**记录-record**

**2021.5.16**

1. 验证二叉搜索树

2.恢复二叉搜索树

3.相同的树

核心： 使用中序树来进行判断root.left>>root >>root.right

4.

React 视频学习todoMvc demo

React.fragement setState({}) e.preventDefault() e=> this,get(e)

解构赋值 findIndex 返回第一个符合条件否则就是underfined

Let {} = {};

数组对象去重

const arr = [{id: 1, name:’123’},{id: 1, name:’1234}]

let map = new Map(); arr.filter() !map.has(id) && map.set(id, 1);

hashChange

dva基于react和reduce

window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_())是为了能在谷歌浏览器中用redux devtools调试工具。

**2021.5.17**

1. 对称二叉树
2. 层序遍历
3. 锯齿形遍历
4. 最大深度

IOS橡皮效果去除

1. window.addEventListener('touchmove', function(event) {
2. event.preventDefault();
3. }, false);
4. window.addEventListener('touchstart', function(event) {
5. event.preventDefault();
6. }, false);

字体抗锯齿 font-smoothing: antialiased;

Dva框架使用，reducer react-saga react-router结合起来的

**2021.5.18**

1. 前序和中序还原二叉树

2. 后序和中序还原二叉树

3．二叉树的遍历2 reverse

4. 数组还原为二叉树

5. 链表还原二叉树

DVA学习

    const app = dva();

    app.model({

      namespace: 'index',

      state: 0,

      reducers: {

        add(state) {

          return state + 1;

        }

      },

      effects: {

        \*adds(action, {call, put}) {

          yield call(delay, 1000);

          yield put({type: 'add'});

        }

      }，

Subscription: {

Keys({dispatch}) {

Dispatch({type:’add’});

}

}

    });

    app.router(() => <Home/>);

    app.start('#root');

Effect 是一个 Generator 函数，内部使用 yield 关键字，标识每一步的操作（不管是异步或同步）。

闭包

为一个页面添加几个用于调整字体大小的按钮。一种方法是以像素为单位指定 body 元素的 font-size，然后通过相对的 em 单位设置页面中其它元素（例如页眉）的字号

      function makeSizes(size) {

        return function() {

          document.body.style.fontSize = size + 'px';

        }

      }

      document.getElementById('ids').onclick = makeSizes(12);

document.getElementById('ids').onclick = makeSizes(142);

document.getElementById('ids').onclick = makeSizes(16);

**用闭包模拟私有方法**

      var counter = (function () {

        var privateCounter = 0;

        function changeBy(val) {

          privateCounter += val;

        }

        return {

          increment: function () {

            changeBy(1);

          },

          decrement: function () {

            changeBy(-1);

          },

          value: function () {

            return privateCounter;

          },

        };

      })();

      console.log(counter.value()); // logs 0

      counter.increment();

      counter.increment();

      console.log(counter.value()); // logs 2

      counter.decrement();

