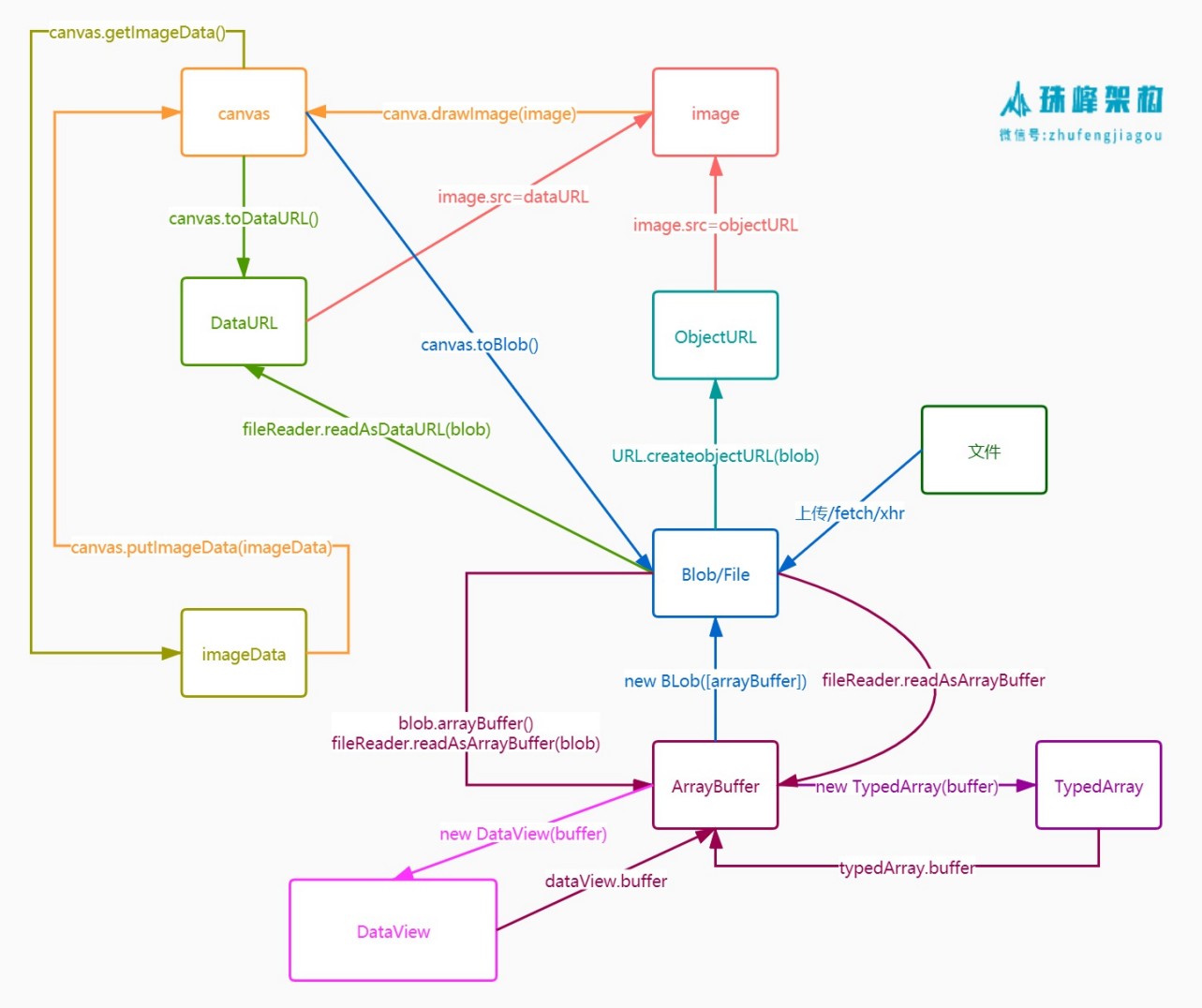
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=62 sentences=242, words=1406

