## 项目结构

|  |
| --- |
|  |

## 1.有FontLoader和TTFLoader，FontLoader直接加载字体的json文件，

|  |
| --- |
| import { useEffect } from 'react'  import './App.css'  import \* as THREE from 'three'  import {FontLoader,TextGeometry,TTFLoader} from './lib/addonutil'  import ThreeApp from './lib/threeapp'  function main1() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    //text1    let fontLoader = new FontLoader()    fontLoader.load(      'node\_modules/three/examples/fonts/droid/droid\_serif\_regular.typeface.json',      (droidFont) => {        let textGeometry = new TextGeometry('Threejs', {          size: 20,          height: 4,          font: droidFont        })        let textMaterial = new THREE.MeshNormalMaterial()        let text = new THREE.Mesh(textGeometry, textMaterial)        text.position.x = -45        text.position.y = 0        threeApp.scene.add(text)      })    //text2,ttfLoader    let ttfLoader = new TTFLoader()    ttfLoader.load('../fonts/jet\_brains\_mono\_regular.ttf', (json) => {      //注意ttfLoader需要调用fontLoader的parse功能      let jet\_brains\_font = fontLoader.parse(json)      let textGeometry2 = new TextGeometry('Hello People', {        size: 4,        height: 10,        font: jet\_brains\_font      })      let textMaterial2 = new THREE.MeshNormalMaterial()      let textMesh = new THREE.Mesh(textGeometry2, textMaterial2)      textMesh.position.x = -46      textMesh.position.y = 10      textMesh.name = 'text2'      threeApp.scene.add(textMesh)    })    function animate() {      requestAnimationFrame(animate);    }    animate()  }  async function createFont(loader) {    let font = await new Promise((resolve, reject) => {      loader.load(        'node\_modules/three/examples/fonts/droid/droid\_serif\_regular.typeface.json',        (droidFont) => {          resolve(droidFont)        })    })    return font  }  async function getTTFFont(ttfLoader, fontLoader) {    let font = await new Promise((resolve, reject) => {      ttfLoader.load('../fonts/jet\_brains\_mono\_regular.ttf', (json) => {        //注意ttfLoader需要调用fontLoader的parse功能        let jet\_brains\_font = fontLoader.parse(json)        resolve(jet\_brains\_font)      })    })    return font  }  async function main2() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    //text1    // let fontLoader = new AddOn.FontLoader()    let fontLoader =new FontLoader()    let font1 = await createFont(fontLoader)    let textGeometry = new TextGeometry('Threejs', {      size: 20,      height: 4,      font: font1    })    let textMaterial = new THREE.MeshNormalMaterial()    let text = new THREE.Mesh(textGeometry, textMaterial)    text.position.x = -45    text.position.y = 0    threeApp.scene.add(text)    //text2,ttfLoader    // let ttfLoader = new AddOn.TTFLoader()    let ttfLoader = new TTFLoader()    let ttfFont = await getTTFFont(ttfLoader, fontLoader)    let textGeometry2 = new TextGeometry('Hello People', {      size: 4,      height: 10,      font: ttfFont    })    let textMaterial2 = new THREE.MeshNormalMaterial()    let textMesh = new THREE.Mesh(textGeometry2, textMaterial2)    textMesh.position.x = -46    textMesh.position.y = 10    textMesh.name = 'text2'    threeApp.scene.add(textMesh)    function animate() {      textMesh.rotation.x += 0.01      textMesh.rotation.y += 0.01      requestAnimationFrame(animate);    }    animate()  }  function App() {    useEffect(() => {      // main1()      main2()    }, [])    return (      <>        <div>          <canvas id="myThreeJSCanvas"></canvas>        </div>      </>    )  }  export default App |

