大文件分片上传和断点续传

在上传大文件时，为了提高上传的效率，我们一般会使用 Blob.slice 方法对大文件按照指定的大小进行切割，然后通过多线程进行分块上传，等所有分块都成功上传后，再通知服务端进行分块合并。

本文主要介绍在 react 中使用 antd 如何实现大文件的分片上传和断点续传。

一、获取文件对象 file

首先我们在 antd 中可以使用 upload 组件实现上传功能，但是它默认是不支持分片上传的，所以它提供了一个可以让我们实现自定义上传的方法

customRequest	通过覆盖默认的上传行为，可以自定义自己的上传实现	function	-

customRequest

Allows for advanced customization by overriding default behavior in AjaxUploader. Provide your own XMLHttpRequest calls to interface with custom backend processes or interact with AWS S3 service through the aws-sdk-js package.

customRequest callback is passed an object with:

- `onProgress: (event: { percent: number }): void`
- `onError: (event: Error, body?: Object): void`
- `onSuccess: (body: Object): void`
- `data: Object`
- `filename: String`
- `file: File`
- `withCredentials: Boolean`
- `action: String`
- `headers: Object`

```
customRequest file
```

```
uploadProps = {
    customRequest: (options) => {
      const { action, data, file, filename, headers, onError, onProgress,
onSuccess, withCredentials } = options
      this.splitUpload(file)
    }
  }
  render() {
    return (
      <div>
        <Upload {...this.uploadProps}>
          <Button icon={<UploadOutlined />}></Button>
        </Upload>
      </div>
    )
  }
```

获取到这个文件对象之后，就可以对其进行切片操作

二、对文件对象进行切片

切片操作主要使用 file.slice() 实现

```
let start = 0
      let end = 0
      while (1) {
        end += this.chunkSize
        const chunk = file.slice(start, end)
        start = end
        if (!chunk.size) break
        this.fileChunks.push(chunk)
      }
```

例如一个文件的大小是100kb, 我们设置的分片大小是10kb, 那么自然就可以分成10片，把分片都存入数组 fileChunks 中，在后面上传时使用

对文件分片完成后，我们需要想一个问题，服务器接收到若干分片后，如何知道哪些分片是属于同一个文件的呢？因此每次上传分片时我们都需要给属于同一文件的分片带上一个唯一标识

最保险的方式就是生成这个文件的md5当做唯一标识

三、生成文件唯一标识

这里主要是使用了 spark-md5 这个库来生成文件的 MD5

```
const token = await this.calcFileMD5(file)


// md5, ,
  calcFileMD5(file) {
    return new Promise((resolve, reject) => {
      let chunkSize = this.chunkSize,
        chunks = Math.ceil(file.size / chunkSize),
        currentChunk = 0,
        spark = new SparkMD5.ArrayBuffer(),
        fileReader = new FileReader()

      fileReader.onload = (e) => {
        spark.append(e.target.result)
        currentChunk++
        //
        this.md5Percent = Math.floor((currentChunk / chunks) *
100).toFixed(2)
        // console.log('md5Percent: ', this.md5Percent + '%')
        if (currentChunk < chunks) {
          loadNext()
        } else {
          resolve(spark.end())
        }
      }

      fileReader.onerror = (e) => {
        reject(fileReader.error)
        fileReader.abort()
      }

      function loadNext() {
        let start = currentChunk * chunkSize,
          end = start + chunkSize >= file.size ? file.size : start +
chunkSize
        fileReader.readAsArrayBuffer(file.slice(start, end))
      }
      loadNext()
    })
  }
```

现在我们有了文件的分片和MD5，可以准备上传了，但是在上传之前我们需要先发个请求确认一下，这个文件是否已经存在于服务器上了，如果已经存在了，我们就可以立马告诉用户上传成功，等于秒传了

四、查询文件是否存在

这一步比较简单，主要就是发起一个get请求，参数就是文件的 md5 和文件名，查询一下服务器上是否已经存在该文件

```
const fileStatus = await this.checkFileExist(token, fileName)


  //
  checkFileExist(token, fileName) {
    return request
      .get('/exist', {
        params: {
          token,
          fileName
        }
      })
      .then((result) => {
        return result.data
      })
      .catch((err) => {
        console.log(err)
        this.onError(err)
      })
  }
```