1.生成项目

```
create-react-app zhufengbinary
cd zhufengbinary
yarn start
```



2.ArrayBuffer

- ArrayBuffer (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer)对象用来表示通用的、固定长度的原始二进制数据缓冲区
- 它是一个字节数组，通常在其他语言中称为 `byte array`
- 你不能直接操作 `ArrayBuffer` 的内容，而是要通过 `&#x7C7B;&#x578B;&#x6570;&#x7EC4;&#x5BF9;&#x8C61;`或 `DataView` 对象来操作，它们会将缓冲区中的数据表示为特定的格式，并通过这些格式来读写缓冲区的内容。

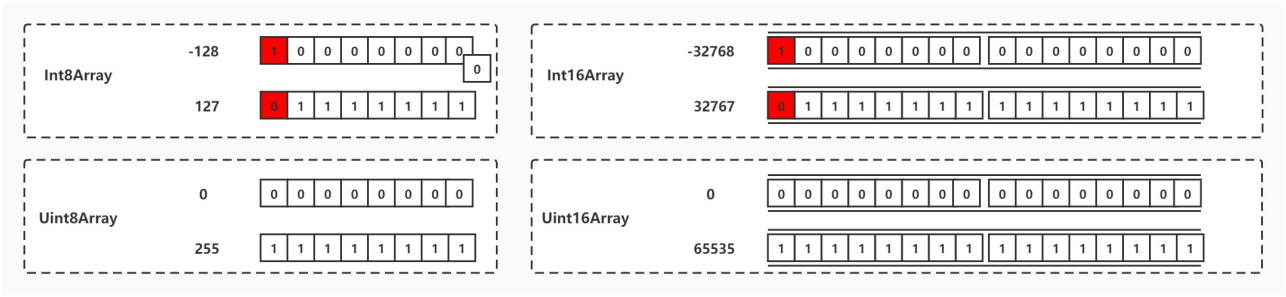
```
const buffer = new ArrayBuffer(8);
console.log(buffer.byteLength);
```

3.TypedArray

TypedArray (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/TypedArray)对象描述了一个底层的二进制数据缓冲区(binary data buffer)的一个类数组视图(view)

- 但它本身不可以被实例化，甚至无法访问，你可以把它理解为接口,它有很多的实现

类型 单个元素值的范围 大小(bytes) 描述 Int8Array -128 to 127 1 8 位二进制有符号整数 Uint8Array 0 to 255 1 8 位无符号整数 Int16Array -32768 to 32767 2 16 位二进制有符号整数 Uint16Array 0 to 65535 2 16 位无符号整数

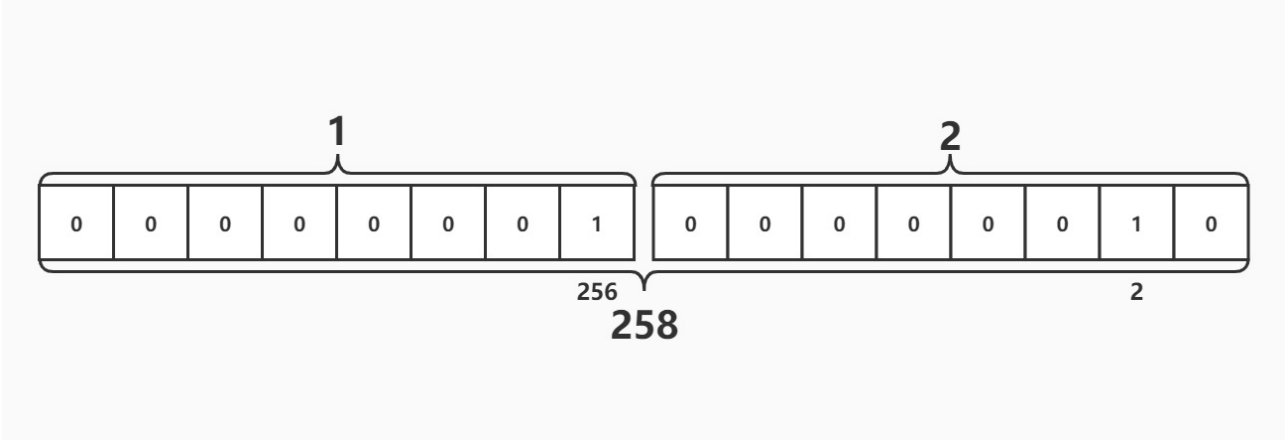



```
const buffer = new ArrayBuffer(8);
console.log(buffer.byteLength);
const int8Array = new Int8Array(buffer);
console.log(int8Array.length);
const int16Array = new Int16Array(buffer);
console.log(int16Array.length);
```