## TTFLoader能够把.ttf字体文件加载位json格式，然后调用FontLoader的parse方法来解析字体

|  |
| --- |
|  |

## 2.创建字体加载器后，在调用它的load方法时，需要传递一个回调函数，文字对象需要在这个回调函数里面创建标签添加到场景中

## 3.创建字体对象和创建其他3D对象是类似的，首先需要创建字体几何体对象，然后选择材质，再利用new THREE.Mesh(几何体对象,对象)来创建字体对象

## 4.然后需要把字体对象添加到场景中，注意所有这些操作都是在字体加载器的load方法的回调函数里面完成的。

# 完整代码，为了方便调用，我在lib里面新建了一个addonutil文件，在里面把常用的类先导入,还有，使用老师的代码里面的方法创建对象没有动感效果，而且它有一个很不好的地方，就是如果你需要在之后操作对象，会很难，所以我们把它改为Promise和async-await的写法，可以很方便操作物体，参考我们的main2函数，注意他是异步的，另外我们还有封装两个异步函数createFont和getTTFFont

## src/lib/addonutil.js

|  |
| --- |
| import {GUI} from 'three/examples/jsm/libs/lil-gui.module.min.js'  import {OrbitControls} from 'three/examples/jsm/controls/OrbitControls.js'  import {GLTFLoader} from 'three/examples/jsm/loaders/GLTFLoader.js'  import {FBXLoader} from 'three/examples/jsm/loaders/FBXLoader.js'  import {VOXLoader} from 'three/examples/jsm/loaders/VOXLoader.js'  import {TTFLoader} from 'three/examples/jsm/loaders/TTFLoader.js'  import { FontLoader } from 'three/examples/jsm/loaders/FontLoader';  import { TextGeometry } from 'three/examples/jsm/geometries/TextGeometry';  //很简单就是把这些类导入进来又导出，做的编写一次，使用多次的目的  export {    GUI,OrbitControls,GLTFLoader,FBXLoader,VOXLoader,TTFLoader,FontLoader,TextGeometry  } |

## src/lib/threeapp.js

|  |
| --- |
| import \* as THREE from 'three'  import { OrbitControls } from './addonutil'  import Stats from 'three/examples/jsm/libs/stats.module';  //注意：这是第二课的class，内容有点不一样所以我们把class名称改为ThreeApp2  //帧率使用聚光灯而不是平行光  export default class ThreeApp{      constructor(canvasId){          //1.定义场景，相机，渲染器          this.scene = undefined          this.camera = undefined          this.renderer = undefined          //2.定义相机参数,但是保存在本类中          this.fov = 45          this.nearPlane = 1          this.farPlane = 1000          this.canvasId = canvasId          //3.定义额外组件          this.clock = undefined          this.controls = undefined          this.stats = undefined          //4.定义环境光和聚光灯          this.ambientLight = undefined          this.directionalLight = undefined          this.fonts = []      }      initApp(){          //创建场景对象并且赋值给成员变量          this.scene = new THREE.Scene()          //创建相机对象并且用相机成员变量接收          this.camera = new THREE.PerspectiveCamera(              this.fov,              window.innerWidth/window.innerHeight,              this.nearPlane,              this.farPlane          )          //调整相机位置          this.camera.position.z = 120 //cameraz轴值越大，图像越小          //根据传入的id获取画布对象          let canvas = document.getElementById(this.canvasId)          //创建渲染器          this.renderer = new THREE.WebGLRenderer({               canvas,               antialias:true           })            //设置渲染器的渲染尺寸          this.renderer.setSize(window.innerWidth,window.innerHeight)          //添加到body中          document.body.appendChild(this.renderer.domElement)          //创建时钟，轨道控制器，检测帧数(FPS)的工具          this.clock = new THREE.Clock()          this.controls = new OrbitControls(this.camera,this.renderer.domElement)          this.stats = Stats()          //将检测帧数(FPS)的工具添加到body中          document.body.appendChild(this.stats.domElement)          //创建环境光          this.ambientLight = new THREE.AmbientLight(0xffffff,0.5)          this.ambientLight.castShadow = true          //把环境光添加到场景中          this.scene.add(this.ambientLight)          //创建方向光          this.directionalLight = new THREE.SpotLight(0xffffff,1)          //设置方向光的位置          this.directionalLight.position.set(0,32,64) //位置也不一样            //添加方向光到场景中          this.scene.add(this.directionalLight)          //给window对象添加事件监听，用来实现窗口的响应式功能          window.addEventListener('resize',()=>this.onWindowResize(),false)      }        //定义animate成员函数，这个函数在外部调用      animate(){          window.requestAnimationFrame(this.animate.bind(this))          this.render()          this.stats.update()          this.controls.update()      }      render(){          this.renderer.render(this.scene,this.camera)      }      onWindowResize(){          //重新计算相机的宽高比          this.camera.aspect = window.innerWidth/window.innerHeight          //更新相机的投影矩阵          this.camera.updateProjectionMatrix()          //重新设置渲染器的渲染大小          this.renderer.setSize(window.innerWidth,window.innerHeight)      }  } |

