link: null
title: 珠峰架构师成长计划
description: src\Upload.tsx
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=81 sentences=603, words=4499

上传功能

- 整体上传
- 分片上传
- 进度条
- 秒传
- 断点续传(支持单个分片续传和刷新浏览器续传)
- 暂停和恢复

1.初始化项目

```
create-react-app --template=typescript
cd create-react-app
cnpm i antd -S
```

2.单个上传

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

```
import React from 'react';
import Upload from './Upload';
import './App.css';

function App() {
    return (
        <div className="App">
            <Upload />
        div>
    );
}

export default App;
```

```
@import '~antd/dist/antd.css';;
.App {
  padding: 20px;
}
```

src\Upload.tsx

```tsx
import React, { ChangeEvent, useState, useEffect } from 'react';
import { Input, Row, Col, Button, message } from 'antd';
import { request } from './utils';
enum UploadStatus {
    INIT,
    PAUSE,
    UPLOADING
}
interface Props {

}
function Upload(props: Props) {
    const [uploadStatus, setUploadStatus] = useState(UploadStatus.INIT);
    const [currentFile, setCurrentFile] = useState();
    const [objectUrl, setObjectUrl] = useState('');
    useEffect(() => {
        if (currentFile) {
            const URL = window.URL;
            let objectUrl = URL.createObjectURL(currentFile);
            setObjectUrl(objectUrl);
            return () => {
                URL.revokeObjectURL(objectUrl);
            }
        }
    }, [currentFile]);
    const handleChange = (event: ChangeEvent) => {
        const file: File = event.target.files![0];
        setCurrentFile(file);
        console.log('file', file);
        reset();
    }
    function reset() {
        setUploadStatus(UploadStatus.INIT);
    }
    const handleUpload = async () => {
        if (!currentFile)
            return message.error('你尚未选择文件');
        if (!beforeUpload(currentFile))
            return message.error('不支持此类型文件的上传');
        const formData = new FormData();
        formData.append("chunk", currentFile);
        formData.append("filename", currentFile.name);
        let result = await request({
            url: `/upload`,
            method: 'POST',
            data: formData,
        });
        console.log('上传结果', result);
        message.info('上传成功!');
        reset();
    }
    return (
        <div className="upload">
            <Row>
                <Col span={12}>
                    <Input type="file" style={{ width: 300 }} onChange={handleChange} />
                    {uploadStatus === UploadStatus.INIT && <Button style={{ marginLeft: 10 }} type="primary" onClick={handleUpload}>上传Button>}
                Col>
                <Col span={12}>
                    {objectUrl && <img style={{ maxWidth: 150, maxHeight: 150 }} src={objectUrl} />}
                Col>
            Row>
        div>
    )
}
function beforeUpload(file: File) {
    const isValidFileType = ['image/jpeg', 'image/png', 'application/pdf', 'video/mp4'].includes(file.type);
    if (!isValidFileType) {
        message.error('不支持此文件类型!');
    }
    const isLt2G = file.size / 1024 / 1024 < 1024 * 1024 * 1024;
    if (!isLt2G) {
        message.error('上传的图片不能大于2MB!');
    }
    return isValidFileType && isLt2G;
}

export default Upload;
```

src\utils.tsx

```
function request(options: any) {
    let _default: any = {
        baseURL: 'http://localhost:8000',
        method: 'GET',
        header: {},
        data: {}
    };
    options = { ..._default, ...options, headers: { ..._default.headers, ...(options.headers || {}) } };
    return new Promise((resolve: Function, reject: Function) => {
        const xhr = new XMLHttpRequest();
        xhr.open(options.method, options.baseURL + options.url, true);
        Object.entries(options.headers).forEach(([key, value]) => xhr.setRequestHeader(key, value as string));
        xhr.responseType = 'json';
        xhr.onreadystatechange = () => {
            if (xhr.readyState === 4) {
                if (/(2|3)\d{2}/.test('' + xhr.status)) {
                    resolve(xhr.response);
                } else {
                    reject(xhr.response);
                }
            }
        }
        xhr.send(options.data);
    });
}

export {
    request
}
```