4.DataView对象

- DataView (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/DataView)视图是一个可以从二进制ArrayBuffer对象中读写多种数值类型的底层接口
- `setInt8()`从DataView起始位置以byte为计数的指定偏移量(byteOffset)处储存一个8-bit数(一个字节)
- `getInt8()`从DataView起始位置以byte为计数的指定偏移量(byteOffset)处获取一个8-bit数(一个字节)

```
new DataView(buffer [, byteOffset [, byteLength]])
```




```
const buffer = new ArrayBuffer(8);
console.log(buffer.byteLength);
const view1 = new DataView(buffer);
view1.setInt8(0, 1);
console.log(view1.getInt8(0));

view1.setInt8(1, 2);
console.log(view1.getInt8(1));

console.log(view1.getInt16(0));
```

5.Blob

- Blob (https://developer.mozilla.org/zh-CN/docs/Web/API/Blob)对象表示一个不可变、原始数据的类文件对象。Blob 表示的不一定是JavaScript原生格式的数据。File 接口基于Blob，继承了 blob 的功能并将其扩展使其支持用户系统上的文件。
- Blob() (https://developer.mozilla.org/zh-CN/docs/Web/API/Blob/Blob) 构造函数返回一个新的 Blob 对象。 blob的内容由参数数组中给出的值的串联组成。
- FileReader (https://developer.mozilla.org/zh-CN/docs/Web/API/FileReader) 对象允许Web应用程序异步读取存储在用户计算机上的文件（或原始数据缓冲区）的内容，使用 File 或 Blob 对象指定要读取的文件或数据。
    - `readAsText()`：读取文本文件（可以使用txt打开的文件），返回文本字符串，默认编码是UTF-8
    - `readAsDataURL()`：读取文件获取一段以data开头的字符串，这段字符串的本质就是DataURL，DataURL是一种将文件嵌入到文档的方案。DataURL是将资源转换为base64编码的字符串形式,并且将这些内容直接储存在url中
- 构造函数 `var aBlob = new Blob( array, options );`
    - array 是一个由 `ArrayBuffer`, `ArrayBufferView`, `Blob`, `DOMString` 等对象构成的 Array ，或者其他类似对象的混合体，它将会被放进 Blob。DOMStrings会被编码为UTF-8。
    - options 是一个可选的 `BlobPropertyBag`字典
        - `type` 默认值为 "",它代表了将会被放入到blob中的数组内容的MIME类型

```
<body>
    <script>
     let debug = { name: "zhufeng" };
     let str = JSON.stringify(debug);
     console.log("str", str);

     var blob = new Blob([str], { type: "application/json" });
     console.log("blob.size", blob.size);

        function readBlob(blob, type) {
            return new Promise((resolve) => {
                const reader = new FileReader();
                reader.onload = function (event) {
                    resolve(event.target.result);
                };
                switch (type) {
                    case "ArrayBuffer":
                        reader.readAsArrayBuffer(blob);
                        break;
                    case "DataURL":
                        reader.readAsDataURL(blob);
                        break;
                    case "Text":
                        reader.readAsText(blob,'UTF-8');
                        break;
                    default:
                        break;
                }
            });
        }
        readBlob(blob, "ArrayBuffer").then((buffer) => {
            console.log("buffer", buffer);
        });
        readBlob(blob, "DataURL").then((base64String) => {
            console.log("base64String", base64String);
        });
        readBlob(blob, "Text").then((text) => {
            console.log("text", text);
        });
    script>
body>
```

