# J S 相关笔记

(附加){CSDN 开发网站 github 代码网站}

1. js 都是在script标签内执行的
2. Var：varble，变量 = 赋值；代码结束；变量在使用之前必须先声明
3. 驼峰命名法：小驼峰：首字母小写，其余单词首字母大写，大驼峰：所有首字母大写，

字母 + 数字 + 下划线 ，需要做到见名知意

1. Alert（） alert ，出现一个弹出框 框内文本为（）里的内容
2. Age = age- 1 数字类型 将age的值-1 重新赋予age
3. 自增自减运算：

count ++ ；先使用count的值，然后count + 1 ；count = count + 1；

++ count ； 先让count + 1 ，然后再使用count的值； count = count + 1

1. console.log（log）：console：输出，log：日志，在计算机的控制台输出日志消息；

Console.log里面可以写变量，公式

1. 一个变量可以多次重复的声明，但是不建议这种写法
2. ReferenceError:XX is not defined；reference 关系，error 错误 ，is 是， not 不 ，defined 定义；关系引用错误：使用了一个没有定义过的变量XX
3. % 求余： 10 % 10 ;任意一个数X求余其他的数Y 余数为（Y-1 y-2 直到0）
4. var content = "1"

var name = '小美'；string 字符串类型(叫什么都可以但必须加''或"")

1. 字符串使用 + ，表示拼接 （‘9’ + ‘1’ 结果显示‘91’）
2. 字符串和数字相加，结果为字符串类型（string）
3. Console.log（typeof age）typeof查看变量的类型，后面直接写变量名称；
4. 布尔类型，第一个值为true 真，第二个值为false 假
5. 三目表达式/三元运算 ：X > Y ? 变量1 ：变量2
6. Var XX = ‘aa’ = 表示赋值，将= 右边的值赋予左边的变量
7. == 逻辑等 表示判断相等（在判断的时候，如果遇到类型不一致的情况下，会将数据转 化成同样类型，再做判断）逻辑等，忽视变量类型

=== 逻辑等 既判断类型 也判断类型的值

1. 字符串总结： + 表示拼接

字符串 + 字符串 = 字符串 ； 字符串 + 数字 = 字符串

数字 + 数字 = 数字 字符串类型数字 + 字符串类型数字 = 字符串

字符串类型数字 + 数字 = 字符串 字符串类型数字 / 字符串类型数字 = 数字

字符串类型数字 - 字符串类型数字 = 数字 字符串类型数字 \* 字符串类型数字 = 数字

‘abcf’/ ‘123’ = NaN 字符串 / 字符串类型数字 = NaN

-- 必须为字符串类型的数字才可以减

1. js里面代码执行有三种顺序：顺序执行 循环执行 分支执行
2. 顺序执行 ： 条件判断 从上往下顺序判断 有一个条件满足就不再判断下面的条件
3. 循环执行 ： 重复的执行某一段代码
4. If(xxxxx){},else if {xxxx} else {xxxxx}
5. && : and 表示而且 ； 只要有一个条件为假，即为假
6. || ： or 表示或者 ； 只要有一个条件为真 ， 即为真
7. 循环条件表达式 ： 初始化语句 index = 0 ； 限定范围 index < 3 ； 增量 index ++

For(index = 0 ; index < 3 ; index ++){ }

双循环 For(index = 0 ; index < 3 ; index ++){ For(index = 0 ; index < 3 ; index ++){ } }

1. While循环 当条件满足执行{ }里面的内容，条件不满足则不执行
2. do{ }while( ) 不管条件是否满足（为真） 都会先执行do里面的内容，如果条件满足， do里面的内容会一直执行，直到不再满足条件，如果条件不满足，do里面的内容只会 执行一次
3. 随机数 var num = Math . Random（） 计算机当中的随机数 称之为 伪随机数
4. Case 如果值为0 break 打断 必须要写 固定格式

Switch（ ）{ case xx : 0 { console.log( 石头 ) break} case xx : 1 { console.log( 剪子 ) case xx : 2 { console.log( 布 ) }

1. break ： 会跳出当前的循环 剩下的循环次数不再执行
2. Continue ： 继续； 程序遇到continue的时候 会跳出本次循环，继续执行下次循环， 直到所有循环结束 （跳出这次循环，剩余的循环继续执行）
3. Return : 跳出方法 ，必须写在函数方法内，执行到return 剩下的代码不再执行
4. Var result = confirm（‘xxxxx’） 网页弹出框显示文本内容 有确定，取消
5. Var result = prompt （‘xxxx’） 弹出框 + 文本框 输入什么就得到什么
6. 方法 ； 函数 function功能

Function固定公式 ； 方法名称 驼峰 见名知意 ； （ ）参数 ； { }方法具体的实现

1. 声明方法 无参；无返回值方法 function test（ ） { }
2. 有参方法 声明方法时候所使用的参数，称之为 形参（形式参数），方法后 实参要与 形参的个数保持一致

Function test（a , b）{ console.log(‘计算结果’ + （a , b） ) }

a b 如果不被（ ）括住 ，那么会先后变成字符串而不是数字之和

Test (10 , 20 ) test (‘小明’ ，’小美’ ) 调用方法时候的参数 叫做实参（有具体值）

1. 全局变量： 从声明该变量的这一行，到代码结束，都是他的作用范围/ 作用域；

全局变量可以在函数当中使用

获取全局变量 全局变量默认前面+ window

1. 局部变量 ： 方法内部声明的变量 ；只在声明变量的方法体内有效；出了范围无效
2. 带有返回值的函数 function test（）{ var result = xx \* xx \* xx return result }

Return result 将result的值返回给方法外面

Var sum= test 接受方法内部返回到外面的值

带参数 带返回值的函数也通过

1. return 返回 将方法内部的结果返回给方法外面

44.对象 声明一个变量 对象包含属性，属性可以任意类型，

变量内部，用：（冒号）显示值

1. 数组 ；var array = [1 , 2 , 3 , 4 , 5]可以为数字，字符串
2. 长度 ；array . length 获取数字的元数个数总数，个数从1 开始计算
3. 索引 ； array[ 0 ] 找到第几个元素 ， 从零开始数
4. Array . push（ ） 在数组的后面添加一个新值 push推 可推到新的数组
5. Array . unshift （ ） 往数组当中的第一位添加一个新值
6. Array . pop （ ） 删除数组的最后一位
7. Array . shift （ ） 删除数字的第一位
8. Array . reverse （ ） 数组相反排序 [1 , 2 , 3 , 4 ]变为[4 , 3 , 2 , 1 ]
9. Array = [1 , 2 , 3 , 4 ] var array1 = array . slice（2 ， 4 ）

从指定位置开始复制（包含该位置），复制到指定位置（不包含该位置）

1. Var array2 = array . splice（2 , 4）

从指定位置开始剪切 ， 剪切指定 个数

Var array2 = array . splice （2 ， 4 ‘xx’）

用指定部分‘xx’来替换剪切掉的内容 ； 将内容插到指定位置处

1. Array . toString （） 将数组转化成字符串
2. Array . toLocaleString（）将数组转化成本地字符串
3. Var str = ‘a , b , c , d , e , f,’ 变量为字符串
4. Str . toUpperCase（） 将字符串全部转化为大写
5. Str . toLowerCase（） 将字符串全部转化为小写
6. Str . charAt（3） 返回指定索引3所对应的字符
7. Str . charCodeAt （） 字符在ASCII 所对应的编号
8. Str . indexOf （） 查看指定字符在字符串当中的索引，不存在显示-1，存在则显示第一个被找到的字符的位置
9. Str . substr（3 ， 5） 从指定位置开始复制，复制指定的长度
10. Str . substring （3 ， 5） 从指定位置开始复制（包含该位置），复制到指定位置（不包含该位置）
11. str . split ( ‘ ? ’ ) 分隔字符串 将( ‘ ? ’ ) 改为 ( ‘ , ’ ) 对象序列化
12. $ ( ‘ #kind ’ ) . children ( ) . eq (index ) 找到每一个子标签
13. 冒泡排序

Var num= [6 , 7 , 3 , 5 , 9 ]

For （index = 0 ; index < num . length - 1 ; index ++ ）

{

For （var j = 0 ; j < num . length - 1 - index ; j ++）

{

If （num[ j ] > num [ j + 1 ]）

{

Var temp = num[ j ]

Num[ j ] = num [ j + 1 ]

Num [ j + 1 ] = temp

}

}

} console.log（num）

1. Sort 方法 /正序倒序

var array = [7 , 6 , 3 , 9 , 1 ,2 ]

Array . sort （function（e1 , e2））{ return e1 - e2 } console.log（array）

Sort 后面可以添加一个函数， 函数有两个参数（e1, e2），可以对两个参数进行对比 （ e1 - e2 / e2 - e1 ）以达到正序或倒序的目的

67.forEach方法 / 查找

Var array = [1 , 2 , 3 , 4 , 5 ]

Array . forEach （function （e , i , a））{ console.log( e ) ( i ) ( a ) }

Foreach方法可以找到数组当中的每一个元素，元素的索引，以及所在数组

Each ： 每一个

E ：element 元素

I ： index 元素索引

A ：array 元素所在的数组

1. Every 方法 / 判断符合条件1

Var array = [11 , 33 , 66 , 99]

Var num = array . every ( function ( e ) { return e > 11 }) console.log( num )

Every 每一个 ；方法里面的参数表示数组当中的元素，分别判断每一个元素是否符合条件； 如果全部符合，则返回true，如果任意一个不符合，则返回false

只要有一个为假，剩下的不再判断

1. some 方法 / 判断符合条件2

Var num = array . some (function ( e ) { return e > 50 })console.log( num )

Some 一些 ；分别判断每一个元素是否符合条件；如果符合，元素返回true ， 如果 不符合， 元素返回false ，有所元素如果至少有一个符合条件，那么整体返回true否则整体返回 false

只要有一个为真，剩下的不再判断

1. Filter方法 / 过滤元素

Var array = [33 , 44 , 55 , 66]

Var array1 = array . filter (function ( e ) { return e > 5 })console.log( array1 )

Filter 过滤 ，过滤出所有符合条件的元素，放人新的数组中

1. Reduce 方法 / 获得前数和

Var array = [4 , 5 , 6 , 7 , 8 , 9]

Var num = array . reduce (function ( total , e ) {

Console.log( total )