3.切片上传

src\Upload.tsx

```
import React, { ChangeEvent, useState, useEffect } from 'react';
import { Input, Row, Col, Button, message } from 'antd';
import { request } from './utils';
const SIZE = 1024 * 1024 * 100;
//const SIZE = 1024 * 10;
enum UploadStatus {
    INIT,//初始态
    PAUSE,//暂停中
    UPLOADING//上传中
}
interface Part {
    size: number;
    chunk: Blob;
    filename?: string;
    chunk_name?: string;
}
interface Props {

}
function Upload(props: Props) {
    const [uploadStatus, setUploadStatus] = useState(UploadStatus.INIT);
    const [currentFile, setCurrentFile] = useState();
    const [objectUrl, setObjectUrl] = useState('');
    const [worker, setWorker] = useState(null);
    const [hashPercent, setHashPercent] = useState(0);
    const [filename, setFilename] = useState('');
    const [partList, setPartList] = useState([]);
    useEffect(() => {
        if (currentFile) {
            const URL = window.URL;
            let objectUrl = URL.createObjectURL(currentFile);
            setObjectUrl(objectUrl);
            return () => {
                URL.revokeObjectURL(objectUrl);
            }
        }
    }, [currentFile]);
    const handleChange = (event: ChangeEvent) => {
        const file: File = event.target.files![0];
        setCurrentFile(file);
        console.log('file', file);
        reset();
    }
    function reset() {
        setUploadStatus(UploadStatus.INIT);
    }
    const calculateHash = (partList: Part[]): Promise => {
        return new Promise(resolve => {
            let worker = new Worker("/hash.js");
            setWorker(worker);
            worker.postMessage({ partList });
            worker.onmessage = (event) => {
                const { percent, hash } = event.data;
                setHashPercent(percent);
                if (hash) {
                    resolve(hash);
                }
            };
        });
    }
    const handleUpload = async () => {
        if (!currentFile)
            return message.error('你尚未选择文件');
        if (!beforeUpload(currentFile))
            return message.error('不支持此类型文件的上传');
        let partList: Part[] = createChunks(currentFile);
        let fileHash: string = await calculateHash(partList);
        let lastDotIndex = currentFile.name.lastIndexOf('.');;
```

```
        let extName = currentFile.name.slice(lastDotIndex);//.jpg .png
        let filename = `${fileHash}${extName}`;
        setFilename(filename);
        partList = partList.map((part, index: number) => ({
            filename,//文件名
            chunk_name: `${filename}-${index}`,//分块的名称
            chunk: part.chunk,//代码块
            size: part.chunk.size//此代码块的大小
        }));
        setPartList(partList);
        await uploadParts(partList, filename);
    }
    async function uploadParts(partList: Part[], filename: string) {
        let requests = createRequests(partList);
        await Promise.all(requests);
        await request({
            url: '/merge',
            method: 'POST',
            headers: { 'Content-Type': "application/json" },
            data: JSON.stringify({ filename })
        });
        message.info('上传成功!');
        reset();
    }
    function createRequests(partList: Part[]) {
        return partList.map((part: Part) => {
            return request({
                url: `/upload/${part.filename}/${part.chunk_name!}`,
                method: 'POST',
                header: { 'Content-Type': 'application/octet-stream' },
                data: part.chunk
            });
        })
    }
    return (

                    {uploadStatus === UploadStatus.INIT && 上传}

                    {objectUrl && }

    )
}
function beforeUpload(file: File) {
    const isValidFileType = ['image/jpeg', 'image/png', 'application/pdf', 'video/mp4'].includes(file.type);
    if (!isValidFileType) {
        message.error('不支持此文件类型!');
    }
    const isLt2G = file.size / 1024 / 1024 < 1024 * 1024 * 1024;
    if (!isLt2G) {
        message.error('上传的图片不能大于2MB!');
    }
    return isValidFileType && isLt2G;
}
function createChunks(file: File): Part[] {
    let current = 0;
    const partList: Part[] = [];
    while (current < file.size) {
        const chunk = file.slice(current, current + SIZE);
        partList.push({ chunk, size: chunk.size });
        current += SIZE;
    }
    return partList;
}
export default Upload;
```