6. Object URL

- 可以使用浏览器新的 API URL 对象通过方法生成一个地址来表示 `Blob` 数据
- 格式为 `blob:<origin>/<uuid></uuid></origin>`
- URL.createObjectURL (https://developer.mozilla.org/zh-CN/docs/Web/API/URL/createObjectURL) 静态方法会创建一个 DOMString，其中包含一个表示参数中给出的对象的URL。这个 URL 的生命周期和创建它的窗口中的 document 绑定。这个新的URL 对象表示指定的 File 对象或 Blob 对象。
- revokeObjectURL (https://developer.mozilla.org/zh-CN/docs/Web/API/URL/revokeObjectURL) 静态方法用来释放一个之前已经存在的、通过调用 `URL.createObjectURL()` 创建的 URL 对象

```
<body>
    <button onclick="download()">下载jsonbutton>
    <script>
     function download () {
        let debug = { hello: "world" };
        let str = JSON.stringify(debug);
        var blob = new Blob([str], { type: 'application/json' });
        let objectURL = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.download = 'hello.json';
        a.rel = 'noopener';
        a.href = objectURL;
        a.dispatchEvent(new MouseEvent('click'));
        URL.revokeObjectURL(objectURL);
     }
    script>
body>
```

7.图片预览和裁剪上传

- bootstrap.min.css (https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css)

7.1 src\index.js

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

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

7.2 src\Image.js

src\Image.js

```
import React from 'react';
export default class Image extends React.Component{
   imageRef = React.createRef()   //图片
   canvasRef = React.createRef()  //完整的canvas
   avatarRef = React.createRef()  //裁剪后的头像
   state = {
    file:null,//添加的文件
    dataURL:'',//转换成的base64字符串
    times:1,//图片放大的倍数
    lastLeft:0,//最后一个左边的距离
    lastTop:0,//最后一个上面的距离
    avatarDataURL:''//头像的base64字符串
   }
   handleChange = (event)=>{//头像选择改变
    let file = event.target.files[0];
    let fileReader = new FileReader();//读取文件
```

```
    fileReader.onload =  (event)=> {
      this.setState({file,dataURL:event.target.result});
      this.imageRef.current.onload = ()=>this.draw();//绘制到canvas上
    };
    fileReader.readAsDataURL(file);
  }
  draw=(left=this.state.lastLeft,top=this.state.lastTop)=>{
    let image = this.imageRef.current;//图像
    let canvas = this.canvasRef.current;//canvas
    const ctx = canvas.getContext("2d");
    ctx.clearRect(0,0,canvas.width, canvas.height);//清掉老图片
    let imageWidth = image.width;//图片宽度
    let imageHeight = image.height;//图片高度
    if(imageWidth>imageHeight){//如果宽比高度大
      let times = canvas.width/imageWidth;
      imageWidth=canvas.width*this.state.times;//让宽度等于canvas宽度
      imageHeight=times*imageHeight*this.state.times;//然后让高度等比缩放
    }else{
      let times = canvas.height/imageHeight;
      imageHeight=canvas.height*this.state.times;
      imageWidth=times*imageWidth*this.state.times;
    }
    ctx.drawImage(image, (canvas.width-imageWidth)/2+left, (canvas.height-imageHeight)/2+top,imageWidth,imageHeight);
  }
  bigger = ()=>{
      this.setState({times:this.state.times+0.1},()=>this.draw());
  }
  smaller = ()=>{
    this.setState({times:this.state.times-0.1},()=>this.draw());
  }
  confirm = ()=>{
    let canvas = this.canvasRef.current;//头像canvas
    const ctx = canvas.getContext("2d");
    const imageData = ctx.getImageData(100, 100, 100, 100);//获取头像数据
    let clipCanvas = document.createElement('canvas');
    clipCanvas.width=100;
    clipCanvas.height=100;
    const clipContext = clipCanvas.getContext("2d");
    clipContext.putImageData(imageData, 0, 0);
    let dataUrl = clipCanvas.toDataURL();
    this.avatarRef.current.src = dataUrl;
    this.setState({avatarDataURL:dataUrl});
  }
  handleMouseDown = (event)=>{
    this.setState({startX:event.clientX,startY:event.clientY,startDrag:true});
  }
  handleMouseMove = (event)=>{
    if(this.state.startDrag)
        this.draw((event.clientX - this.state.startX)+this.state.lastLeft,(event.clientY - this.state.startY)+this.state.lastTop);
  }
  handleMouseUp = (event)=>{
    this.setState({lastLeft:(event.clientX - this.state.startX)+this.state.lastLeft,lastTop:(event.clientY -
this.state.startY)+this.state.lastTop,startDrag:false});
  }
  upload = ()=>{
    let bytes = atob(this.state.avatarDataURL.split(",")[1]);
    let arrayBuffer = new ArrayBuffer(bytes.length);
    let uint8Array = new Uint8Array(arrayBuffer);
    for (let i = 0; i < bytes.length; i++) {
      uint8Array[i] = bytes.charCodeAt(i);
    }
    let blob =  new Blob([arrayBuffer], { type: 'image/png' });
    let request = new XMLHttpRequest();
    let formData = new FormData();
    formData.append("name", "zhufeng");
    formData.append("avatar", blob);
    request.open("POST", 'http://localhost:8080/upload', true);
    request.send(formData);
  }
  render(){
      return (

                      {
                          this.state.file&&(

                          )
                      }

                      {
                          this.state.file&&(
                           <>

                                 this.bigger()}>放大
                                 this.smaller()}>缩小
                                 this.confirm()}>确定

                           </>
                          )
                      }

                      {
                          this.state.file&&(
                           <>

                              上传
                           </>
                          )
                      }

      )
  }
}
```