Console.log( e )

Return total + e

} )

Console.log( num )

Reduce 减少，total 总数 ； 获取前面所有两数之和；第一个参数为前面两数之和；第 二个参数为数组的元素；第一个参数 + 第二个参数的和会作为下一次循环当中的第一个参数

1. 随机数 var random = Math . random ( )
2. 获取一个不大于原来数值的最大整数/ 向下取整 Math . floor ( )
3. 获取一个不小于原来数值的最大整数/ 向上取整 Math . ceil ( )
4. 圆周率 Math . PI ;3.1415926
5. N 的 N 次方 MAth . pow (3 , 3 ) ;27
6. 开方 Math . aqrt ( 27 , 1 / 3) ;5.19615
7. 获取绝对值 Math . abs ( -5 ) ; 5
8. 角度转弧度 Math . sin ( 180 / Math . PI \* 24) -0.79408

角度 / 180 \* PI

1. 四舍五入 Math . round ( 4.9 ) ;5
2. 获取当前最新的时间 time = new Date ( ) ;date 日期 data 数据
3. 获取从1900年到现在的年数 time . getYear( )
4. 获取当前的年份 time . getFullYear( )
5. 获取当前月份索引 time . gerMonth ( ) + 1
6. 获取当前的日期 time . getDate ( )
7. 获取当前的星期 time . getDay ( )
8. 获取当前的小时 time . getHours ( )
9. 获取当前的分钟 time . getMinutes ( )
10. 获取当前的秒 time . getSeconds ( )
11. 匿名函数链式调用

Function test（）{ } test（）

Var f1 = function（）{ }f1（）

匿名函数在调用的时候，必须在方法声明以后调用

1. 以下三种情况默类型为false 不显示

Var a = undefind ; undefind 未定义 变量没有声明

Var b = null ；null 空 变量声明但是没有值

Var c = NaN ; NaN 不是数字 变量声明，有值，但是不是数字

1. 找到当前文档 （document）
2. 找到html标签 （document . documentElement）
3. 找到网页标题 （document . title）
4. 找到head标签 （document . head）
5. 找到body标签 （document . body）
6. 创建标签 var div = document . createElement( ‘div’ )
7. 放入body里 document . body . appendChild( div )
8. 获取标签

Document . getElementById( ) 通过id 获取

Document . getElementsByName( ) [0 ] 通过标签标题获取并找到索引为 0 的这个

Documetn . querySelector( ) 找到所有指定标签当中第一个

Document . querySlectorAll( )[0 ] 找到所有指定标签并获取索引为0 的这个

Document . getElementsByTagName( )[ ] 通过标签名字获取并找到索引为0 的这个

1. 获取标签id名字 （ div . id ）
2. 获取所有属性 （ div . attributes ）
3. 获取类名 （ div . className ）
4. 获取类名列表 （ div . calssList ）
5. 删除指定类名 div . classList . remove(‘指定类名’ )
6. 增加类名 div . calssList . add ( ‘类名’ )
7. 判断是否有指定类名 var result = div . classList . contains( ‘指定类名’ )
8. 查看是否有指定类名，有则删除，无则添加 div . classList . toggle( ‘指定类名’ )
9. 设置属性并赋值 div . setAttribute( ‘fond’ , ‘美食’ )
10. 获取某个属性的值 div . getAttribute( ‘fond’ )
11. 判断是否有指定属性 div . hasAttribute( ‘指定属性’ )
12. 获取data - XX 开头的属性 div . dataset . XX
13. 加载文本 div . innerText 标签形式的字符串会被认为是文本的一部分
14. 加载html文本 div . innerHTML 标签形式的字符串会被当成标签
15. 设置文本内容 div . innerContent 子标签为display：none 还显示子标签内容
16. 设置文本超出省略 textOverflow ： ‘ ellipsis ’
17. 定时器格式 setInterval( function ( ) { } , 1000)逗号后为时间间隔 每隔多久调用一次
18. 定时器每隔 N秒执行一次 共十次

Var index = 0 setInterval ( function ( ) {

Var a = function ( ) { var index = 0

Index ++ index + +

Console . log ( index ) if ( index < 10 )

If ( index = = 10 ) {

{ console . log ( index )

Return }

} } , 2000 )

}

setInterval ( a , 2000 )

1. 延时器格式 function do( ) { } setTimeout( do , 5000)逗号后为间隔多少时长后调用一次
2. 延时器每隔N秒执行一次

Var index = 0

Var a = function ( ) {

Index + +

Console . log ( index )

If ( index == 10 )

{

return

}

setTimeout ( a , 2000 )

}

a ( )

1. 关闭计时器

Var timer = setInterVal( a , 2000)

$ ( ‘ main ‘ ) . on ( ‘ mouseover ’ , function ( ) { clearInterVal ( timer ) })

1. 当鼠标点击 oncolick = ‘ XX()’ XX名字不能叫做click

<div id = ‘first’ onclick = ‘clickfirst()’ > </div>

Function clickfirst( ){ }

<div id=’down’ onclick = ‘clickMe( event )’> </div>

当前被点击的标签/按钮信息 标签添加（event）：function( ) clickMe(event ){

获取被点击按钮的id var select = event . target . id

}

1. 点击事件 设置一个标签 ； 标签 . onclick = function ( event ){ }
2. 鼠标事件
3. 当鼠标按下 标签 . onmousedown = function( ){ }
4. 当鼠标松开 标签 . onmouseup = function ( ){ }
5. 当鼠标移动 标签 . onmousemove = function ( ){ }
6. 当鼠标进入 标签 . onmouseenter = function ( ){ }
7. 当鼠标离开 标签 . onmouseleave = function ( ){ }
8. 当鼠标在标签之上 标签 . onmouseover = function ( ){ } 标签里套标签 冒泡事件传递
9. 当鼠标离开标签 标签 . onmouseout = function ( ){ }
10. 当滑轮滚动 标签 . onwheel = function ( ){ }
11. 鼠标左右和滑轮 window . onmousedown = function ( e ) { console . log ( e . button ) }
12. 跟随鼠标移动（鼠标到哪，标签到哪） funciton ( e ) { e . pageX 水平范围 e . pageY 垂直范围 }
13. 键盘事件
14. 按键按下去输出 window . onkeydown = function (e ){ }
15. 按键松开输出 window . onkeyup = function ( e ){ }
16. 相似输出，部分功能键响应不到 window . onkeypress = function( e ){ }
17. 数组去重

Var array = [ 3 , 4 , 5 , 7 , 3 , 4 , 5 , 6 ]

Var result = [ ]

For ( var index = 0 ; index < array . length - 1 ; index ++ )

{

Var beforeNum = array [ index ]

For ( var j = index + 1 ; j < array . length ; j ++ )

{

Var afterNum = array [ j ]

If ( beforeNum == afterNum )

{

break

}

Else{

If ( j == array . length - 1 )

{

Result . push ( beforeNum )

}

}

}

}

Result . push ( array [ array . length - 1 ] )

Console.log( result )

1. 最大最小值

Var array = [ 22 , 23 , 0 , 55 , 30 ]

Var max = array[0]

Var min = array[0]

For (var index = 1 ; index < array . length ; index ++ )

{

Var num = array [ index ]

Num > max ? max = num : null

Num < min ? min = num : null

}

Console . log ( ‘ 最大值 ：’ + max + ‘ , 最小值 ： ’ + min)

1. 判断是否在做动画

Var isPlaying = false

动画名称 图总张数 第几章播放声音

Function beginAnimation ( animation , imgToal , playAt ){

如果正在做动画

If ( isPlaying == true ){

方法结束

return

}

isPlaying = true

}

}

1. 获取当前屏幕宽高

Var clientWidth = document.documentElement.clientWidth

Var clientHeight = document.documentElement.clientHeight

1. 获取当前屏幕高 jquery方式

Var lunboWidth = $ ( window ) . width ( )

Var lunboHeight = $ ( window ) . height ( )

1. 随着浏览器尺寸的改变而改变

Window . onresize = function( ) {

XXwidth = document.documentElement.clientWidth

XXheight = document.documentElement.clientHeight

1. 随机大小并向上/向下取整

Var size = Math . floor ( Math . random ( ) \* X ) + Y

Math . cell ( Math . random ( ) \* X ) + Y

1. 设置随机坐标点

Var pointX = Math.random() \* (clientWidth - size) 宽的范围 size为标签本身大小

Var pointY = Math.random() \* (clientHeight - size) 高得范围

XX . style . left = pointX + ‘ px ’

XX . style . top = pointY + ‘ px ’

1. 设置移动距离

Var moveX = Math.random() \* 5 左右运动距离

Var moveY = Math.random() \* 3 上下运动距离

1. 设置反弹 在定时器方法中使用

如果 随机坐标点 + 速度 > = 屏幕宽/高 - 本身大小 或者 < = 0 的时候

If ( pointX + moveX >= ( clientWidth - size ) || pointX + moveX <= 0 )

{ moveX = - moveX}

If ( pointY + moveY >= ( clientHeight - size ) || pointY + moveY <= 0 )

{ moveY = - moveY}

1. 设置每次循环让标签移动的距离

PointX += moveX : pointX = pointX + moveX

pointY += moveY : pointY = pointY + moveY

XX . style . left = pointX + ‘ px ’

XX . style . top = pointY + ‘ px ’

1. 事件监听

<Div onclick = “ clickMe( ) ”> </div>

1. function clickMe ( ){ }
2. (获取div ). onclick = function ( ) { }
3. div . addEventListener ( ‘ click ’ , function ( ) { } )

当div 监听道自己被点击时调用方法，后面写监听事件名称，不可乱写;

里面可以监听的事件为鼠标事件 on 开头的事件 输入时需去掉 on

1. 事件传递链
2. 当点击标签时，事件会先响应给window对象，然后一层一层往下传递，这种过程 叫做捕获 / 挖洞
3. 如果事件从下往上一层一层传递，最终传递给window对象，这种过程叫做冒泡

A.中断事件传播/阻止事件响应链 E . stopPropagation ( )

B.立即停止事件传递 E . stopImmediatePropagation ( )