public\hash.js

```
self.importScripts('https://cdn.bootcss.com/spark-md5/3.0.0/spark-md5.js');
self.onmessage = async (event) => {
    var { partList } = event.data;
    const spark = new self.SparkMD5.ArrayBuffer();
    var percent = 0;
    var perSize = 100 / partList.length;
    var buffers = await Promise.all(partList.map(({ chunk }) => new Promise((resolve) => {
        const reader = new FileReader();
        reader.readAsArrayBuffer(chunk);
        reader.onload = (event) => {
            percent += perSize;
            self.postMessage({ percent: Number(percent.toFixed(2)) });
            resolve(event.target.result);
        }
    })));
    buffers.forEach(buffer => spark.append(buffer));
    self.postMessage({ percent: 100, hash: spark.end() });
    self.close();
}
```

4.进度条

src\Upload.tsx

```
import React, { ChangeEvent, useState, useEffect } from 'react';
+import { Input, Row, Col, Button, message, Progress, Table } from 'antd';
import { request } from './utils';
const SIZE = 1024 * 1024 * 100;
//const SIZE = 1024 * 10;
enum UploadStatus {
    INIT,//初始态
    PAUSE,//暂停中
    UPLOADING//上传中
}
interface Part {
    size: number;
    chunk: Blob;
```

```
    filename?: string;
    chunk_name?: string;
+    percent?: number;
}
interface Props {

}
function Upload(props: Props) {
    const [uploadStatus, setUploadStatus] = useState(UploadStatus.INIT);
    const [currentFile, setCurrentFile] = useState();
    const [objectUrl, setObjectUrl] = useState('');
    const [worker, setWorker] = useState(null);
    const [hashPercent, setHashPercent] = useState(0);
    const [filename, setFilename] = useState('');
    const [partList, setPartList] = useState([]);
    useEffect(() => {
        if (currentFile) {
            const URL = window.URL;
            let objectUrl = URL.createObjectURL(currentFile);
            setObjectUrl(objectUrl);
            return () => {
                URL.revokeObjectURL(objectUrl);
            }
        }
    }, [currentFile]);
    const handleChange = (event: ChangeEvent) => {
        const file: File = event.target.files![0];
        setCurrentFile(file);
        reset();
    }
    function reset() {
        setUploadStatus(UploadStatus.INIT);
+        setHashPercent(0);
    }
    const calculateHash = (partList: Part[]): Promise => {
        return new Promise(resolve => {
            let worker = new Worker("/hash.js");
            setWorker(worker);
            worker.postMessage({ partList });
            worker.onmessage = (event) => {
                const { percent, hash } = event.data;
                setHashPercent(percent);
                console.log('percent', percent);
                if (hash) {
                    resolve(hash);
                }
            };
        });
    }
    const handleUpload = async () => {
        if (!currentFile)
            return message.error('你尚未选择文件');
        if (!beforeUpload(currentFile))
            return message.error('不支持此类型文件的上传');
+        setUploadStatus(UploadStatus.UPLOADING);
        let partList: Part[] = createChunks(currentFile);
        let fileHash: string = await calculateHash(partList);
        let lastDotIndex = currentFile.name.lastIndexOf('.');;
        let extName = currentFile.name.slice(lastDotIndex);//.jpg .png
        let filename = `${fileHash}${extName}`;
        setFilename(filename);
        partList = partList.map((part, index: number) => ({
            filename,//文件名
            chunk_name: `${filename}-${index}`,//分块的名称
            chunk: part.chunk,//代码块
            size: part.chunk.size,//此代码块的大小
+            percent: 0
        }));
        setPartList(partList);
        await uploadParts(partList, filename);
    }
    async function uploadParts(partList: Part[], filename: string) {
        let requests = createRequests(partList);
        await Promise.all(requests);
        await request({
            url: '/merge',
            method: 'POST',
            headers: { 'Content-Type': "application/json" },
            data: JSON.stringify({ filename })
        });
        message.info('上传成功!');
        reset();
    }
    function createRequests(partList: Part[]) {
        return partList.map((part: Part) => {
            return request({
                url: `/upload/${part.filename}/${part.chunk_name!}`,
                method: 'POST',
                header: { 'Content-Type': 'application/octet-stream' },
                data: part.chunk,
+                onProgress: (event: ProgressEvent) => {
+                    part.percent = event.loaded / part.chunk.size * 100;
+                    setPartList([...partList]);
+                }
            });
        })
    }
+    const columns = [
+        {
+            title: '切片名称',
+            dataIndex: 'filename',
+            key: 'filename',
+            width: '20%'
+        },
```

```
+        {
+            title: '切片进度',
+            dataIndex: 'percent',
+            key: 'percent',
+            width: '80%',
+            render: (value: number) => {
+                return
+            }
+        },
+    ];
+    let totalPercent = partList.length > 0 ? Math.round(partList.reduce((acc, curr) => acc + curr.percent!, 0) / (partList.length * 100) * 100) : 0;
+    let uploadProgress = uploadStatus != UploadStatus.INIT ? (
+        <>
+
+
+                    哈希计算:
+
+
+
+
+
+
+
+
+                    总体进度:
+
+
+
+
+
+
+                columns={columns}
+                dataSource={partList}
+                rowKey={(row: Part) => row.chunk_name!}
+            />
+        </>
+    ) : null;
    return (

                    {uploadStatus

                    {objectUrl && }

+            {uploadProgress}

    )
}
function beforeUpload(file: File) {
    const isValidFileType = ['image/jpeg', 'image/png', 'application/pdf', 'video/mp4'].includes(file.type);
    if (!isValidFileType) {
        message.error('不支持此文件类型!');
    }
    const isLt2G = file.size / 1024 / 1024 < 1024 * 1024 * 1024;
    if (!isLt2G) {
        message.error('上传的图片不能大于2MB!');
    }
    return isValidFileType && isLt2G;
}
function createChunks(file: File): Part[] {
    let current = 0;
    const partList: Part[] = [];
    while (current < file.size) {
        const chunk = file.slice(current, current + SIZE);
        partList.push({ chunk, size: chunk.size });
        current += SIZE;
    }
    return partList;
}
export default Upload;
```

