**CloudMusic项目复盘**

1. **遇到的问题：**
2. **零碎知识点：**
3. **拼接字符串：**

“${xx}” 便捷点在于不用再像以前那样用多对双引号拼接多对字符串了

1. **Styled-component声明全局样式：**

export const GlobalStyle = createGlobalStyle``

通过createGlobalStyle声明全局样式，然后在项目根文件中导入即可

function App() {

  return (

    <Provider store={store}>

      <HashRouter>

        <GlobalStyle></GlobalStyle>

        <IconStyle></IconStyle>

        { renderRoutes(routes) }

      </HashRouter>

    </Provider>

  )

}

1. **模板字符串：**

ES6出现的，可以用来携带字符串，用反引号（`）表示。

用处1：保证多行字符串的原样式，即保留空格、换行等等。

用处2：配合${}使用，在字符串中嵌套变量或者函数运算

1. **ES6对象的简洁表示：**

例：const x='';

  const y=function(){};

  const z={

    x,

    y,

    "j":""

  }

这里就传入了三种类型：已有对象、已有方法、新定义对象

1. **路由传参：**

通过使用通配符向路由对应的组件传递参数，组件根据拿到的参数动态变换显示内容，同时地址栏的地址与传入的参数保持一致。

例：

  {

        path: "/recommend/",

        component: SuspenseComponent (RecommendComponent),

        routes: [

          {

            path: "/recommend/:id",

            component: SuspenseComponent (AlbumComponent)

          }

        ]

      },

Album组件通过props.match.params.id获取传入的歌单id，请求对应的歌曲资源渲染页面，同时路由也变成/recommend/:id，如下图

1. **CSS样式最高优先级！Important**

例：

font-family: "iconfont" !important;

！important可以突破css引用的自然优先级，优先使用！Important标记的样式

1. **导航组件的保留：**

想要在导航以后组件还在的话只要将导航组件声明在导向页面的上级路由中即可。

1. **扩大点击区域：**

position: relative;

    &:after {

      content: '';

      position: absolute;

      top: -10px; bottom: -10px; left: -10px; right: -10px;

    };

原理：通过没有宽度和高度的伪元素，设置他的top、bottom、left、right，这四个量的值都是相对父元素的。话又说来，父元素不就是需要扩大点击区域的元素吗，他的四维相对于父元素都放大了10px，那整体就胖了一圈。

1. **’>’选择器**

'>'代表后代第一层选择器

1. **Flex布局常用属性：**

**主轴方向：flex-direction**

**flex-direction**: row | row-reverse | column | column-reverse;

row（默认值）：主轴为水平方向，起点在左端。

row-reverse：主轴为水平方向，起点在右端。

column：主轴为垂直方向，起点在上沿。

column-reverse：主轴为垂直方向，起点在下沿。

**项目溢出换行：flex-wrap**

**flex-wrap**: nowrap | wrap | wrap-reverse;

nowrap（默认）：不换行。

wrap：换行，第一行在上方。

wrap-reverse：换行，第一行在下方。

**★flex-flow：flex-flow**属性是flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。

**★主轴对齐方式：justify-content**

**justify-content**: flex-start | flex-end | center | space-between | space-around;

flex-start（默认值）：左对齐

flex-end：右对齐

center： 居中

space-between：两端对齐，项目之间的间隔都相等。

space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

**★交叉轴对齐方式：align-items**

**align-items**: flex-start | flex-end | center | space-between | space-around | stretch;

flex-start：交叉轴的起点对齐。

flex-end：交叉轴的终点对齐。

center：交叉轴的中点对齐。

baseline: 项目的第一行文字的基线对齐。

stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。

**项目属性：**

**flex-grow**属性定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。flex-grow: <number>

**flex-shrink**属性定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。flex-shrink: <number>

**flex-basis**属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为auto，即项目的本来大小。flex-basis: <length> | auto;

**★flex**属性是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]

**★align-self**属性允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch。align-self: auto | flex-start | flex-end | center | baseline | stretch;