共同点： 都可中断事件响应链

不同点： A，中断传递链之间，最后一个标签会进行冒泡；B，直接中断，不冒泡

1. 阻止默认事件传递 E . preventDefault ( )
2. Input事件 input.onfocus = function( ){ }
3. 获取焦点 input.addEventListener( ' focus ' , function ( ) { } （ focus ）
4. 失去焦点 ( ' blur ' , function ( ) { } （ blur ）
5. 输入内容的时候 ( 'input ' , function ( ) { } （ input ）
6. 输入内容被选择 ( ' select ' , function ( ) { } （ select ）
7. change内容改变且失去焦点 ( ' change' , function ( ) { } （ change ）
8. 颜色设置 #aaaaa ;

red ;

rgb (225 , 225 , 225 )

rgba ( 225 , 225 , 225 , 0 ~ 1 )

Hsl ( 0 ~ 360 , 100% , 50 % ) 色度 ， 饱和度 ， 亮度

1. 颜色线性渐变 background : linear - gradient ( red , blue )

圆形 内层 中层 外层

1. 颜色径向渐变 background : radial - gradient (clrcle， red , green , blue )
2. 动画帧

动画控制器 animationContrller

设置动画帧animationContrller = requestAnimationFrame ( )括号内写动画方法 1s/60次

取消动画帧cancelAnimationFrom( animationContrller )

1. 尺寸位置
   1. 设备宽 标签 . clientWidth 内容宽 + 内间距
   2. 设备高 标签 . clientHeight 内容高 + 内间距
   3. 从所在标签原点到本标签的距离

标签 . clientTop

标签 . clientLeft

* 1. 偏移距离 ： 内容 + 内间距 + 边框

标签 . offsetHight

标签 . offsetWidth

* 1. 距离浏览器上下 / 左右边框距离 外间距穿透

标签 . offsetTop

标签 . offsetLeft 左右外间距不能穿透

1. scroll滚动轴 标签 . onscroll = function ( e ){ }
2. 滚动距离 this . scrollHeight
3. Body滚动距离 document . documentElement . scrollHeight
4. 内容大小 this . scrollTop
5. 改变滚动颜色

滚动距离 / 最大滚动长度 = 滚动比例

标签. style . color :

‘ hsl ( ’ + This . scrollTop / ( this . scrollHeght - 标签高 + 上下8像素‘16’) + ‘, 100% , 50% ) ’

1. 设置自动滚动

Var index = 0

Function move( ){

标签 . scrollBy ( 0 , index ) 值1 水平方向移动距离 值2 垂直方向移动距离

If ( 标签 . scrollTOP <= 0 || 标签 . scro;;Top >= 总滚动距离 )

{ index = - index } 当值为负数时 往相反方向移动

setTimeout ( move , 200 ) }

1. Body的偏移距离 document . documentElement . scrollTop

Body . onscroll = function ( ) {

If ( document . documentElement . scrollTop >= 一个距离 200 )

{

标签 . style .top = 0

}

Else

{ 标签 . style . top = ‘ -75 px ’ }

}

1. 控制台台式技巧
   1. Console . log ( )
   2. Console . dir ( ) 基本同log相似
   3. Console . warn ( ) warn警告 ： 注意 ， 代码可能会遇到未知的错误
   4. Console . error ( ) error错误 ： 使用了没有定义的变量XX
   5. Console . info ( ) info提示 ：
   6. Console . assert ( age >= 18 , ’未成年’ ) 断言 ，后跟两个值

值1 判断条件 只能为真假

值2 当条件为假的时候输出的内容

* 1. Console . time( ‘for’ )

Console . timeEnd ( ‘for’ ) 成对出现，名称一致 获取中间的内容加载时间

* 1. Console . group ( ‘gp’ )

Console . groupEnd ( ‘gp’ ) 团;组 成对出现 表示在同一个表达组内

* 1. Console . table ( 对象名称 ) 获取表格模式 其上添加对象

例： Var people = { name : ‘ ’ , age : ‘ ’ , fond : ‘ ’ }

1. DOM ( Document object Model )文档对象模型 标签组成的树状结构，称之为DOM
2. DOM结构由三部分组成：标签 文本 属性
3. BOM ( borwser object Model )浏览器对象模型

在代码里面进行一些浏览器操作，这些操作都是由一个浏览器对象控制，操作的方法都 是浏览器对象的方法，这个对象就是window

1. 随意一个标签‘script’都是window对象的作用范围
2. 任意全局变量和方法默认前面都会有一个window对象
   1. 例：test（）实际调用的是 window . test（）
3. 获取一个目标的坐标中心点

Var XpointX = 标签 . offsetLeft + 标签宽

Var XpointY = 标签 . offsetTop + 标签高

1. 方法设置动画旋转
   1. 在style设置动画开关 @keyframes zhuan { 0%{ } 100%{ } }
   2. Function rotate ( start , time ) 值1 谁要转 值2 转一周时间
   3. 方法内{ start . style . animation = ‘ zhuan ’ + time + ‘ s linear infinite ’ }
   4. 调用方法 rotate（标签 ，时间 ）
2. Input操作

颜色选择 <input type = “ color ” onchange = “ changeColor ( event ) ” >

文件选择 <input type = “ file” multiple accept = “ image/png , jpg , gif ”>

Multiple 多选 accept 指定类型文件

数据日期 <input = “ date ” onchange = “ changeDate( event ) ”>

1. 标签. oninput = function ( ){ } 将内容转化成数字 前提可转化

输入框里面不管写什么 ， 获取的内容永远是字符串！！

1. 标签 . value = parseInt ( 标签 .vlue ) parseInt 将内容转化成整数
2. 标签 . value = parFloat ( 标签 .vlue ) parFloat 将内容转化成小数
3. 获取随机字母
   1. 数组随机

Var charString = ‘ abcdefghijklmnopqrstuvwxyz ’

Var code = charString[ Math.floor( Math.random() \* 26 ) ]

* 1. ASCII码表随机

var randomNum = 0

Var random = Math.floor( Math.random( ) \* 26 ) + 65

Var code = String.fromCharCode( random )

1. 保留N位小数 ： XX . toFixed（ N ）
2. 将控制台节点列表转化成数组

XX = Array . from( XX )

1. 找到元素在数组当中的索引 [ li , li , li ]

Console.log ( lis . indexOf ( this ) )

1. Location位置 是window对象的一个对象属性 ， 本身负责跳转 ， 历史记录等功能
   1. Location . href = ‘ 链接 ’ ： 在本页面跳转
   2. Window . open ( ‘ 链接 ’ ) ： 新打开一个页面
   3. Location . replace ( ‘ 链接 ’ ) ： 上面两种会在历史记录留下痕迹 ， 都是新打开一个页面 ， 本方式是将当前页面替换成指定页面 ， 不会增加历史记录
   4. Location . reload( ) : 重新加载本页面 刷新
2. Console.log ( navigator . userAgent ) : 获取用户使用的设备
3. 通过jQery 来获取标签 ： jQuery 本身为一个函数 ， 向实现什么样的功能 ， 就在函数后面传什么样的参数 , 通常情况下 用 $ 来代替jQuery

<div id = ” first ”></div>

<div class = “ item ”>1</div>

<div class = “ item ”>2</div>

<div class = “ item ”>3</div>

<main></main>

1. $ ( ‘ #first ’ ) . text ( ‘ nice ’ ) : 找到id为first的标签 ， 并设置文本为 nice
2. $ ( ‘ .item ’ ) . text( ‘ good ’ ) : 找到类名为item的标签 ，并设置文本
3. $ ( ‘ .item ’ ) [0] . innerText = ‘ pretty ’ : 找到指定标签 ， 并且设置文本
4. $ ( ‘ main ’ ) . text ( ‘ amazing ’ ) : 通过标签名字找到指定对象 并设置文本
5. $ ( first ) . text ( ‘ lucky ’ ) : 通过变量来找到指定的变量并且修改其文本 ， 如果找不到指定的变量 ， 就通过id 来找 ， 如果能找到同名的id 就获取拥有这个id 的标签 ， 如果找不到同名 id ， 则报错
6. $ ( ‘ div ’ ) . text ( ‘ 周二 ’ )
7. $ ( ‘ div ’ ) . html ( ‘ <h1>今天是周二</h1> ’ )

区别 ： text 全部都为字符串 html 标签会加载

1. $ ( ‘ div ’ ) . css ( ‘ color ’ , ‘ red ’ ) 添加css样式
2. $ ( ‘ div ’ ) . css ( { ‘ color ’ : ‘ blue ’ } , font - size : ‘ 30px ’// color : ‘ blue ’ , fontSize : ‘ 30px ’ )
3. Jquery属性

$ ( ‘ div ’ ) . attr ( ‘ id ’ , ‘ first ’ ) attr 属性 后面跟两个值 值1 属性名称 值2 属性值

获取div的属性：id值

当attr后面跟两个值 ， 表示设置属性为指定的内容

当attr 后面跟一个值 ， 表示获取指定的属性

1. jquery链式调用

Jquery 的任意方法 ， 最后都会返回一个 jquery对象 $ ( ) . terxt ( ) = $

$ ( ‘ div ’ ) . text ( ‘ 我 ’ ) . html ( ‘ <strong>是</strong> ’ ) . css ( ‘ color ’ , ‘ red ’ ) . css ( ‘ font - size ’ , ‘ 50px ’ ) . css ( { border : ‘ 5px solid red ‘ } ) . attr ( ‘ id ’ , ‘ first ’) . text ( ‘ 谁 ’ )

1. Jquery元素创建与添加

创建标签： $ ( ‘ <div> hello world </div> ’ ) . appendTo ( document . body )

添加到.... : $ ( ‘ <h1>放假</h1> ’ ) . appendTo ( ‘ main : nth - chlid ( 1 ) ’ ) 添加到最后面

被添加.... : $ ( ‘ main : nth - chlid ( 2 ) ’ ) . append ( ‘ <h2> 添加h2 </h2> ’ )

添加到.... : $ ( ‘ <h1></h1> ’ ) . prependTo ( document.body ) 添加到最前面

1. Jquery事件

当点击的时候 ， 调用相关方法

$ ( ‘ div ’ ) . on ( ‘ click ’ , clickFirst)

$ ( ‘ div ’ ) . on ( ‘ click ’ , clickSecond)

开启点击事件

Function clickFirst ( ) { }

Function clickSecond ( ) { }

关闭点击事件

$ ( ‘ div ’ ) . off ( ‘ click ’ , clickFirst )

