Vue3大文件上传：让你的文件飞！秒传、断点续传、分片上传全攻略！

开篇

> 你是否曾经遇到过上传大文件时，进度条缓慢前进的煎熬？或者上传中断后，又要重新开始上传的烦恼？别担心，Vue3大文件上传攻略来了！本文将教你如何实现秒传、断点续传和分片上传，让你的文件飞快地传！快来跟着小编的步伐，让我们一起解决这个烦人的问题吧！

首先需要明确，大文件上传需要用到后端的支持。本文中我将选择使用Node.js框架下的 Express 来搭建我们的后台。

秒传

秒传指的是已经上传过的文件，不必重复传输，可以直接从后台取回文件资源。

前端实现思路如下：

- 用户选择要上传的文件
- 计算文件的 `hash值` ，利用 `hash值` 判断该文件是否已经存在
- 如果已经存在，直接从后台获取该文件资源，上传结束
- 如果不存在，则走正常的分片上传流程

在Vue3中可以通过使用第三方库 `spark-md5` 来计算文件的hash值。

安装库：

```
1 npm install spark-md5
```

在页面中引用：

```
import SparkMD5 from 'spark-md5'

// 计算hash值
const fileReader = new FileReader() // 文件读取器
fileReader.onload = function() {
  const spark = new SparkMD5.ArrayBuffer() // 构建hash值对象
  spark.append(fileReader.result) // 添加文件二进制内容
  const hash = spark.end() // 计算hash值
  console.log(hash)
}
fileReader.readAsArrayBuffer(file)
```

后端实现思路：

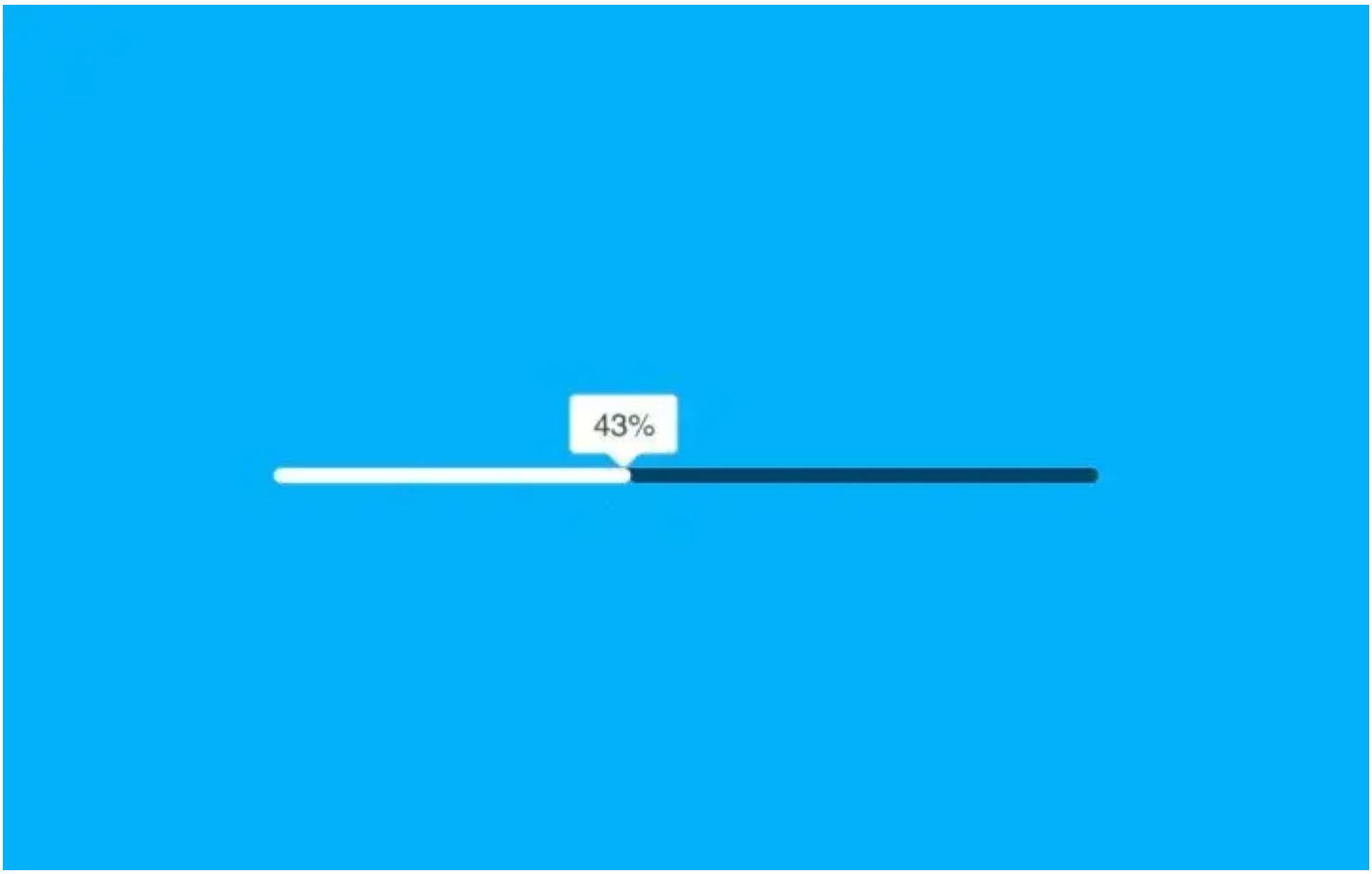
- 前端传来文件的hash值
- 后台查询数据库中是否存在该hash对应的文件
- 如果存在，返回文件资源，否则报错