src\utils.tsx

```
function request(options: any) {
    let _default: any = {
        baseURL: 'http://localhost:8000',
        method: 'GET',
        header: {},
        data: {}
    };
    options = { ..._default, ...options, headers: { ..._default.headers, ...(options.headers || {}) } };
    return new Promise((resolve: Function, reject: Function) => {
        const xhr = new XMLHttpRequest();
        xhr.open(options.method, options.baseURL + options.url, true);
        Object.entries(options.headers).forEach(([key, value]) => xhr.setRequestHeader(key, value as string));
        xhr.responseType = 'json';
+        xhr.upload.onprogress = options.onProgress;
        xhr.onreadystatechange = () => {
            if (xhr.readyState
                if (/(2|3)\d{2}/.test('' + xhr.status)) {
                    resolve(xhr.response);
                } else {
                    reject(xhr.response);
                }
            }
        }
        xhr.send(options.data);
    });
}

export {
    request
}
```

5.断点续传和秒传

src\Upload.tsx

```
import React, { ChangeEvent, useState, useEffect } from 'react';
import { Input, Row, Col, Button, message, Progress, Table } from 'antd';
import { request } from './utils';
const SIZE = 1024 * 1024 * 100;
//const SIZE = 1024 * 10;
enum UploadStatus {
    INIT,//初始态
    PAUSE,//暂停中
    UPLOADING//上传中
}
interface Part {
    size: number;
    chunk: Blob;
    filename?: string;
    chunk_name?: string;
    percent?: number;
+    loaded: number;
+    xhr?: XMLHttpRequest
}
+interface Uploaded {
+    filename: string,
+    size: number;
+}
interface Props {

}
function Upload(props: Props) {
    const [uploadStatus, setUploadStatus] = useState(UploadStatus.INIT);
    const [currentFile, setCurrentFile] = useState();
    const [objectUrl, setObjectUrl] = useState('');
    const [worker, setWorker] = useState(null);
    const [hashPercent, setHashPercent] = useState(0);
    const [filename, setFilename] = useState('');
    const [partList, setPartList] = useState([]);
    useEffect(() => {
        if (currentFile) {
            const URL = window.URL;
            let objectUrl = URL.createObjectURL(currentFile);
            setObjectUrl(objectUrl);
            return () => {
                URL.revokeObjectURL(objectUrl);
            }
        }
    }, [currentFile]);
    const handleChange = (event: ChangeEvent) => {
        const file: File = event.target.files![0];
        setCurrentFile(file);
        reset();
    }
    function reset() {
        setUploadStatus(UploadStatus.INIT);
        setHashPercent(0);
+        setPartList([]);
+        setObjectUrl('');
+        setWorker(null);
+        setFilename('');
    }
    const calculateHash = (partList: Part[]): Promise => {
        return new Promise(resolve => {
            let worker = new Worker("/hash.js");
            setWorker(worker);
            worker.postMessage({ partList });
            worker.onmessage = (event) => {
                const { percent, hash } = event.data;
                setHashPercent(percent);
                if (hash) {
                    resolve(hash);
                }
            };
        });
    }
    const handleUpload = async () => {
        if (!currentFile)
            return message.error('你尚未选择文件');
        if (!beforeUpload(currentFile))
            return message.error('不支持此类型文件的上传');
        setUploadStatus(UploadStatus.UPLOADING);
        let partList: Part[] = createChunks(currentFile);
        let fileHash: string = await calculateHash(partList);
        let lastDotIndex = currentFile.name.lastIndexOf('.');;
        let extName = currentFile.name.slice(lastDotIndex);//.jpg .png
        let filename = `${fileHash}${extName}`;
        setFilename(filename);
        partList = partList.map((part, index: number) => ({
            filename,//文件名
            chunk_name: `${filename}-${index}`,//分块的名称
            chunk: part.chunk,//代码块
            size: part.chunk.size,//此代码块的大小
+            loaded: 0,
            percent: 0
        }));
        setPartList(partList);
        await uploadParts(partList, filename);
    }
+    const verify = async (filename: string) => {
+        const result = await request({
+            url: "/verify",