## src/App.jsx

|  |
| --- |
| import { useEffect } from 'react'  import './App.css'  import \* as THREE from 'three'  import {FontLoader,TextGeometry,TTFLoader} from './lib/addonutil'  import ThreeApp from './lib/threeapp'  function main1() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    //text1    let fontLoader = new FontLoader()    fontLoader.load(      'node\_modules/three/examples/fonts/droid/droid\_serif\_regular.typeface.json',      (droidFont) => {        let textGeometry = new TextGeometry('Threejs', {          size: 20,          height: 4,          font: droidFont        })        let textMaterial = new THREE.MeshNormalMaterial()        let text = new THREE.Mesh(textGeometry, textMaterial)        text.position.x = -45        text.position.y = 0        threeApp.scene.add(text)      })    //text2,ttfLoader    let ttfLoader = new TTFLoader()    ttfLoader.load('../fonts/jet\_brains\_mono\_regular.ttf', (json) => {      //注意ttfLoader需要调用fontLoader的parse功能      let jet\_brains\_font = fontLoader.parse(json)      let textGeometry2 = new TextGeometry('Hello People', {        size: 4,        height: 10,        font: jet\_brains\_font      })      let textMaterial2 = new THREE.MeshNormalMaterial()      let textMesh = new THREE.Mesh(textGeometry2, textMaterial2)      textMesh.position.x = -46      textMesh.position.y = 10      textMesh.name = 'text2'      threeApp.scene.add(textMesh)    })    function animate() {      requestAnimationFrame(animate);    }    animate()  }  async function createFont(loader) {    let font = await new Promise((resolve, reject) => {      loader.load(        'node\_modules/three/examples/fonts/droid/droid\_serif\_regular.typeface.json',        (droidFont) => {          resolve(droidFont)        })    })    return font  }  async function getTTFFont(ttfLoader, fontLoader) {    let font = await new Promise((resolve, reject) => {      ttfLoader.load('../fonts/jet\_brains\_mono\_regular.ttf', (json) => {        //注意ttfLoader需要调用fontLoader的parse功能        let jet\_brains\_font = fontLoader.parse(json)        resolve(jet\_brains\_font)      })    })    return font  }  async function main2() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    //text1    // let fontLoader = new AddOn.FontLoader()    let fontLoader =new FontLoader()    let font1 = await createFont(fontLoader)    let textGeometry = new TextGeometry('Threejs', {      size: 20,      height: 4,      font: font1    })    let textMaterial = new THREE.MeshNormalMaterial()    let text = new THREE.Mesh(textGeometry, textMaterial)    text.position.x = -45    text.position.y = 0    threeApp.scene.add(text)    //text2,ttfLoader    // let ttfLoader = new AddOn.TTFLoader()    let ttfLoader = new TTFLoader()    let ttfFont = await getTTFFont(ttfLoader, fontLoader)    let textGeometry2 = new TextGeometry('Hello People', {      size: 4,      height: 10,      font: ttfFont    })    let textMaterial2 = new THREE.MeshNormalMaterial()    let textMesh = new THREE.Mesh(textGeometry2, textMaterial2)    textMesh.position.x = -46    textMesh.position.y = 10    textMesh.name = 'text2'    threeApp.scene.add(textMesh)    function animate() {      textMesh.rotation.x += 0.01      textMesh.rotation.y += 0.01      requestAnimationFrame(animate);    }    animate()  }  function App() {    useEffect(() => {      // main1()      main2()    }, [])    return (      <>        <div>          <canvas id="myThreeJSCanvas"></canvas>        </div>      </>    )  }  export default App |

### 效果：在App函数的钩子里面调用main2函数，可以在浏览器中看到此时小文字会旋转

|  |
| --- |
|  |
|  |
|  |