在Node.js中可以通过使用 mongoose 这个库来连接数据库，并且使用 `findOne` 这个方法来查询数据库中是否存在某个hash值对应的文件。

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

// 定义文件资源的数据结构
const fileSchema = new Schema({
  name: { type: String, required: true },
  hash: { type: String, required: true },
  size: { type: Number, required: true },
  type: { type: String, required: true },
  createTime: { type: Date, default: Date.now },
  updateTime: { type: Date, default: Date.now }
})

// 定义文件资源模型
const File = mongoose.model('File', fileSchema)

// 查询是否存在该hash值对应的文件
File.findOne({ hash: req.query.hash }, (err, file) => {
  if (err) {
    console.error(err)
    res.status(500).json({ message: 'Internal Server Error' })
```

```
  } else {
    if (file) {
      res.status(200).json({ message: 'File exists', url: file.url })
    } else {
      res.status(404).json({ message: 'File not found' })
    }
  }
})
```

断点续传



断点续传指的是上传文件时，如果因为网络等原因导致上传中断，下次上传可以从中断处继续进行，而不需要重新上传整个文件。

前端实现思路：

- 使用 `FileReader` 对象读取文件内容
- 将读取内容分为多个片段，并将每个片段上传到后台
- 若上传中断，则在下次上传时从最后一次成功上传的片段之后继续上传，而不是重新上传整个文件

在Vue3中，可以使用 `Promise和async/await` 来实现断点续传功能。

```
async uploadFileChunk(file, start, end) {
  const chunk = file.slice(start, end) // 获取当前分片
```