$ ( ‘ div ’ ) . off ( ‘ click ’ , clickSecond )

鼠标进入 $ ( ‘ div ’ ) . on ( ‘ monseenter ’ , function ( ) { } )

按下键盘 $ ( widow ) . on ( ‘ keydown ’ , function ( ) { } )

1. Jquery动画

$ ( div ) . fadeIn ( 200 ) 淡入/渐隐

$ ( div ) . fadeOut ( 200 ) 淡出/渐现

$ ( div ) . hide ( 200 ) 隐藏

$ ( div ) . show ( 200 ) 显示

$ ( div ) . slideup ( 200 ) 上拉/上滑动

$ ( div ) . slidedown ( 200 ) 下拉/下滑动

<div> hello world </div>

隐藏div ： Function clickHidden ( ) { $ ( ‘ div ’ ) . fadeOut ( 2000 ) } 通过 display：none

$ ( ‘ div ’ ) . hide ( 2000 ) // $ ( ‘ div ’ ) . slideUp ( 2000 )

显示div ： Function clickShow ( ) { $ ( ‘ div ’ ) . faddeIn ( 2000 ) }

$ ( ‘ div ’ ) . show ( 2000 ) // $ ( ‘ div ’ ) . slideDown ( 2000 )

设置动画 ： function clickAnimation ( )

{ $ ( ‘ div ’ ) . animate ( {

left : ‘ 500px ’ ,

top : ‘ 500px ’ ,

fontSize : ‘ 100px ’ ,

} , 1000 , ‘ linear , function ( ) {

console . Log( ‘ 动画完成 ’ ) } ) }

1. 搜索框下拉列表

<input type = “ text ”> 设置input宽

<main></main> 设置main 宽/高 ， 边框

$ ( ‘ main ’ ) . slideUp ( 0 ) 解决刷新显示main问题

获取input 焦点时 main 显示

$ ( ‘ input ’ ) . on ( ‘ focus ’ , function ( ) { $ ( ‘ main ’ ) . slideDown ( 500) } )

失去input 焦点时 main 隐藏

$ ( ‘ input ’ ) . on ( ‘ blur ’ , function ( ) { $ ( ‘ main ’ ) . slideUp ( 500) } )

1. Typeof 与 instanceof 的使用

区别 ： typeof 查看某个对象的类型 返回值有六种

Instance 查看某个对象是否为某个类的实例 返回值有两种

Type类型 ： string , number , boolean , object , function undefined

不管任何类型的变量 他的类型只能为其中的一种

Instanceof : 判断某个对象是否为某个类的实例对象 ， 判断结果只能为真假

Var array2 = [ 3 , 4 , 5 , 6 , 7 ]

Console . log ( array2 instanceof Array )

1. 构造函数

构造函数的字面量创建方式 优点是简单，缺点是不便使用 ，当有多个对象需要创建的时候，非常麻烦。

构造函数要求首字母必须大写，以便和方法以作区分。

Var people = { name : ‘ 张三 ’ ， age : ‘ 17 ’ , say : function ( ) { } } people . say ( )

Function People ( name , age ) {

this . name = name

this . age = age

this . work = function ( ) { this . name + ‘ ’ + this . age } }

Var p = new people ( ‘ name ’ , age )

P . work ( )

1. 原型方法 所有对象共享的方法叫做原型方法 prototype 原型

People . prototype . work = function ( ) { }

1. $ ( ‘ div ’ ) . eq ( index ) . fadeIn ( 500 ) 找到div里面的第index个元素
2. 面向对象 / 过程
   1. 面向过程 典型代表 ： C语言
   2. 面向对象 java , oc , js , prthon . 主流开发

面向过程：侧重于问题的解决方式，以解决问题为重点，解决问题要靠函数（方法 function （）），所以函数是面向过程编程语言的第一要素。

面向对象：侧重于问题的解决对象，由哪个对象来解决问题，所以对象是面向对象编程 语言的第一要素，在面向对象的编程语言中，函数不能独立存在，必须依托 于对象中。

面向过程：吃 （ 张三 ， 饭 ）

面向对象：张三 （ 吃 ， 饭 ）

面向对象的编程语言的基本概念：

类： 汽车 人类 美食 Data（） array（）

类是属性和方法的集合

对象： 宝马 张三 炒饭 time array1

从类中分离出来的具体的个体，对象拥有类的一切属性和方法

面试题：面对对象编程语言的三大特性？

1. 封装 只露出外部需要的功能 ，在对象内部隐藏外部不需要的功能

（理解 ：没有毫无保留的付出）

1. 继承 一个类A，可以继承另外一个类B，从此类A 就拥有了类B的一切属性

（理解 ：站在巨人的肩膀上）

1. 多态 当不同类型的对象调用同一个方法的时候，会产生不同的结果

（理解 ：种瓜得刮 ，种豆得豆）

1. Call , apply 和 bind
2. call

Function fun1 ( a , b ){ consol . log( this ) } this 指的是 window

Window . fun1 ( ‘ name ’ , 17 )

Fun1 . call ( document . body , ‘ 张三 ’ ， ‘ 李四 ‘ ) window 变 body

此代码体现了三大特性中的多态，call ： 改变方法的this指向 ， 后面跟参数 。

值1 ： 想让this 变成谁 值2 ： 方法需要的参数

不同的对象（ window ， document . body ）调用同样的方法 （ fun1 ），产生了不同的结果（ this指向改变 ， 参数a ， b值改变 ）叫做多态

创建一个构造函数 = 类 属性和方法的集合  
function People ( name ){

this . name = name

this . say = function {

Console.log ( this . name )

setInterVal(function ( ) {

Console.log ( this . name + ‘ 说，hello world ’ )

} , call ( this ) , 2000 )

}

This . work = function ( ) {

Console.log( this )

Var that = this

SetInterVal ( function ( ) {

Console . log ( this )

Console . log ( that )

} , 2000 )

}

}

Var p = new People ( ‘ 小明 ’ ， 12 )

Fun1 . call ( p , 2 , 3 )

P . say ( )

方法A 里面套用了方法 B , 方法A 里面的this 和方法B 里面的 this指向是不同的， 通过this只能找到方法B里面的内容 ， 而不能找到方法A 里面的内容 ， 如果想要 找到方法A 里面的内容， 就需要更改this指向， / / 这种方法不是最好的

1. Apply

Function fun2 ( name , age ){ console . log ( this ) }

Fun2 ( ‘ 张三 ’ , 16 )

Fun2 . apply ( document . body , [ ‘ 李四 ’ ， 17 ] )

Function fun3 ( ) {

Console . log ( arguments ) // arguments 所有参数

Console . log ( arguments . length ) // 参数的长度

Console . log ( arguments [ 0 ] ) // 参数索引

}

1. Bind

Function fun4 ( ) { console . log ( this ) }

Var fun5 = fun4 . bind ( document . body )

Fun5 ( )

面试题 ：call 和 apply 的区别

Call后面多个参数 ， 第一个参数为this指向的对象 ， 剩下的为函数的参数

Apply后面两个参数 ，第一个参数为this指向的对象 ， 第二个参数为数组，数组里面为函数的参数

面试题 ：call ， apply和 bind的区别

Call和 apply 在更改完this指向以后 ， 会立马调用

Bind在更改完this指向以后 ，不会自己调用 ， 需要手动调用

1. 类继承

Function Baby ( name , age ) {

this . name = name

this . age = age

this . play = function ( ) {

Console . log ( ‘ 天生会玩 ’ )

}

}  
 var b = new Baby ( ‘ 狗蛋 ’ ， 1 )

B . play ( )

Function student ( name , age , homework ) {

Bady . apply ( this , arguments ) // 类继承 继承的类会拥有继承类的所有属性

This . homework = homework

This . study = function ( ) { }

}

Var s = new student( ‘ 二狗 ’ , 15 , ‘ 编程 ’ )

S . sduty ( ) s . play ( )

1. 对象原型

Function People ( name , age ) {

This . name = name

This . age = age

Thiis . say = function ( ) { } // 每个对象都有自己的 say 方法

}

People . prototype . work= function ( ) { } // 每个对象使用同一个work方法

Var p1 = new People ( ‘ 张三 ’ , 17 ) // protutype 原型 , people类有一个原型方法 work

Var p2 = new People ( ‘ 李四 ’ , 18 ) // 所有对象共享的方法 叫做原型方法

1. Nodeforget 回到上一级方法 ： cd . . ， 下一级 cs + 下级文档名称

所有内容清空 ： cls

Localhost : 8080 获取本机IP

Error: listen EADDRUNUSE : : : 8080 端口被占 更改 8080 为其他

步骤

1. 创建node文件夹
2. 在node文件夹里面创建文件index . js
3. 在node文件夹内创建新的文件夹public 在里面添加 index . html
4. 打开所在文件，点击win + R输入打开命令台
5. 命令台输入cmd 打开终端
6. cd 空格 ，进入指定文件夹 将所在文件拖入页面
7. 输入npm init 回车 初始化node项目
8. 第一个name名字需更改为英文
9. 全部回车直到出现路径
10. 输入 npm install express 安装指定的模块包 express = require ( ‘ express ’ )
11. 输入 npm install parser 安装解析包 bodyParser = require (‘ body - parser ’)
12. Node index . Js 启动服务器
13. Node - v 查看node版本
14. Npm - v 查看npm版本
15. Ipconfig 查看本机ip
16. Control + c 终止命令
17. Cls 清除终端
18. Get 请求

<Form action=” /ligin ”><form> action后面些服务器地址 可以连接到服务器

当form没有设置请求方法的时候 默认方法为get

1. Var express = require ( ‘ express ’ ) 数据请求的基础模块

Require需求: 引入一个express模块包，并且赋值给变量express

1. Var web = express ( ) ： 创建一个express的具体对象，使用这个对象来进行数据请求的操作/返回等 一系列的操作
2. Web . use ( express . static ( public ) ) 当作为服务器时，对方只能访问public这个文件夹里面的内 容，，任何非public文件夹内部的文件 统统访问不到，static静态资源，内容不会变得资源
3. Web . get ( ‘ /login ’ , function ( request , response) { } )