1. **immutable的多层级查询：**

 {a:1}

immutableData.get('a') 得到1。

{a:{b:2}}

immutableData.getIn(['a', 'b']) 得到2。

1. **Swiper使用方法：**

  1.创建swiper容器，类名设置为‘slider-container’

  2.在useeffect中实例化swiper对象，并配置好初始化的参数，将类名传入到该对象中去，以供其定位

  3.在swiper容器中创建div，类名‘swiper-wrapper’，并在该div中插入轮播图图片数组，图片数组的使用需按照固定格式来，做好相关配置即可

**13.元素高度百分比设置无效的问题**

元素的高度默认是缺省值，所以特别要注意当高度设置为百分比时，父元素的高度要确定，否则设置无效，尤其是在height：100%时，最终可能会追溯到设置html，body的高度为100%

**14.display属性对布局带来的影响**

**块级元素：**每个块级元素通常都会独占一行或者是多行，可以对其单独设置高度,宽度以及对齐等属性。

**特点：**

块级元素会独占一行

高度，行高，外边距和内边距都可以单独设置

宽度默认是容器的100%

可以容纳内联元素和其他的块级元素

**行内元素：**不占有独立的区域，仅仅依靠自己的字体大小或者是图像大小来支撑结构。一般不可以设置宽度，高度以及对齐等属性。

**特点：**

和相邻的行内元素在一行上

高度和宽度无效，但是水平方向上的padding和margin可以设置，垂直方向上的无效

默认的宽度就是它本身的宽度

行内元素只能容纳纯文本或者是其他的行内元素（a标签除外）

#### **行内块级元素：**

和相邻的行内元素（行内块）在一行上，但是中间会有空白的间隙

默认的宽度就是本身内容的宽度

高度，行高，内边距和外边距都可以设置

**15.react-lazyload使用方法**

1.在图片的外面包裹一层LazyLoad，并设置placeholder属性为要懒加载时显示的内容（可以是图片或者其他）

2.向scroll的onscroll属性传入执行函数，当滑动的时候执行该函数，即传入forcecheck函数，作用是解除懒加载状态显示图片，实现手段是。。。有待考证

forcecheck

**16.占位容器的技巧**

容器的margin\padding使用百分比时是以父元素的width为基准的，所以'padding-top: 100%'就是父元素的width，同时不能使用margin的理由是如果用了margin容器就没有高度了，但是用padding有，因为默认的元素的高度是content的高度+padding的高度，如果用了margin，高度会受到挤压，反之padding则不会

**17.利用背景阴影做衬底的技巧**

background: linear-gradient(hsla(0,0%,43%,.4),hsla(0,0%,100%,0));

hsla(hue, saturation, lightness, alpha)

hue- 色相:范围0-360，其中0 (或 360) 为红色, 120 为绿色, 240 为蓝色

saturation - 饱和度：定义饱和度; 0% 为灰色， 100% 全色

lightness - 亮度：定义亮度 0% 为暗, 50% 为普通, 100% 为白

alpha - 透明度：定义透明度 0（透完全明） ~ 1（完全不透明）

linear-gradient(direction, color-stop1, color-stop2, ...)

direction：  用角度值指定渐变的方向（或角度）。

color-stop1, color-stop2,...：用于指定渐变的起止颜色。

**18.better-scroll使用步骤：**

1.创建一个div作为betterscroll的容器

2.创建一个bscroll对象，配置好bscroll的各项参数，并将该对象绑定到该div容器上

3.在useeffect中监听bscroll的各种状态，相应的做出相应的操作

**19.useImperativeHandle：**

useImperativeHandle限定了外部对ref定位到的元素的操作方式，只限定在useImperativeHandle提供了的操作方法范围内

本质是外部传入的ref只定位到函数组件，通过外部传入函数组件的ref调用useImperativeHandle给出的方法控制函数组件内部元素

需配合转发ref使用

1. **PropTypes：**

Proptypes帮助组件进行类型检查

例：

import PropTypes from 'prop-types';

class Greeting extends React.Component {

  render() {

    return (

      <h1>Hello, {this.props.name}</h1>

    );

  }

}

Greeting.defaultProps

= {

  name: ”吴灏”

};