```
  const formData = new FormData() // 构建formdata用于上传文件
  formData.append('chunk', chunk)
  formData.append('hash', this.hash)
  formData.append('name', file.name)
  formData.append('start', start)
  const config = {
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  }
  try {
    const response = await axios.post(`http://localhost:3000/upload`, formData, config)
    if (response.data.message === 'Chunk uploaded') { // 如果分片上传成功
      this.uploadFile(this.file, end) // 上传完当前分片后，继续上传下一个分片
    } else if (response.data.message === 'Upload successful') { // 如果整个文件上传成功
      this.uploading = false
      this.$emit('uploadFinish', response.data.url)
    }
  } catch (error) {
    console.error(error)
    this.uploading = false
  }
},
async uploadFile(file, start = 0) {
  const end = start + this.chunkSize // 当前分片的终止位置
  await this.uploadFileChunk(file, start, end) // 上传当前分片
}
```

后端实现思路：

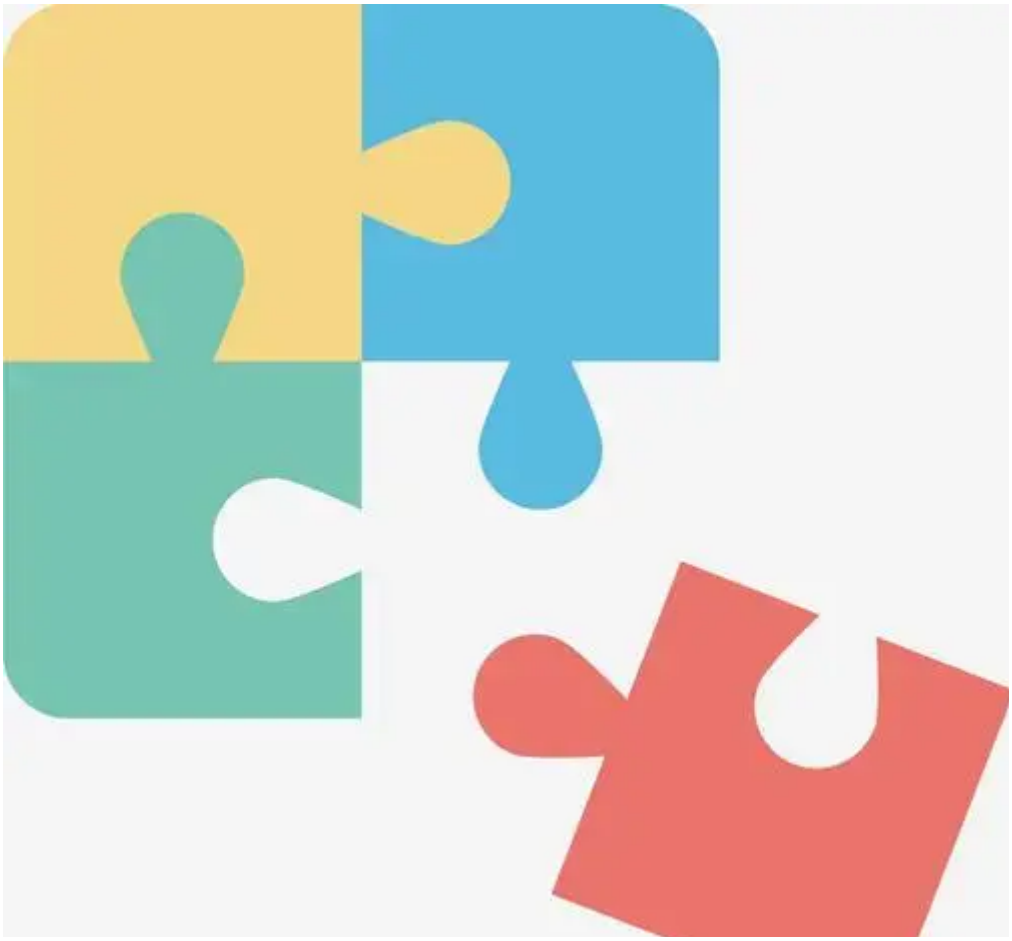
- 通过文件的hash值来判断服务器上是否存在该文件，从而实现断点续传。如果该文件在服务器上不存在，则新建一个文件，否则继续上传该文件。
- 将上传的分片存储在服务器的磁盘上，并记录该分片在整个文件中的起始位置，方便合并文件。

在Node.js中，可以使用 `formidable` 这个库来解析前端发送过来的formdata数据，并且将上传的每个分片暂存在服务器上。当所有分片上传完成后，可以根据分片信息将所有分片合并成一个完整的文件。同时，可以使用 `fs` 模块实现断点续传的功能，通过判断文件在服务器上的大小来确定从哪一个位置继续上传。

```
const formidable = require('formidable')
const fs = require('fs')
const path = require('path')

```

```
const fileDir = path.resolve('./uploads')
if (!fs.existsSync(fileDir)) fs.mkdirSync(fileDir)
let start = 0 // 文件上传的起始位置
form.on('field', (name, value) => {
  if (name === 'hash') {
    hash = value
  } else if (name === 'name') {
    name = value
  } else if (name === 'start') {
    start = Number(value)
  }
})
form.on('file', (_, file) => {
  const filePath = path.join(fileDir, name)
  const stream = fs.createWriteStream(filePath, { start, flags: 'a' })
  fs.createReadStream(file.path).pipe(stream) // 将当前分片写入指定文件
  stream.on('close', () => {
    res.status(200).json({ message: 'Chunk uploaded' })
  })
})
form.on('end', () => {
  if (fs.statSync(path.join(fileDir, name)).size === size) { // 如果文件已上传完成
    // 将文件信息存入数据库
    const newFile = new File({
      name: name,
      hash: hash,
      size: size,
      type: type,
      url: `/uploads/${name}`
    })
    newFile.save((err, file) => {
      if (err) {
        console.error(err)
        res.status(500).json({ message: 'Internal Server Error' })
      } else {
        res.status(200).json({ message: 'Upload successful', url: file.url })
      }
    })
  }
})
```

分片上传



前端实现思路：

- 将文件进行分片，每个分片大小一般为1MB
- 上传每个分片
- 所有分片上传完成后，后台将分片合并成文件

在Vue3中，可以使用 `File.slice()` 方法将文件进行分片，同时使用 `axios` 库来发送分片数据到后台。