Web . get : 表示当前接口的API ，使用get方法来连接

/Login ：当前接口的名字

Function : 回调函数，当服务器接收到前端氢气的时候，会自动调用

Request ：接收前端数据的对象

Response ：给前端返回数据的对象

Get获取内容 : request . query . zhanghao ; requset . query . mima

1. Web . listen ( ‘ 8080 ’ , function ( ) { consloe . log ( “ 服务器启动......... ” ) } )
2. Post 请求

Post 请求方式 ：url 本身有三部分组成 ： 请求头 请求行 请求体

Get 会将所有信息放在请求头 能够看得到

Post 会将所有信息放在请求体 看不到 ，如果想要使用 ，需要对请求体进行解析

< form action = “ /denglu ” method = “ POST ”><form> method 方式 设置请求方式为post

常用的只有 get post

1. Var express = require ( ‘ express ’ ) 模块包
2. Var bodyParser = require ( ‘ body - parser ’ ) 解析包 将数据解析成key = value这种形式
3. Var web = express ( )
4. Web . use ( express . static ( ‘ public ’ ) ) 使用静态资源文件夹public
5. Web . use ( bodyParser . urlencoded ( { extended : false } ) ) 不支持扩展

（ 支持扩展 ， url 里面就可以包含数组等内容，不支持就只能为字符串 ）

1. Xhr 操作

Var xhr = new XMLHttpRequest ( )自动刷新 不需要点击刷新

Xhr . onreadystatechange = function ( ) { 当准备状态发生改变的时候

（状态 ： 0 ，1 ，2 ，3 ，4）

if ( xhr . readyState == 0 )

{ console . log ( ‘ open方法已经调用了，但是send方法还未调用 ’ ) }

If ( xhr . readyState == 1 )

{ console . log ( ‘ send方法已经调用 ，但是数据请求还没开始 ’ ) }

If ( xhr . readyState == 2 )

{ console . log ( ‘ 服务器已经接收到前端请求 ，但是还没有返回数据 ’ ) }

If ( xhr . readyState == 3 )

{ console . log ( ‘ 服务器开始返回数据 ，并且已经返回来部分数据 ’ ) } 数据大

If ( xhr . readyState == 4 )

{ console . log ( ‘ 服务器已经返回全部数据 ，本次请求流程结束 ’ )

Console . log ( xhr . response ) 获取服务器返回内容 response 响应

Console . log ( xhr . responseText ) 获取返回的text文本

Console . log ( xhr . responseURL ) 获取请求地址

Console . log ( xhr . status ) 获取请求状态

Console . log ( xhr . getAllResponseHeaders ) 获取所有的响应头消息

}

}

Xhr . open ( ‘ get ’ , /showGet?goodsname=哑铃&goodsstart=4&goodscomment=价格公道，快递小哥人很好 ` )

Open方法里面有两个参数，参数1 ：数据请求方式 （get /post），参数2 ：接口地址以及参数，接口和参数之间用 ? 隔开，参数和参数之间用 & 隔开 ( /login?name=xx&age=17 )

Xhr . send ( ) 发送请求到服务器

1. Onclick = “ location . href =’ login . html ‘ ” 鼠标点击跳转到login . html
2. 面试题 ： get 和 post 有什么不同 ？
   1. Get 将请求的参数放在url当中，而post请求将参数放在请求体当中，

后果： get 没有 post 安全，涉及到金钱，账号，隐私数据的时候，一定要用post！！

* 1. Url长度是有限制的 ( 长度是1kb )，get将参数放在url中，所以get的参数大小不能超过1kb，post理论上没有限制，具体要看服务器容量的大小。

后果： 上传图片，电影，文件只能用post请求，因为图片电影文件的大小，都会超过1kb

1M = 1024kb

1kb = 1024b

1b = 8byte

1,2,3,4 = 1b

a,b,c，%，#，@ = 2b

你，好，吗 = 4b

1. 对象序列化 （不能序列化对象的函数属性，如果有，会剔除掉！！）

Var people = {   
 name ： ‘ 张三 ’ ，

Age ： 17 ,

Fond : ‘ 美食 ’

Des ： ‘ 单身使我快乐 ’

Work function ( ) { console . log ( ‘ 早9点，晚9点，一周六天 ， 996 ’ ) }

}

Console . log ( people )

对象转字符串 ： var peopleString = JSON . stringify ( people )

字符串转对象 ： vat peopleParser = JSON . parse ( peopleDtring )

1. 文件操作

Var fs = require ( ‘ fs ’ ) fs : file system 文件操作系统，系统自带fs模块，所以无需下载

Var obj = { time : ‘ ’ , title : ‘ ’ , des : ‘ ’ }

Wirtefile 写入文件 后面跟多个参数，

参数1 ： 写入到哪个文件夹当中，如果文件存在，则直接写入，如果文件不存在，则创建再写入。

参数2 ： 写入的内容

参数3 ： 写完以后的回调函数

Fs . writeFile ( ‘ data . txt ’ , JSON . stringify ( ob ) , function ( err ) {

If ( err ) { console . log ( ‘ 写入失败：’ + err ) }

Else { console . log (‘ 写入成功：’ ) }console . log ( err )

} )

readFile 读取文件 后面跟两个参数

参数1:读取路径

参数2:读取成功或者失败以后的回调函数

Fs . readFile ( ‘ data . txt ’ , function ( err , data ) {

If ( err ) { console . log ( ‘ 读取内容失败 ’ + err ) }

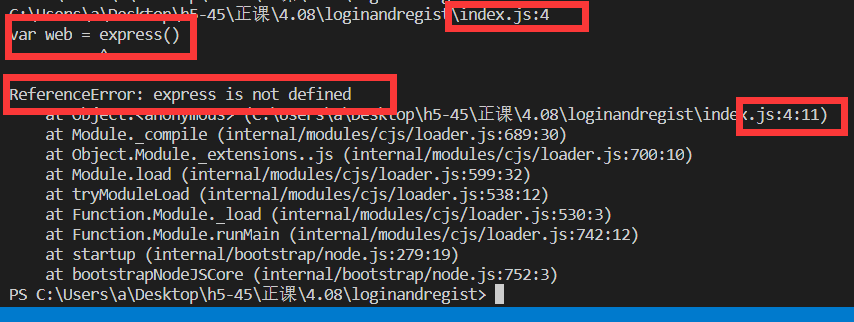
else { console . log ( ‘ 读取内容成功 ’ + data ) }

} )

1. localhost:3000   cannot get

G}$50_JC}TJWC0[%)35H]%H

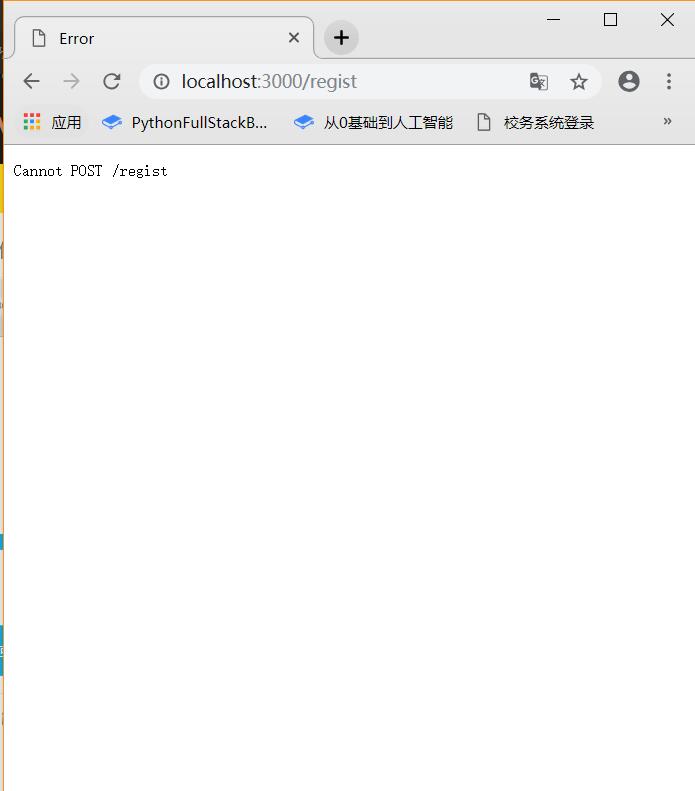
没有找到index.html 看有没有index.html 有没有public 有没有端口 或者是这三者对不对



变量名字写错

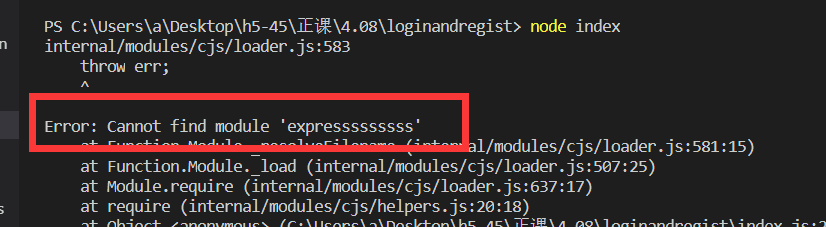
![BH~35~QO]H68IT](B_02V2E](data:image/png;base64,)

端口被占 解决办法 更改端口

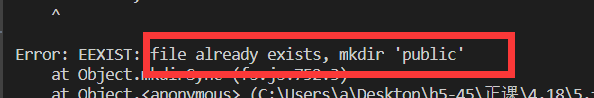


Cannot get + 接口名称

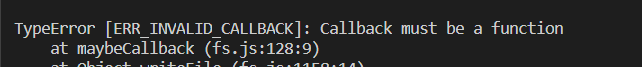
接口在服务器中未找到 原因： 前端接口跟后端接口不一致



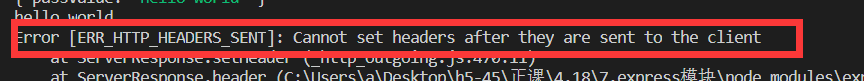
模块未找到 注意模块名称



重复创建一个同样的文件夹

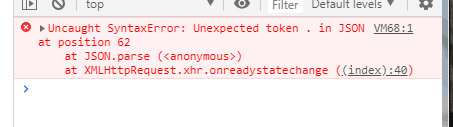


缺少一个回调函数



原因：多次使用send()方法发送内容到前端

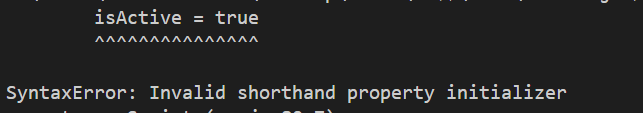
208 .