+            method: 'POST',
+            headers: { "content-type": "application/json" },
+            data: JSON.stringify({ filename })
+        });
+        return result;
```

```
+    }
+    async function uploadParts(partList: Part[], filename: string) {
+        const { needUpload, uploadedList } = await verify(filename) as any;
+        if (!needUpload) {
+            message.success("秒传成功");
+            return reset();
+        }
+        try {
+            let requests = createRequests(partList, uploadedList);
+            await Promise.all(requests);
+            await request({
+                url: '/merge',
+                method: 'POST',
+                headers: { 'Content-Type': "application/json" },
+                data: JSON.stringify({ filename })
+            });
+            message.info('上传成功!');
+            reset();
+        } catch (err) {
+            message.info('上传失败!');
+        }
+    }
+    function createRequests(partList: Part[], uploadedList: Uploaded[]) {
+        return partList.filter((part: Part) => {
+            let uploadedFile = uploadedList.find(item => item.filename === part.chunk_name);
+            if (!uploadedFile) {
+                part.loaded = 0;
+                part.percent = 0;
+                return true;
+            }
+            if (uploadedFile.size < part.chunk.size) {
+                part.loaded = uploadedFile.size;
+                part.percent = Number(((part.loaded / part.chunk.size) * 100).toFixed(2));
+                return true;
+            }
+            return false;
+        }).map((part: Part) => {
+            return request({
+                url: `/upload/${part.filename}/${part.chunk_name!}/${part.loaded!}`,
+                method: 'POST',
+                header: { 'Content-Type': 'application/octet-stream' },
+                data: part.chunk.slice(part.loaded!),
+                setXHR: (xhr: XMLHttpRequest) => { part.xhr = xhr },
+                onProgress: (event: ProgressEvent) => {
+                    part.percent = Number((Number(part.loaded + event.loaded) / part.chunk.size * 100).toFixed(2));
+                    setPartList([...partList]);
+                }
+            });
+        })
+    }
+    const handlePause = () => {
+        partList.forEach((part: Part) => part.xhr && part.xhr.abort());
+        setUploadStatus(UploadStatus.PAUSE);
+    }
+    const handleResume = async () => {
+        setUploadStatus(UploadStatus.UPLOADING);
+        await uploadParts(partList, filename);
+    }
    const columns = [
        {
            title: '切片名称',
            data
            key: 'filename',
            width: '20%'
        },
        {
            title: '切片进度',
            data
            key: 'percent',
            width: '80%',
            render: (value: number) => {
                return
            }
        },
    ];
    let totalPercent = partList.length > 0 ? Math.round(partList.reduce((acc, curr) => acc + curr.percent!, 0) / (partList.length * 100) * 100) : 0;
    let uploadProgress = uploadStatus != UploadStatus.INIT ? (
        <>

                    哈希计算:

                    总体进度:

             row.chunk_name!}
            />
        </>
    ) : null;
    return (

                    {uploadStatus
+                    {uploadStatus === UploadStatus.UPLOADING && 暂停}
+                    {uploadStatus === UploadStatus.PAUSE && 恢复}

                    {objectUrl && }

            {
                uploadProgress
            }

    )
}
function beforeUpload(file: File) {
    const isValidFileType = ['image/jpeg', 'image/png', 'application/pdf', 'video/mp4'].includes(file.type);
    if (!isValidFileType) {
```

```
        message.error('不支持此文件类型!');
    }
    const isLt2G = file.size / 1024 / 1024 < 1024 * 1024 * 1024;
    if (!isLt2G) {
        message.error('上传的图片不能大于2MB!');
    }
    return isValidFileType && isLt2G;
}
function createChunks(file: File): Part[] {
    let current = 0;
    const partList: Part[] = [];
    while (current < file.size) {
        const chunk = file.slice(current, current + SIZE);
+        partList.push({ chunk, size: chunk.size, loaded: 0 });
        current += SIZE;
    }
    return partList;
}
export default Upload;
```