Greeting.propTypes = {

  name: PropTypes.string

};

[https://zh-hans.reactjs.org/docs/typechecking-with-proptypes.html#\_\_\_gatsby](https://zh-hans.reactjs.org/docs/typechecking-with-proptypes.html" \l "___gatsby)

1. **axios使用步骤：**

    1.创建一个axios实例并配置axios相关属性，export出去以供调用

    2.调用上一步创建好的axios实例，使用它的get，post...方法

3.可以设置拦截器在axios请求发出前和接收后对数据进行处理

export const baseUrl = 'http://111.229.243.7:3000/';

const axiosInstance = axios.create ({

  baseURL: baseUrl

});

axiosInstance.interceptors.response.use (

  res => res.data,

  err => {

    console.log (err, "网络错误");

  }

);

1. **immutable的set和fromJS方法：**

在immutable.js中，object经过fromjs函数以后默认转成map，array经过fromjs以后默认转成list

set方法在list和map中的使用：

   list：list.set(index, xx);

   map：map.set('key',xx)

1. **immutable的fromJS**

对于基本值类型无需进行fromjs转化即可存入immutable数据结构中了，因为fromJS的作用是将 JS 对象转换为 immutable 对象

1. **react-thunk源码分析**
2. **CSS动画的编写**

keyframes是专门用来写动画用的. "from" 和 "to"，等同于 0% 和 100%，0% 是动画的开始，100% 是动画的完成.

animation要规定至少两个属性：动画名称、动画时长

步骤：

1.编写keyframes

@keyframes myfirst

{

from {background:red;}

to {background:yellow;}

}

2.在animation属性调用keyframes

div

{

animation:myfirst 5s;

}

1. **CSS过渡属性transition使用：**

语法：

transition: property duration timing-function delay;

[property](https://links.jianshu.com/go?to=http://www.runoob.com/cssref/css3-pr-transition-property.html" \t "https://www.jianshu.com/p/_blank):规定应用过渡的 CSS 属性的名称(默认为all)。(可选)

[duration](https://links.jianshu.com/go?to=http://www.runoob.com/cssref/css3-pr-transition-duration.html" \t "https://www.jianshu.com/p/_blank):定义过渡效果花费的时间。默认是 0s。（必需）

[timing-function](https://links.jianshu.com/go?to=http://www.runoob.com/cssref/css3-pr-transition-timing-function.html" \t "https://www.jianshu.com/p/_blank):规定过渡效果的时间曲线。默认是 "ease"函数。(可选)

[delay](https://links.jianshu.com/go?to=http://www.runoob.com/cssref/css3-pr-transition-delay.html" \t "https://www.jianshu.com/p/_blank):规定过渡效果何时开始。默认是 0s.(可选)

[timing-function](https://links.jianshu.com/go?to=http://www.runoob.com/cssref/css3-pr-transition-timing-function.html" \t "https://www.jianshu.com/p/_blank)表：

| 值 | 描述 |
| --- | --- |
| linear | 规定以相同速度开始至结束的过渡效果，即匀速。（等于 cubic-bezier(0,0,1,1)）。 |
| ease | 规定慢速开始，然后变快，然后慢速结束的过渡效果（cubic-bezier(0.25,0.1,0.25,1)）。 |
| ease-in | 规定以慢速开始的过渡效果（等于 cubic-bezier(0.42,0,1,1)）。 |
| ease-out | 规定以慢速结束的过渡效果（等于 cubic-bezier(0,0,0.58,1)）。 |
| ease-in-out | 规定以慢速开始和结束的过渡效果（等于 cubic-bezier(0.42,0,0.58,1)）。 |
| cubic-bezier(n,n,n,n) | 在 cubic-bezier 函数中定义自己的值。可能的值是 0 至 1 之间的数值。 |

例子：

div

{

transition: width 2s;

}

1. **CSS transform属性**
2. **CSS xx:nth-child(y)属性**

规定xx元素的父元素的第y个子元素的属性

例：这里就是让包裹2的span标签字体变红

<style>

div:nth-child(2){

Color:red

}

</style>

<div>

<span>1</span>

<span>2</span>

</div>