原因：1.查找写入内容的文件

2.查找JSON。Stringify( 变量 )该方法的变量是否是标准格式（对象数组）

3.JSON . Parse ( 变量 )查找该变量是否为对象或者数组形式的字符串

1. 

对象里面的赋值不是用的“=” 而是‘:’

1. 模块制作

新建text . js

Console . log ( ‘ 这是文件 ’ )

Function calc ( a , b ) { return sum = a + b }

Module . exports . add = calc 将calc功能导出为新的名字 叫做add

1. 路径
   1. Var a = require ( ‘ . / lujing / a . js ’ ) . / 是同级路径
   2. Var b = require ( ‘ . . / b . js ’ ) . . / 是上级路径
   3. Var c = require ( ‘ . . / . . / c . js ’ ) . . / . . / 两层以上路径
2. Fs

mkdirSync 创建文件夹

Exist 判断文件夹是否存在

WriteFileSync 同步写入文件夹

writeFile 异步写入文件夹

readFileSync 同步读取文件夹

ReadFile 异步读取文件夹

\ n 换行符

1. 写入文件 : 参数1 ： 写入的文件名称

参数2 ： 被写入的内容

参数3 ： 回调函数，里面包含错误信息，如果没有错误，则错误信息为空

1. 异步写入文件

Fs . writeFile ( ‘ . / public / this . txt ’ , ‘ 写入内容 ’ ， function ( err ) { if ( err ){ } else { } } )

同步写入文件

1. Var result = Fs . writeFileSync ( ‘ . / public / that . txt ’ , ‘ 写入内容 ’ ) consloe . log ( ‘ relust ’ )
2. 读取文件
3. 异步读取文件

Fs . readFile ( ‘ . / public / this . txt ’ , function ( err , data ) { if ( err ) { } else { } } )

1. 同步读取文件

Var result = fs . readFileSync ( ‘ . / public / that . txt ’ ) consloe . log ( result . toString ( ) )

当读取的内容为 buffer 需要转化成字符串

1. http 模块

Var http = require ( ‘ http ’ ) 引入http模块包 作用是创建一个网络服务

创建服务完成以后，调用的方法，回调函数

Var app = http . createServer ( functio ( req , res ) {

设置响应头，为了让程序识别我们给前端发送的内容的类型

Res . wrietHead ( 200 , ‘ ok ’ , { ‘ Content - type ’ : ‘ text / html ’ } ) 固定格式 防止乱码

Var data = ‘ < meta charset = “ utf - 8 ” > < h1 > 你好< /h1 > ’

Res . write ( data )

Res . end ( )

} )

App . listen ( ‘ 8080 ’ , function ( ) { consloe . log ( ‘ 服务器已启动......... ’ ) } )

1. Var allContent = ‘ ’

Var content = req . query . content

allContent + = content : writeFile会将新的内容覆盖掉，那么在写新的内容之前，现将老的内容获取到，在老的基础上拼接内容，再将增加后的内容，重新写入到文件当中

Fs . writeFile ( ‘ . / public / show . txt ’ , allContent , function ( err ) { } )

1. Xhr . open ( ‘ post ’ , ‘ / login ’ )

Get 请求会将参数拼接到url当中，所以url后面需要追加参数

Post 请求不会将参数拼接到url当中，所以url后面不能追加参数

如果使用post方式，必须指明内容类型

1. 设置请求头 ------- 固定格式

Xhr . setRequestHeader ( ‘ Content - type’ , ‘application / x - www - form - urlencoded’ )

1. 拼接参数到请求体当中

Xhr . send ( ‘ zhanghao= ’ + user + ‘ &mima= ’ + password )

1. form 请求

Form请求本质也是post请求

创建一个form表单( 此表单跟form标签没有任何关系，但是跟form表单相同的一点是 :

都可以作为数据的载体，将所有需要发送额的服务器的数据放进里面 )

Var formData = new FormData ( )

formData . append ( ‘ zhanghao ’ , user )

formData . append ( ‘ mima ’ , password )

append后面跟两个参数 ： 参数1 ：参数名 ； 参数2 ：参数值

好处 ： 1.结构更清晰 2.不用设置请求头

1. Get / post / form 对比

Get post form

请求方式 get post post

参数 url当中 请求体 body 请求体 FormData

Open方法 两个参数： 两个参数 两个参数

1. 请求方式 1. 请求方式 1.请求方式

2.路径+参数 2. 路径 2.路径

Send方法 空 参数 FormData

接收返回数据 onreadystateChange onreadystateChange onreadystateChange

状态4 状态4 状态4

1. Var multer = require ( ‘ multer ’ ) 将body里面的数据解析成form表单
2. Var form = muter ( )

Form . array ( ) 将数据从formData当中取出来，然后在使用bodyParser进行数据解析

Web.post( ‘ / login ’ , form . array ( ) , function ( req , res ) {

Console . log ( req . route ) 路由 ， 页面路径

Console . log ( req . ip ) ip地址/ipv4地址 --------------loacalhost

Console . log ( req . xhr ) 获取xhr对象

Console . log ( req . url ) 获取当前请求地址 http地址

Console . log ( req . body ) 获取请求体

Console . log ( req . method ) 获取请求方式

Console . log ( req . host ) 获取主机地址

} )

1. 创建文件夹

Fs . exists ( ‘ . / data ’ , function ( isExists ) {

If ( ! isExists ) {

Fs . mkdirSync ( ‘ . / data ’ )

}

} )

1. Fs . writeFile ( ) 直接在文件当中写 ，现在写的内容会覆盖之前的内容

Fs . appendFile ( ) 在文件内容的基础上 ，追加的新的内容

1. 数值写入文件夹 需要将值转化成字符串类型

Fs . appendFile ( ‘ . / data / people . txt ’ , JSON . Stringify ( req . body ) , function ( err ) { } )

1. 头像上传

头像上传必须用post请求 ， 如果上传头像 ，enctype属性必写 ，值为multipart / form - data ,

如果该属性不写 ，那么默认值为urlencoded

1. 单张照片

< form action = “ / upload ” method = “ POST ” enctype = “ multipart / form - data ”>< / form >

< input tyep = “ file ” name = “ photo ” accept = “ image / \* ”>< / input >

Accept = “ iamge / \* ” 所有image 类型的图片

1. 设置图片存储位置

Var storage = multer . diskStorage ( { 参数1. 获取文件失败时候的错误信息

1.请求体2.文件内容3.回调函数 参数2. 文件存储路径

Destination : function ( req , file , callback ) { callback ( null , ‘ . / public ’) } ，

1. 设置图片的名字为原来的名字

Filename : function ( req , file , callback ) { callback ( null , file . originalname ) }

} )

1. 上传单张照片到指定的硬盘

Var upload = multer ( { storage : storage } ) ： 参数一必须为storage

Web . post ( ‘ / upload ’ , upload . single ( ‘ phiot ’ ) , function ( req , res ) {

1. 将头像名称写入到本地

Fs . writeFile ( ‘ . / public / use . txt ’ , fullName , function ( err ) { } )

1. 上传后直接显示为头像

res . send ( ‘ < script > window . location . href = “ http : / / localhost : 8080 ” < / script > ’ )

} )

1. Enctype = “ ” 三种值
2. Text / html : http请求设置响应头 ，用来表示文本包含html
3. Application / x - www - form - urlencoded : xhr设置请求头，用来解析body数据
4. Multipart / form - data : form表单属性 ，用来上传头像
5. 多张图片

< form action = “ / upload ” method = “ POST ” enctype = “ multipart / form - data ”>< / form >

< label for = “ select ” > + < / label >

< input id = “ select ” tyep = “ file ” name = “ photo ” multiple accept = “ image / \* ”>< / input >

1. 上传多张图片

Var upload = multer ( { storage : storage } )

Web . post ( ‘ / upload ’ , upload . array ( ‘ photo ’ , 4 ) , funciton ( req , res ) { } )

1. 区别 单张 / 多张

前端 name type accept name type multiple

后端 upload . single ( ‘ ’ ) upload . array ( ‘ ’ , 张数 )

1. 设置图片存储路径以及图片名字

Var storage = multer.dikStorage ( {

Destination : function ( req , file , callback ) {

Callback ( null , ‘ . / public / upload ’ ) } ,

Filename : function ( req , file , callback ) {

Callback ( null , file . originaln ) }

1. 上传配置

Var uploadConfig = multer ( { storage : storage } )

1. Ajax请求

$ . ajax ( {

Url : ‘ postInfo ’ , 设置请求接口

Type : ‘ post ’ , 设置请求类型

Data : { name : ‘ ’ , age : ‘ ’ } , 设置传递的参数

Timeout : 100 设置请求多长时间 算超时 单位毫秒

Success : function ( data ){ 当请求成功的时候 调用的方法

Console . log ( ‘ success ’ ) 当请求失败的时候 调用的方法

Console . log ( data ) 当前请求完成的时候，不管成功或是失败 调用的方法

} ,

Error : function ( err ) { console . log ( ‘ error ’ ) } ,

Comp;ete : function ( ) { console. Log ( ‘请求结束’ ) }

} )

1. $ ( ‘ form ’ ) . serialize ( ) 将form表单里面的所有input标签的value值序列化
2. $ ( ‘ form ’ ) . serialize ( ) 序列化并封装成数组
3. Eq ( n ) 找到索引为 n的这个
4. $ ( ‘ 标签 ’ ) . eq ( 0 ) 找到标签
5. $ ( ‘ 标签 ’ ) . eq ( 0 ) . children ( ) 找到标签的所有子标签
6. $ ( ‘ 标签 ’ ) . eq ( 0 ) .children ( ) .eq ( 0 ) 找到标签的第一个子标签
7. $ ( ‘ 标签 ’ ) . eq ( 0 ) .find ( $ ( ‘ div ’ ) ) 在标签找到所有的子类div
8. Cookie