src\utils.tsx

```
function request(options: any) {
    let _default: any = {
        baseURL: 'http://localhost:8000',
        method: 'GET',
        header: {},
        data: {}
    };
    options = { ..._default, ...options, headers: { ..._default.headers, ...(options.headers || {}) } };
    return new Promise((resolve: Function, reject: Function) => {
        const xhr = new XMLHttpRequest();
        xhr.open(options.method, options.baseURL + options.url, true);
        Object.entries(options.headers).forEach(([key, value]) => xhr.setRequestHeader(key, value as string));
        xhr.responseType = 'json';
        xhr.upload.onprogress = options.onProgress;
        xhr.onreadystatechange = () => {
            if (xhr.readyState
                if (/(2|3)\d{2}/.test('' + xhr.status)) {
                    resolve(xhr.response);
                } else {
                    reject(xhr.response);
                }
            }
        }
+        options.setXHR && options.setXHR(xhr);
        xhr.send(options.data);
    });
}
export {
    request
}
```

1.初始化项目

```
mkdir server
cd server
cnpm init -y
npx tsconfig.json
cnpm i fs-extra express morgan http-errors http-status-codes cors  multer multiparty -S
cnpm i @types/fs-extra @types/node   @types/express @types/morgan @types/http-errors cross-env typescript ts-node ts-node-dev nodemon @types/cors @types/multer
@types/multiparty -D
```

2.单个上传

```
"scripts": {
    "start": "cross-env PORT=8000 ts-node-dev --respawn ./src/www.ts",
    "dev": "cross-env PORT=8000  nodemon --exec ts-node --files ./src/www.ts",
    "utils": "ts-node ./src/utils.ts"
}
```

```
import app from './app';
import http from 'http';

const port = process.env.PORT || 8000;

const server = http.createServer(app);

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
function onError(error: any) {
    console.error(error);
}
function onListening() {
    console.log('Listening on ' + port);
}
```