7.3 server.js

server.js

```
let express = require('express');
let path = require('path');
let cors = require('cors');
let app = express();
app.use(cors());
app.use(express.static(path.join(__dirname,'public')));
const multer = require('multer');
app.use(multer({dest: './uploads'}).single('avatar'));
app.post('/upload', function(req, res){
    res.json({success:true});
});
app.listen(8080,()=>{
    console.log('server started at port 8080');
});
```

8.音频的裁剪和预览

- ffmpeg (https://github.com/Kagami/ffmpeg.js)
- song (http://img.zhufengpeixun.cn/song.mp3)
- ffmpeg-worker-mp4.js (http://img.zhufengpeixun.cn/ffmpeg-worker-mp4.js)

8.1 index.js

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

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

8.2 src\Audio.js

src\Audio.js

```
import React from 'react';
import axios from 'axios';
export default class Audio extends React.Component {
  audioRef = React.createRef()
  audioClipRef = React.createRef()
  startRef = React.createRef()
  endRef = React.createRef()
  clip = async () => {
    this.worker = createWorker('/ffmpeg-worker-mp4.js');
    let response = await axios({ url: '/song.mp3', method: 'get', responseType: 'arraybuffer' });
    let originArrayBuffer = response.data;
    let start = parseInt(this.startRef.current.value);
    let end = parseInt(this.endRef.current.value);
    let duration = end - start;
    let resultArrayBuffer = (await toPromise(
      this.worker,
      getClipCommand(originArrayBuffer, start, duration)
    )).data.data.MEMFS[0].data;

    let clipBlob = audioBufferToBlob(resultArrayBuffer);
    let audio= this.audioClipRef.current
    audio.src = URL.createObjectURL(clipBlob);
    audio.load();
    audio.play();
  };
  render() {
    return (
      <div>
        <input ref={this.startRef} defaultValue={0} />
        <input ref={this.endRef} defaultValue={10} />
        <button type="button" className="primary" onClick={this.clip}>clipbutton>
        <audio ref={this.audioRef} controls src="/song.mp3">audio>
        <audio ref={this.audioClipRef} controls>audio>
      div>
    )
  }
}

function createWorker(workerPath) {
  return new Worker(workerPath);
}
function getClipCommand(arrayBuffer, start = 0, duration = 10) {
  return {
    type: "run",
    arguments: `-ss ${start} -i input.mp3 ${
      duration ? `-t ${duration} ` : ""
      }-acodec copy output.mp3`.split(" "),
    MEMFS: [
      {
        data: new Uint8Array(arrayBuffer),
        name: "input.mp3"
      }
    ]
  };
}
function toPromise(worker, info) {
  return new Promise((resolve) => {
    const onSuccess = function (event) {
      switch (event.data.type) {
        case "done":
          worker.removeEventListener("message", onSuccess);
          resolve(event);
          break;
        default:
          break;
      }
    };
    worker.addEventListener("message", onSuccess);
    info && worker.postMessage(info);
  });
}

function audioBufferToBlob(arrayBuffer) {
  const file = new File([arrayBuffer], 'test.mp3', {
    type: 'audio/mp3',
  });
  return file;
}
```