```
const sliceSize = 1024 * 1024 // 1MB的分片大小
const chunks = Math.ceil(fileSize / sliceSize) // 文件分片数
const requests = [] // 分片请求的promise列表
for (let i = 0; i < chunks; i++) {
  const start = i * sliceSize // 当前分片在文件中的起始位置
  const end = Math.min(start + sliceSize, fileSize) // 当前分片在文件中的终止位置
  const chunk = file.slice(start, end) // 获取当前分片
  const formData = new FormData() // 构建formdata用于上传文件
  formData.append('chunk', chunk)
  formData.append('hash', hash)
  formData.append('name', file.name)
  formData.append('chunkIndex', i)
  formData.append('chunks', chunks)
  const config = {
    headers: {
```

```
      'Content-Type': 'multipart/form-data'
    },
    onUploadProgress: progressEvent => {
      const uploaded = start + progressEvent.loaded // 已上传的大小
      const total = fileSize // 文件总大小
      const percentCompleted = Math.floor((uploaded / total) * 100) // 计算上传进度
      // 更新当前分片的上传进度
      this.$set(this.chunks, i, percentCompleted)
      // 更新已上传总大小
      this.uploadedSize += progressEvent.loaded
      // 更新已上传总进度
      this.totalPercentCompleted = Math.floor((this.uploadedSize / this.fileSize) * 100)
    }
  }
  const request = axios.post(`http://localhost:3000/upload`, formData, config)
  requests.push(request)
}
Promise.all(requests) // 所有分片上传完成
```

后端实现思路：

- 分片上传时，存储每个分片，以及该分片所属文件的hash值、分片索引、分片总数等信息
- 如果所有分片上传完成，将分片合并成一个完整的文件
- 将文件信息存入数据库

在Node.js中，可以使用 `formidable` 这个库来解析前端发送过来的formdata数据，并且将上传的每个分片暂存在服务器上。当所有分片上传完成后，可以根据分片信息将所有分片合并成一个完整的文件。

```
const formidable = require('formidable')
const fs = require('fs')
const path = require('path')

// 存储分片文件
const fileDir = path.resolve('./uploads')
if (!fs.existsSync(fileDir)) fs.mkdirSync(fileDir)
form.on('field', (name, value) => {
  if (name === 'chunkIndex') {
    chunkInfo.index = Number(value)
  } else if (name === 'chunks') {
    chunkInfo.total = Number(value)
  }
})
```

```
form.on('file', (_, file) => {
  const chunkPath = path.join(fileDir, `${hash}_${chunkInfo.index}`)
  fs.renameSync(file.path, chunkPath) // 将分片文件存入指定目录
})
form.on('end', () => {
  // 如果所有分片上传完成
  if (chunkInfo.total - 1 === chunkInfo.index) {
    const file = fs.createWriteStream(path.join(fileDir, name)) // 创建新文件流
    for (let i = 0; i < chunkInfo.total; i++) {
      const chunkPath = path.join(fileDir, `${hash}_${i}`)
      const chunk = fs.readFileSync(chunkPath) // 读取分片内容
      file.write(chunk) // 将分片内容写入新文件流
      fs.unlinkSync(chunkPath) // 删除该分片
    }
    file.end()
    // 将文件信息存入数据库
    const newFile = new File({
      name: name,
      hash: hash,
      size: size,
      type: type,
      url: `/uploads/${name}`
    })
    newFile.save((err, file) => {
      if (err) {
        console.error(err)
        res.status(500).json({ message: 'Internal Server Error' })
      } else {
        res.status(200).json({ message: 'Upload successful', url: file.url })
      }
    })
  } else {
    res.status(200).json({ message: 'Chunk uploaded' })
  }
})
```

如果本文中有任何错误或不足之处，还请各位大佬多多指教，让我们一起共同进步！

结束

现在，你已经掌握了Vue3大文件上传的技巧，让你的文件不再受限于大小和速度！快来试试吧，让你的文件像闪电一样飞快传输！记得要分享给你的小伙伴哦，让他们也能享受到这个神奇的技能！