src\app.ts

```
import createError from 'http-errors';
import express, { Request, Response, NextFunction } from 'express';
import logger from 'morgan';
import { INTERNAL_SERVER_ERROR } from 'http-status-codes';
import cors from 'cors';
import path from 'path';
import { PUBLIC_DIR } from './utils';
import fs from 'fs-extra';
import multiparty from 'multiparty';
let app = express();
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cors());
app.use(express.static(path.resolve(__dirname, 'public')));
app.post('/upload', async (req: Request, res: Response, next: NextFunction) => {
    let form = new multiparty.Form();
    form.parse(req, async (err, fields, files) => {
        if (err) {
            return next(err);
        }
        let [filename] = fields.filename;
        let [chunk] = files.chunk;
        await fs.move(chunk.path, path.resolve(PUBLIC_DIR, filename), { overwrite: true });
        setTimeout(() => {
            res.json({
                success: true
            });
        }, 3000);
    });
});
app.use(function (_req, _res, next) {
    next(createError(404));
});

app.use(function (error: any, _req: Request, res: Response, _next: NextFunction) {
    res.status(error.status || INTERNAL_SERVER_ERROR);
    res.json({
        success: false,
        error
    });
});

export default app;
```

src\utils.ts

```
import path from 'path';
export const TEMP_DIR = path.resolve(__dirname, 'temp');
export const PUBLIC_DIR = path.resolve(__dirname, 'public');
```

3.文件切割和合并

```
import path from 'path';
import fs, { WriteStream } from 'fs-extra';
export const TEMP_DIR = path.resolve(__dirname, 'temp');
export const PUBLIC_DIR = path.resolve(__dirname, 'public');
export const SIZE = 1024 * 1024 * 100;

export const splitChunks = async (filename: string, size: number = SIZE) => {
    const filePath = path.resolve(__dirname, filename);
    const chunksDir = path.resolve(TEMP_DIR, filename);
    let stat = await fs.stat(filePath);
    let content = await fs.readFile(filePath);
    let current = 0;
    let i = 0;
    await fs.mkdirp(chunksDir);
    while (current < stat.size) {
        await fs.writeFile(
            path.resolve(chunksDir, filename + '-' + i),
            content.slice(current, current + size)
        );
        i++;
        current += size;
    }
}

const pipeStream = (filePath: string, writeStream: WriteStream) => new Promise(resolve => {
    const readStream = fs.createReadStream(filePath);
    readStream.on('end', async () => {
        await fs.unlink(filePath);
        resolve();
    });
    readStream.pipe(writeStream);
});
export const mergeChunks = async (filename: string, size: number = SIZE) => {
    const filePath = path.resolve(PUBLIC_DIR, filename);
    const chunksDir = path.resolve(TEMP_DIR, filename);
    const chunkFiles = await fs.readdir(chunksDir);
    chunkFiles.sort((a, b) => Number(a.split('-')[1]) - Number(b.split('-')[1]));
    await Promise.all(
        chunkFiles.map((chunkFile, index) => pipeStream(
            path.resolve(chunksDir, chunkFile),
            fs.createWriteStream(filePath, {
                start: index * size
            })
        ))
    );
    await fs.rmdir(chunksDir);
}
```

4.切片上传

```
import createError from 'http-errors';
import express, { Request, Response, NextFunction } from 'express';
import logger from 'morgan';
import { INTERNAL_SERVER_ERROR } from 'http-status-codes';
import cors from 'cors';
import path from 'path';
import { TEMP_DIR, PUBLIC_DIR, mergeChunks } from './utils';
import fs from 'fs-extra';
import multiparty from 'multiparty';
let app = express();
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cors());
app.use(express.static(path.resolve(__dirname, 'public')));
app.post('/merge', async (req: Request, res: Response) => {
    let { filename } = req.body;
    await mergeChunks(filename);
    res.json({
        success: true,
        url: `http://localhost:8000/${filename}`
    });
});
app.post('/upload/:filename/:chunk_name', async (req: Request, res: Response, _next: NextFunction) => {
    let file_dir = path.resolve(TEMP_DIR, req.params.filename);
    let exist = await fs.pathExists(file_dir);
    if (!exist) {
        await fs.mkdirs(file_dir);
    }
    const filePath = path.resolve(TEMP_DIR, req.params.filename, req.params.chunk_name);
    let writeStream = fs.createWriteStream(filePath, { start: 0, flags: "a" });
    req.pipe(writeStream);
    req.on('end', () => {
        writeStream.close();
        res.json({
            success: true
        });
    });
});

app.post('/upload', async (req: Request, res: Response, next: NextFunction) => {
    let form = new multiparty.Form();
    form.parse(req, async (err, fields, files) => {
        if (err) {
            return next(err);
        }
        let [filename] = fields.filename;
        let [chunk] = files.chunk;
        await fs.move(chunk.path, path.resolve(PUBLIC_DIR, filename), { overwrite: true });
        setTimeout(() => {
            res.json({
                success: true
            });
        }, 3000);
    });
});
app.use(function (_req, _res, next) {
    next(createError(404));
});

app.use(function (error: any, _req: Request, res: Response, _next: NextFunction) {
    res.status(error.status || INTERNAL_SERVER_ERROR);
    res.json({
        success: false,
        error
    });
});

export default app;
```

