file}){
   this.loading = true
   this.loadingText = '上传中，请耐心等候'
   try{
      let res = null
      if(chunks && hash){
```

```
            res = await this.sliceChunksUpload(chunks, hash, file)
        }else{
            res = await this.videoFileUpload(file)
        }
        if(res){
            this.videoFileList.push({name: res, url: res})
            this.$messageInfo('上传成功')
        }
    }catch(err){
        this.$messageWarn('上传失败')
    }

    this.$refs.videoUpload.clearUploadFiles()
    this.loading = false
    this.loadingText = ''
},
// 分片视频文件上传
sliceChunksUpload(chunks, hash, file){
    return new Promise(async (resolve, reject) => {
        chunks = chunks.map((item, index) => {
            return {
                file: item.file,
                name: `${index}_${hash}`,
                index,
                hash
            }
        })
        try{
            let {data} = await sewingCraftApi.checkChunkUpload({taskId: hash})
            if(data && data.completeUpload){
                resolve(data.uploadUrl)
                return
            }

            if(data.map){
                chunks = chunks.filter(({file, name}) => {
                    return !data.map[name] || data.map[name] != file.size
                })
            }

            if(chunks.length){
                let uploadRes = await httpUtil.multiRequest(chunks, uploadRequest)
                let flag = uploadRes.every(res => res.data)
                if(!flag){
                    reject('上传失败！')
                    return
                }
            }

            let extName = file.name.substring(file.name.lastIndexOf('.') + 1)
            sewingCraftApi.mergeChunk({taskId: hash, extName}).then(res => {
                resolve(res.data)
            })
        }catch(err){
```

```
            reject(err)
        }

        function uploadRequest({hash, index, name, file}){
            let formData = new FormData()
            formData.append('taskId', hash)
            formData.append('chunkNumber', index)
            formData.append('identifier', name)
            formData.append('file', file)
            return sewingCraftApi.uploadChunkUpload(formData)
        }
    })
}
```

引用

字节跳动面试官：请你实现一个大文件上传和断点续传

文件类型的终极校验

128个常见的文件头信息对照表

本文转自 稀土掘金，如有侵权，请联系删除。