      console.log(counter.value()); // logs 1

**2021.5.19**

1. 平衡二叉树

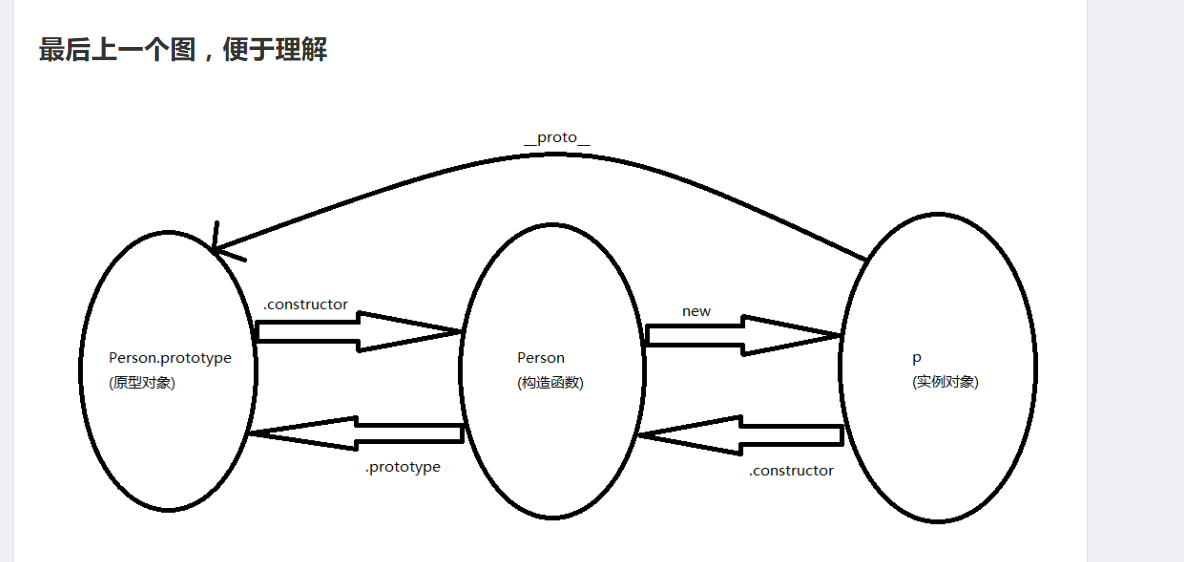
2. 最小深度

3. 路径总和

4. 路径总和2

5. 二叉树展开为链表

**2021.5.20**



function Person(name){

  this.name = name;

}

  var p = new Person("jack");

  console.log(p.\_\_proto\_\_===Person.prototype);//true

  console.log(p.\_\_proto\_\_===p.constructor.prototype);//true

常用的继承 构造函数和原型集成

      function Person() {

        this.name = 'wangbo';

      }

      function Child() {

        this.age = '11';

      }

      Child.prototype = new Person();

      var child = new Child();

      alert(child.name);

      var obj=''; Object.prototype.toString.call(obj)

**2021.5.21**

1. 不同的子序列

2. 填充子节点到右侧指针

3. II

4. 杨辉三角

5. II

created : function(){

var \_this = this

axios.get('static/data.json').then(function(res){

console.log(this) //undefined

console.log(\_this) //VueComponent {\_uid: 1, \_isVue: true, $options: {…}, \_renderProxy: Proxy, \_self: VueComponent, …}

\_this.user = res.data.user

})

}

在created下的函数this指向的是当前创建的vue实例，而在这些函数内部使用例如axios与后台交互后回调函数的内部的this并非指向当前的vue实例;

若想拿到后台回传的数据更新data里的数据，不能在回调函数中直接使用this，而要用在外部函数定义的变量存储的this，也就是当前vue的实例。

var express = require('express');

var proxy = require('http-proxy-middleware');

var app = express();

app.use('/api', proxy({target: 'http://10.119.168.87:4000', changeOrigin: true}));

app.listen(3000);

说明：我们利用express在3000端口启动了一个小型的服务器，利用了app.use('/api', proxy({target: 'http://10.119.168.87:4000/', changeOrigin: true}))这句话，使发到3000端口的/api请求转发到了4000端口。即请求http://localhost:3000/api相当于请求http://10.119.168.87:4000/api。

函数的封装

可以通过立即执行函数表达式（IIFE），来达到隐藏细节的目的

这样在模块外部无法修改我们没有暴露出来的变量、函数

缺点：功能相对较弱，封装过程增加了工作量、仍会导致命名空间污染可能、闭包是有成本的。

export myFun = (fucntion() {

  //

})();

单例模式

  class Singleton {

    constructor(name) {

        this.name = name;

        this.instance = null;

    }

    // 构造一个广为人知的接口，供用户对该类进行实例化

    static getInstance(name) {

        if(!this.instance) {

            this.instance = new Singleton(name);

        }

        return this.instance;

    }

}

  3.网站的登录弹窗，无论再别的地方点击多少次，都只会弹出一个登录弹窗

4.购物车的场景