5.断点续传

```
import createError from 'http-errors';
import express, { Request, Response, NextFunction } from 'express';
import logger from 'morgan';
import { INTERNAL_SERVER_ERROR } from 'http-status-codes';
import cors from 'cors';
import path from 'path';
import { TEMP_DIR, PUBLIC_DIR, mergeChunks } from './utils';
import fs from 'fs-extra';
import multiparty from 'multiparty';
let app = express();
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cors());
app.use(express.static(path.resolve(__dirname, 'public')));
app.post('/merge', async (req: Request, res: Response) => {
    let { filename } = req.body;
    await mergeChunks(filename);
    res.json({
        success: true,
        url: `http://localhost:8000/${filename}`
    });
});
+app.post('/upload/:filename/:chunk_name/:start', async (req: Request, res: Response, _next: NextFunction) => {
    let start = isNaN(Number(req.params.start)) ? 0 : Number(req.params.start);
    let file_dir = path.resolve(TEMP_DIR, req.params.filename);
    let exist = await fs.pathExists(file_dir);
    if (!exist) {
        await fs.mkdirs(file_dir);
    }
    const filePath = path.resolve(TEMP_DIR, req.params.filename, req.params.chunk_name);
```

```
+    let writeStream = fs.createWriteStream(filePath, { start, flags: "a" });
    req.pipe(writeStream);
+    req.on('error', () => {
+        writeStream.close();
+    });
+    req.on('close', () => {
+        writeStream.close();
+    });
    req.on('end', () => {
        writeStream.close();
        res.json({
            success: true
        });
    });
});
+app.post('/verify', async (req: Request, res: Response): Promise => {
+    const { filename } = req.body;
+    const filePath = path.resolve(PUBLIC_DIR, filename);
+    let existFile = await fs.pathExists(filePath);
+    if (existFile) {
+        return res.json({
+            success: true,
+            needUpload: false
+        });
+    }
+    let tempFilePath = path.resolve(TEMP_DIR, filename);
+    let uploadedList: any[] = [];
+    let existTemporaryFile = await fs.pathExists(tempFilePath);
+    if (existTemporaryFile) {
+        uploadedList = await fs.readdir(tempFilePath);
+        uploadedList = await Promise.all(uploadedList.map(async (filename: string) => {
+            let stat = await fs.stat(path.resolve(tempFilePath, filename));
+            return {
+                filename,
+                size: stat.size
+            }
+        }));
+    }
+    res.json({
+        success: true,
+        needUpload: true,
+        uploadedList: uploadedList
+    });
+});
app.post('/upload', async (req: Request, res: Response, next: NextFunction) => {
    let form = new multiparty.Form();
    form.parse(req, async (err, fields, files) => {
        if (err) {
            return next(err);
        }
        let [filename] = fields.filename;
        let [chunk] = files.chunk;
        await fs.move(chunk.path, path.resolve(PUBLIC_DIR, filename), { overwrite: true });
        setTimeout(() => {
            res.json({
                success: true
            });
        }, 3000);
    });
});
app.use(function (_req, _res, next) {
    next(createError(404));
});

app.use(function (error: any, _req: Request, res: Response, _next: NextFunction) {
    res.status(error.status || INTERNAL_SERVER_ERROR);
    res.json({
        success: false,
        error
    });
});

export default app;
```