Cookie 跟 ajax 没有关系 不使用 ajax 也能使用 cookie

Cookie在浏览器关闭的时候 会被自动删除

cookie 作用 ：

1. 判断当前用户书是否 登录 / 注册 / VIP
2. 网站页面之间传值 页面A的值在页面B使用 ， 就可以用cookie传值

Var isLogin = $ . cookie ( ‘ isLogin ’ ) 获取cookie里面名字为isLogin 的这个cookie

$ . cookie ( ‘ caidan ’ , ‘ happy ’ , { expires : 7 } ) 擦混关键cookie ，设置7天保存

Res . cookie ( ‘ isLogin ’ , true ) 发送cookie到前端

Res . send ( ) 会携带cookie到前端

1. Template
2. 制作模板

< script id = ‘ showFilm ’ type = ‘ text / html ’> type 让js 支持使用标签 id为模板名字 固定格式

< h1 > { { film . newPrice | fuzhu } } < / h1 >

< / script >

1. 辅助函数 连接模板 h1 标签里面的 ‘ fuzhu ’

< script >

Template . helper ( ‘ fuzhu ’ , function ( oldValue ) )

Var newValue = Math . floor ( oldValue )

Return newValue

< / script >

1. 使用模板

Var html = template ( ‘ showFilm ’ , res ) 值1 设置使用哪个模板 值2 设置模板要显示的内容

$ ( ‘ body ’ ) . html ( html )

1. Template 循环

{ { each xx as x index} }

{ { if } }

{ { else if } }

{ { / if } }

{ { / each } }

1. 模板引擎

引入模块 Var template = require ( ‘ express - art - template ’ )

Web . engine ( ‘ html ’ , template ) 设置模板引擎对什么类型的文件有效 html为后缀

Web . get ( ‘ / ’ , function ( req , res ){ 当程序一进来的时候 将直接请求

var data = { }

Res . render ( ‘ xuanran . html ’ , data ) render 渲染 创建views 文件夹 固定的 必须这么写

} ) 系统默认hi从这个文件夹当中来找指定的文件

1. Mongoose

Var mongoose = require ( ‘ mongoose ’ ) 引入mongoose模块

1. Mongoose . connect ( ‘ mongodb : // localhost : 27017 / shuju ’ , function ( err ) { if( err ) { console.log( 连接失 败 ) } })

连接到mongo数据库当中 connect 连接 27017 为固定端口号 后面为数据库名称

如果有这个数据库 则直接连接 ； 如果没有 则先创建 在连接

1. Var mongoSchema = new mongoose . Schema ( { 创建图表

Name : String , 该数据为字符串类型

Age ： Number 该数据为数字类型

} )

数据库 存放 数据表 一个数据库里面可以有多个不同类型的数据库

数据库 存放 数据 一个数据表里面可以有多个不同类型的数据

1. Mongo的model模型 Var shuju = mongoose . model ( ‘ shuju ’ , shujuSchema )

Var first = new shuju ( { content : ‘ xx ’ , time : ‘ xx ’ } )

1. 读取数据库 shuju . find ( function ( err , data ) { if ( err ) { } else { } } )
2. 数据库基本知识
3. 计算机当中 数据的存储有三种方式
4. 将数据存储到内存当中

例如:var name = '张三' ; var age = 17

特点:

1. 系统直接读取内存 , 速度最快 。

2. 数据都是临时存储 , 当程序关掉的时候内存被释放 , 数据就消失了

2. 将数据存储到本地

例如:fs.writeFile('./public/test.txt',data,function(err){})

特点:

1. 数据都是永久存储 , 不主动删除的话 , 数据不会消失

2. 操作数据极其麻烦 , 对使用者不友好

3. 将数据存储到数据库当中

例如:mongo , mysql

特点:

1. 数据永久存储 , 操作数据非常便利
2. 数据库按照性质来分，主要分为两种 ：
3. 关系型数据库 mysql sqlit

数据和数据之间有着紧密的联系 , 通过一个数据可以找到其他的数据 (牵一发动全身)

1. 非关系型数据库 mongo redis

数据之间耦合度较低 , 数据本身不会对其他数据造成影响 (一人吃饱全家不饿)

1. 数据库按照大小来分，主要分为四类

1.大型数据库 oracle

2.中型数据库 sqlserver

3.小型数据库 mongo

4.微型数据库 sqlite （手机端 大小3M）

1. 数据库基本操作
2. 增 2. 删 3. 改 4.查
3. 增加 function add ( ) { var student = new Student ( { name : ‘ xx ’ , age : 10 } )

Student . save ( function ( err ) { if ( err ) { } else{ } } )} add ( )

1. 获取 function find ( function ( err , data ) { if ( err ) { 失败 } else { console . log ( data ) } } )

Function find ( ) . exec ( function ( err , data ) { } ) find ( )

1. 通过制定Id来找到对应的数据 ， 每次最多只能找到一条数据

Student . findById ( ‘ 48asdasdfgse5466d ’ , function ( err , data ) { if (err) { } else { data } })

Student . findById (‘ 4567asdf4a7fas4dfff ’ ) . exec ( function ( err , data ) { } )

1. 通过制定数据找到

Stundent . find ( { name : ‘ zhagnsan ’ } , function ( err , daat ) { if ( err ) { } else { data } } )

1. $ lt 小于 $ gt 大于 $ lte 小于等于 $ gte 大于等于
2. 找到小于等于20的数据

Student . find ( { age : { $ lte : 20 } } , funciton ( err , data ) { if ( err ) { } else { data } } )

1. Skip 跳过 表示跳过几条数据

Limit 限制 限制只显示几条数据

1. 找到大于等于10 跳过一条数据并只显示2条数据

Student . find ( { age : { $ gte : 10 } } ) . skip ( 1 ) . limit ( 2 ) .exec ( function ( err , data ) { if ( err ) else { console . log ( data ) } } )

1. Count 数量 符合条件的数据的数量
2. 找到大于等于1 的一条数据

Student . find ( age { $ gte : 1 } ) . count ( 2 ) . exec ( function ( err , data ) { if ( err) { } else { data } } )

1. Update 更新 找到id对应的数据 并且进行更新
2. Function updata ( ) {

Student . findByIdAndUpdate ( ‘ d4ax5c15as4dad ’ , { name : ‘ xx ’, age : 10 } , function ( err ,data ) { if (err) { } else { console . log ( data ) } } ) } update ( )

1. 值1 ：根据条件来找到对应的数据 值2 : 设置为新的数据只能找到第一个符合的更改

Student . update ( { name : ‘ xx ’ } , { name : ‘ aa ’ } , function ( err , data ) { if (err) { } else { data }} )

1. 删除 function del ( {
2. 找到这条Id删除数据

Student . findByIdRemove ( ‘ asd4adad45446 ’ ) , function ( err , data ){ if (err) { } else { data } } } )

1. 通过某项条件删除数据

Student . remove ( { name : ‘ xx ’ } , function ( err , data ) { if (err) { } else { data } } )

1. 直接删除

Student . remove ( function ( err ) { if (err) { } else { find ( ) } } ) del ( )

1. 保存并重定向请求

Mess . save ( function ( ) { res . redirect ( ‘ / ’ ) } )

1. 全局替换 里面写被替换的内容

Req . query . id = req . query . id replace ( / ” / g , ‘ ‘ )

1. token

Var tokenObj = { } 用来存储所有的映射关系（通过...找到...）

tokenObj [ token ] = data 本地保存token

1. Session

Req . session . user 本地保存session 同token

1. Cookie token mession 的区别

相同点 ： 1. 本质上都是cookie

2.可以用来判断登录状态，往前端发送内容

不同点 1.cookie 直接保存在前端浏览器中，当浏览器关闭，cookie消失（设置过期时间除外）

2.token 后台自己手动生成令牌，发送前端一份，自己保存一份，自己保存的一份 会保存在内存中，服务器关闭，令牌消失

3.session 系统自动为后台生成令牌，令牌发送前端一份，自己保存一份，并且将映 射关系保存在数据库当中，到了过期时间才会消失， 最为复杂

1. Promise 许诺 承诺
2. Promise状态有三种
3. 刚创建的时候的状态 （ pending 挂起状态 ）
4. 任务成功以后的状态 （ resolve 解决 ）
5. 任务失败以后的状态 （ reject 拒绝 ）

当pending状态一旦发生改变 不可逆转且不能修改

1. 创建一个图片对象  
    var image = new Image () image . src = ‘ http://dfadf44.jpg ’
2. 当image对象可以正常加载图片的时候调用

Image . onload = function ( ) { resolve ( iamge ) }

1. 当image没有正常加载图片的时候调用

Image . onerror = function ( { reject ( ‘ 未知原因 ’ ) } )

1. Then 然后 当promise对象的pending状态发送改变的时候，会调用对象的then方法

当p状态为resolve的时候，第一个函数会被调用

当p状态为reject的时候，第二个函数会被调用

P . then ( function ( data ) { console . Log( data ) } , function ( data ) { console. Log ( data ) } )

1. Then方法

Then方法必须由promise对象来调用

如果调用then方法的对象不是promise对象，那么该对象会被一个状态为resolve的 promise对象包裹，然后再来调用then方法

1. Var p = Promise . all ( [ p1 , p2 ] ) all 所有的

当数组里面所有对象都为resolve的时候，p的状态才为resolve

当里面的对象有一个为reject的时候，p对象的状态就为reject

1. Var p = Promise . race ( [ p1 , p2 ] ) race 比赛 ，赛跑

p对象最终的状态就为所有对象当中最先改变状态的对象的状态

1. Deferred $ . Deferred 是jquery版的promise
2. Var p = $ . Deferred ( )

If ( ) { p . resolve ( ‘ ok ’ ) else { reject ( ‘ error ’ ) } }

当状态为resolve 调用done方法

当状态为reject 调用fail方法

P . done ( function ( data ) { } )

P . fail ( function ( data ) { } )

1. Function getObj ( time ) {

Return $ . Deferred ( function ( obj ) { 创建一个promise 对象

setTimeout ( function ( ) { 延迟多久以后 将promise对象的状态改成resolve

Obj . resolve ( time + ‘ ms 时间 ’ )

} , time ) } )

}

Var p3 = getObj ( 3000 )

Vat p4 = getObj ( 5000 )

$ . when ( p3 , p4 ) . done ( function ( d1 , d2 ) { alert ( d1 ) dlert ( d2 ) } )

当p3 , p4的状态全部为resolve 调用done

1. 斐波那契函数

别称黄金分割数列 ，兔子数列 指的是 ：1、1、2、3、5、8、13、21、34、.........