这个接口不仅可以查询该文件是否存在，还可以查询该文件的分片是否存在，以及已经上传了哪些分片，我们实现断点续传的关键就在于此。

现在，我们就可以真正上传文件分片了

五、上传文件分片

```
// ,
  this.upload(token, fileName, fileStatus)


  //
  upload(token, fileName, fileStatus) {
    const { chunkIds } = fileStatus.data
    const poolLimit = 4 //
    console.log('array: ', [...new Array(this.fileChunks.length).keys()])
    return asyncPool(poolLimit, [...new
Array(this.fileChunks.length).keys()], (i) => {
      // console.log(chunkIds, i, chunkIds.includes(i))
      if (chunkIds.includes(i)) {
        console.log('', chunkIds, i)
        // +1
        this.alreadySendCount += 1
        this.updateProgress()
        // ,
        return Promise.resolve()
      }
      return this.uploadChunk(token, fileName, i)
    })
  }
```

我们在上面检查文件是否存在的接口中，可以获取到已经上传了哪些分片的信息，实现断点续传的关键就是在上传时跳过这些已经上传的分片

在这里我们使用了一个函数 asyncPool，主要是用来控制并发数的，因为如果我们的分片过多，就会在瞬间发出大量的 http 请求（tcp 连接数不足可能造成等待），或者堆积无数调用栈导致内存溢出。

这个函数有三个参数，第一个是并发限制数，第二个是一个数组，每一个元素代表一个任务，第三个参数是一个迭代器函数，接受两个参数(数组项和数组本身)，前面数组中的每一项都会由该函数做一些处理。

上面代码中的含义就是，先根据文件分片数定义一个数组，比如有6个分片，该数组就为 [0,1,2,3,4,5]，然后遍历数组，针对数组的每一项都执行第三个迭代函数，在迭代函数中首先判断当前分片是否已经存在于服务器，

如果存在就直接跳过，否则才上传该分片

```
//
  uploadChunk(token, fileName, i) {
    const fd = new FormData() //FormData
    fd.append('file', this.fileChunks[i], `${token + '-' + i}`)

    return request
      .post('/bigFile', fd)
      .then((result) => {
        this.alreadySendCount += 1
        this.updateProgress()
        if (this.alreadySendCount === this.fileChunks.length) {
          message.success(', ')
          // ,
          request
            .get('/mergeChunks', {
              params: {
                fileName,
                token: token
              }
            })
            .then((result) => {
              message.success('')
            })
            .catch((err) => {
              console.log(err)
              this.onError(err)
            })
        }
      })
      .catch((err) => {
        console.log(err)
        this.onError(err)
      })
  }
```

上传分片的逻辑也非常简单，就是发起了一个post请求，参数为一个 formData 对象，添加了一个属性。其中第三个参数指定了该分片所属文件的MD5值和该分片的索引，服务器在保存分片时需要用到这两个字段。

这个方法有两个版本：一个有两个参数的版本和一个有三个参数的版本。

```
formData.append(name, value);
formData.append(name, value, filename);
```

参数

`name`

`value`中包含的数据对应的表单名称。

`value`

表单的值。可以是`USVString` 或 `Blob` (包括子类型，如 `File`)。

`filename` 可选

传给服务器的文件名称 (一个 `USVString`), 当一个 `Blob` 或 `File` 被作为第二个参数的时候，`Blob` 对象的默认文件名是 "blob"。 `File` 对象的默认文件名是该文件的名称。

六、通知服务器合并文件

在上面的代码中，我们每次上传完成一个分片，就会对计数器 alreadySendCount + 1，在 alreadySendCount = 分片数的时候，就上传完了所有分片，此时需要发起请求通知服务器进行文件合并，合并请求的参数也是文件名和文件的MD5

基本上整个大文件分片上传的流程就如上所述了。

实现代码：

前端：src/pages/example/bigFileUpload • react-template • tianji-templates / vue_es • GitLab (rong360.com)

服务端：front-end/app.js at master • RJM1996/front-end (github.com)

参考文章：

1.JavaScript 中如何实现大文件并发上传？ (qq.com)

2.文件上传，搞懂这8种场景就够了 (juejin.cn)