Function feibo ( index ) {

If ( index < 2 ) {

Return 1

} else {

Return feibo ( index - 1 ) + feibo ( index - 2 )

} for ( var index = 0 ; index < 100 ; index ++ ){

Console . log ( feibo ( index ) )

}

}

1. 主线程 ： 主线程永远不会关闭 在script标签内的代码都是在主线程执行的
2. 多线程 ： 同时开启多个任务，每个人物都在一个独立的线程当中，这种技术就叫做多线程

特点：有且只有一个主线程，其余的都是分线程，分线程可以自由开启关闭，但是主线程不会关闭

分线程适用 ：当有大量数据需要进行操作，而又不想影响界面响应速度的话，就可以使用分线程

来处理这些数据

多线程和异步的区别 ：

1. 多线程：可以同时执行多个任务，每一个线程里面的任务都是由自己来控制

例如：什么时候做，做什么，怎么做 都要自己控制

1. 异步：可以同时执行多个任务，上个任务没有执行完，写个任务可以开启，

但是每次自己只会做一个任务，剩余的任务由系统帮我们完成

1. 多线程使用

后端 ： 创建斐波那契Js

Onmessage = function ( receive ) { } 当接收前端数据

前端：var work = new Worker ( ‘ feibo . js ’ ) 在菲薄里面开启一个分线程，所有代码在分线程里面执行

Var index = 10

Work . postMessage ( index ) 发送数据到分线程

Work . onmessage = function ( receive ) 接收分线程返回的数据

Work . terminate ( ) 终止运行

1. Async

Asynchronious 异步 synchronious 同步

Js代码在执行的时候 默认是同步的

Var f1 = fs . readFileSync ( ‘ . / file1 . txt ’ )

同步读取文件是没有回调函数的，读取的结果会直接复制个给前面的变量 ( f1 ) ,因为是同步读取，当文件没 有读取完，下一句代码不会执行

Async 将方法变成异步方法，方法配置await

Async function readFile ( ) { var ff1 = await readFile ( ‘ . / file1 .txt ’ ) }

1. sessionStorage 用法 将数据保存到浏览器端 ， 关闭后消失

SessionStorage . setItem ( ‘ name ’ , ‘ zhangsan ’ ) 设置键

SessionStorage . getItem ( ‘ name ’ ) 获取键值

sessionStorage . removeItem ( ‘ name ’ ) 删除指定的某个

SessionStorage . clear ( ) 删除全部

1. Cookie 和 sessionStorage 的区别

相同点 ： 都是保存在浏览器端，都是在浏览器关闭的时候消失

不同点 ： cookie 每个网站只能在浏览器端保存最多20条cookie，每条cookie的大小最大不能超过4kb

sessionStorage 每个网站在浏览器端保存的sessionStorage数量没有限制，每条数据大小最大不 能超过5M

1. LocalStorage 用法 将数据存储到本地，，而且是永久存储

LocalStorage . setItem ( ‘ name ’ , ‘ zhangsan ’ ) 设置键

LocalStorage . getItem ( ‘ name ’ ) 获取键值

sessionStorage . removeItem ( ‘ name ’ ) 删除指定的某个

SessionStorage . clear ( ) 删除全部

1. sessionStorage 和 localStorage 的异同

相同点 ： 写法一样，存储个数没有限制，每条数据最大都是5M

不同点 ： sessionStorage 是会话存储，网页关闭，存储的数据消失（打开会话=打开网页）

localStorage 是本地存储，数据会永久保存 （存储在本地文件）

1. cookie sessionStorage localStorage的异同点

相同点：

1.都是保存在浏览器端

2.本质上都是cookie

3.都能用来判断是否是登陆状态

不同点：

1. cookie 每个网站只能在浏览器端保存最多20条cookie，每条cookie的大小最大不能超过4kb
2. sessionStorage 每个网站在浏览器端保存的sessionStorage 的条数没有限制 ，每条数据大小不 能超过5M, 是会话存储，网页关闭，存储的数据消失
3. localStorage 每个网站在浏览器端保存的localStorage 的条数没有限制 ，每条数据大小不能超 过5M，是本地存储，数据会永久保存
4. Koa模块写法

Var Koa = require ( ‘ koa ’ ) 添加koa

Var web = new Koa ( ) 设置koa

Var static = require ( ‘ koa - static ’ ) 导入静态资源模块

Web , use ( static ( ‘ public ’ ) )

Var Router = require ( ‘ koa - router ’ ) 导入路由模块

Var router = new Router ( ) 创建路由实例化对象

Web.use ( router . routes ( ) ) 使用路由

router . get ( ‘ / home ’ , async function ( ctx , next ) { ctx . body = ‘ hello world ’ } )

koa路由的请求函数里面 没有req , res 对象 取代的是ctx和next对象

Ctx ：上下文 ，获取和发送数据到前端

Next ：床底控制到一个中间件

Ctx. Body : 发送到前端的内容，要放在ctx的body里面

Ctx . request . query : 获取前端内容

1. Git代码管理仓库

Git创建项目

* 1. 创建git文件夹
  2. 在vs中打开
  3. 打开git终端 ：右键桌面文件夹 - git bash here
  4. Git init : 初始化git项目
  5. Git status : 查看当前git状态
  6. Vs中git内新建红楼梦.txt
  7. git add 红楼梦.txt 将指定文件提交到暂存区
  8. git status 查看状态
  9. 增加名称 ：git config --global user . name “ li ”
  10. 增加网址 : git config --global user . email “ 1846336492@qq.com ”
  11. 完成 git commit -m 红楼梦书名完成

Git命令

1. git init : 初始化git项目
2. git status : 查看当前git状态
3. git add 文件名 : 将指定文件提交到暂存区
4. git add . : 将所有被修改的文件提交到暂存区
5. git commit -m : 将暂存区内容提交到持久区
6. git log : 输出日志
7. git log --oneline : 输出简略日志
8. git reset --hard ID : 显示指定ID以及ID以前的内容
9. git reflog : 查看所有git操作
10. git branch gulong : 创建分支gulong
11. git checkout gulong : 切换到gulong分支
12. git merge gulong 合并主分支
13. git clone + 下载链接
14. cd git\_ljp 添加项目
15. git config --system --unset credential > helper 输入账号密码
16. git pull 更新本地仓库
17. git diff 显示具体的变化
18. git push 提交代码

git上传

1. 进入码云 + 新建仓库
2. 设置名称 + 首字母
3. 设置公开 + 点击创建
4. 克隆下载 - 复制
5. 新建文件夹 + 右键git bash here
6. Git clone 下载
7. 拖入vscode
8. Git cd 添加文件夹路径
9. Vs新建作业内容
10. Git add .
11. Git commit -m 提交
12. Git push 上传
13. socket协议

之前的请求用的协议 http HTTPS

这种协议进行的数据请求，叫做短连接，特点是：服务器不会知道发数据到客户端，需要客户端主 动发数据给服务器，服务器才能返回数据

Socket 套接字，是长连接技术，服务器会主动发数据给客户端，而且数据发送完毕以后，连接不 会断开，服务器和客户端会一直跟对方练习，看是否断开连接

！三次握手

第一次握手： 客户端尝试连接服务器，向服务器发送同步(sync)包，

客户端进入等待服务器确认

第二次握手： 服务器接收客户端同步包并确认，同时向客户端发送一个包，

此时服务器进入接收状态

第三次握手： 客户端收到服务器的包，向服务器发送服务包，此包发送完毕，

客户端和服务器进入准备状态，完成三次握手

当前学过的协议

http file mailto ws(websocket) mongodb mysql(未学)

1. Socket步骤
   1. 前端：连接到服务进行三次握手   
      var soc soc = new WebSocket ( ‘ ws : / / localhost : 8080 ’ )
   2. 后端： 如果connection方法被调用的时候，说明已经连接到socket服务器

Var http = require ( ‘ http ’ )

Var httpServer = http . createServer ( )

Var WebSocket = require ( ‘ wsx’ ) . Server

Var wss = new WebSocket ( { servver : httpServer } )

当接收到request请求的时候 使用wss对象来替换wevb对象

httpServer . on ( ‘ request ’ , web )

httpServer . listen ( ‘ 8080 ’ , function ( ) { console . log ( 服务器启动...... ) } )

* 1. 前端：将连接结果发送给后端

soc . onopen = function () { console . log ( ‘ 已经连接到socket服务器 ’

Var obj = { type : ‘ neckname ’ , data : 标签value值 }

Var objString = JSON . stringify ( obj )

) }

* 1. 前端：将数据发送给服务器

res . send ( objString )

* 1. 后端：当服务器接收到客户端发送过来的事件做出方法后发给服务器

Ws . on ( ‘ message ’ , function ( data ) {

....send ( )

} )

* 1. 前端：当服务器接收到服务器返回的内容

Soc . onmessage = function ( data ) { }

* 1. 前端：关闭socket连接

Function outLIne ( ) { soc . close ( ) }

* 1. 后端：当发现前端用户断开连接时

Ws . on ( ‘ close ’ , funcetion ( data ) { } )

1. 邮箱验证
   1. 用xhr请求发送数数据

var xhr = new XMLHttpRequest ( )

Xhr . open ( ‘ post ’ , ‘ /send ’ )

Xhr . setRequestHeader ( ‘ Content - tyoe ’ , ‘ application / x - www - form - urlencoded ’ )

Xhr . send ( ‘ email= ’ + input . value + ‘ &content= ’ + input . value )

* 1. Express 模块 bodyParser模块 emial模块

Var email = require ( ‘ emailjs ’ )

* 1. 设置服务器

Var server = email . server . connect ( {

Host : ‘ stmp . qq . com ‘ , 设置用什么类型的邮箱进行发送

User：’ xxxx@qq . com ’ , 邮箱为发送邮件的邮箱 必须开启SMTP

Passwoprd：’ xxxxxxxxxx ’ , 密码为邮箱的授权码

Ssl : true 开启ssl 验证，确保账户安全

} )

* 1. 接收前端请求

Web . post ( ‘ /send ’ , function ( req , res ) {

* + 1. Var email = req . body . email 收件人邮箱 ( 不必开启SMTP )
    2. Var content = req . body . content 邮件内容
    3. Var data = {

From : ‘ xxxxxxx @qq . com’ 从哪个邮箱发送 (开启SMTP)

To : ‘ email ’ 发送给哪个邮箱

Subject : ‘ xxxxxxx ’ 邮件标题

Text ： content 邮件内容

}

Server . send ( data )

} )