

Web APIs 第五天

Bom操作







- 1. 依托 BOM 对象实现对历史、地址、浏览器信息的操作或获取
- 2. 具备利用本地存储实现学生信息表案例的能力





- ◆ Window对象
- ◆ 本地存储
- ◆ 综合案例





Window对象

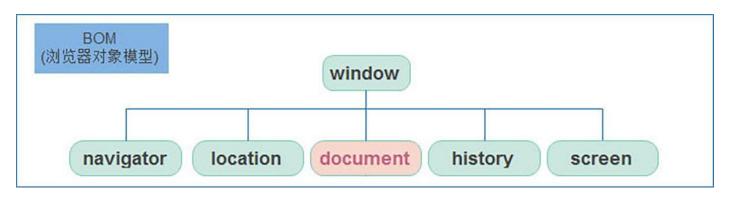
- BOM(浏览器对象模型)
- 定时器-延时函数
- JS执行机制
- · location对象
- navigator对象
- histroy对象

• 目标: 学习 window 对象的常见属性, 知道各个 BOM 对象的功能含义



1.1 **BOM**

● BOM(Browser Object Model)是浏览器对象模型



- window对象是一个全局对象,也可以说是JavaScript中的顶级对象
- 像document、alert()、console.log()这些都是window的属性,基本BOM的属性和方法都是window的。
- 所有通过var定义在全局作用域中的变量、函数都会变成window对象的属性和方法
- window对象下的属性和方法调用的时候可以省略window





Window对象

- BOM(浏览器对象模型)
- 定时器-延时函数
- JS执行机制
- location对象
- navigator对象
- histroy对象

• 目标: 学习 window 对象的常见属性, 知道各个 BOM 对象的功能含义



1.2 定时器-延时函数

- JavaScript 内置的一个用来让代码延迟执行的函数,叫 setTimeout
- 语法:

setTimeout(回调函数,等待的毫秒数)

- setTimeout 仅仅只执行一次,所以可以理解为就是把一段代码延迟执行, 平时省略window
- 清除延时函数:

```
let timer = setTimeout(回调函数, 等待的毫秒数)
clearTimeout(timer)
```

- 注意点
- ▶ 延时器需要等待,所以后面的代码先执行
- > 每一次调用延时器都会产生一个新的延时器



1.2 定时器-延时函数

● 两种定时器对比: 执行的次数

▶ 延时函数: 执行一次

▶ 间歇函数:每隔一段时间就执行一次,除非手动清除

> 延时函数递归实现定时器



1 案例

5秒钟之后消失的广告

需求:5秒钟之后,广告自动消失

分析:

①:设置延时函数

②: 隐藏元素





Window对象

- BOM(浏览器对象模型)
- 定时器-延时函数
- JS执行机制
- location对象
- navigator对象
- histroy对象

• 目标: 学习 window 对象的常见属性, 知道各个 BOM 对象的功能含义



经典面试题

```
console.log(1111)
setTimeout(function () {
    console.log(2222)
}, 1000)
console.log(3333)
// 问, 输出的结果是什么?
```

```
console.log(1111)
setTimeout(function () {
    console.log(2222)
}, 0)
console.log(3333)
// 闷,输出的结果是什么?
```



JavaScript 语言的一大特点就是单线程,也就是说,同一个时间只能做一件事。

这是因为 Javascript 这门脚本语言诞生的使命所致——JavaScript 是为处理页面中用户的交互,以及操作 DOM 而诞生的。比如我们对某个 DOM 元素进行添加和删除操作,不能同时进行。 应该先进行添加,之后再删除。

单线程就意味着,所有任务需要排队,前一个任务结束,才会执行后一个任务。这样所导致的问题是: 如果 JS 执行的时间过长,这样就会造成页面的渲染不连贯,导致页面渲染加载阻塞的感觉。



● 为了解决这个问题,利用多核 CPU 的计算能力,HTML5 提出 Web Worker 标准,允许 JavaScript 脚本创建多个线程。于是,JS 中出现了同步和异步。

同步

前一个任务结束后再执行后一个任务,程序的执行顺序与任务的排列顺序是一致的、同步的。比如做饭的同步做法:我们要烧水煮饭,等水开了(10分钟之后),再去切菜,炒菜。

异步

你在做一件事情时,因为这件事情会花费很长时间,在做这件事的同时,你还可以去处理其他事情。比如做饭的异步做法,我们在烧水的同时,利用这10分钟,去切菜,炒菜。

他们的本质区别: 这条流水线上各个流程的执行顺序不同。



同步任务

同步任务都在主线程上执行,形成一个执行栈。

异步任务

IS 的异步是通过回调函数实现的。

- 一般而言, 异步任务有以下三种类型:
- 1、普通事件,如 click、resize等
- 2、资源加载,如load、error等
- 3、定时器,包括 setInterval、setTimeout 等

异步任务相关添加到任务队列中(任务队列也称为消息队列)。

执行栈

console.log(1

setTimeout(fn,0

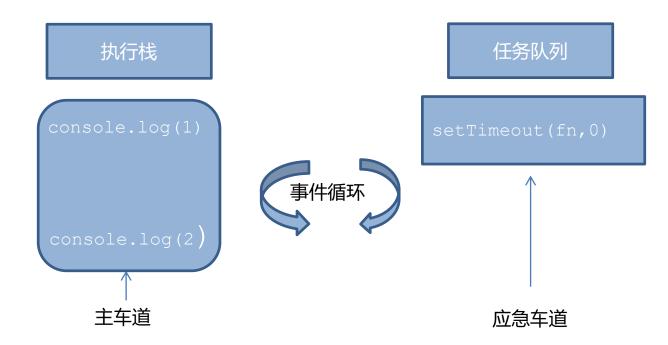
console.log(2)

任务队列

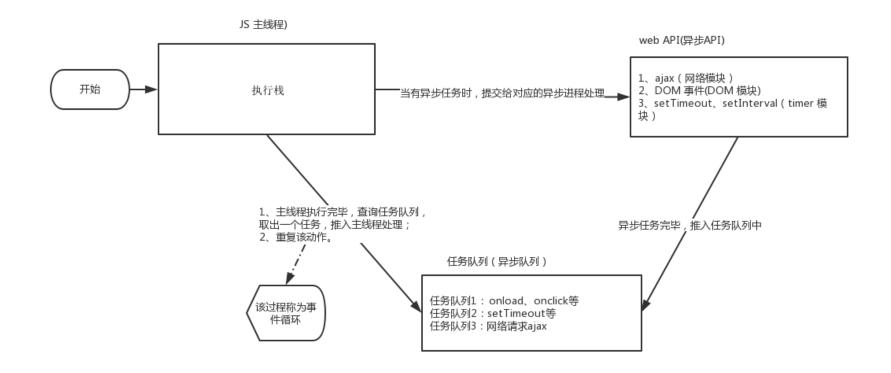
īn



- 1. 先执行执行栈中的同步任务。
- 2. 异步任务放入任务队列中。
- 3. 一旦执行栈中的所有同步任务执行完毕,系统就会按次序读取<mark>任务队列</mark>中的异步任务,于是被读取的异步任务结束等待状态,进入执行栈,开始执行。







由于主线程不断的重复获得任务、执行任务、再获取任务、再执行,所以这种机制被称为事件循环(event loop)。





```
console.log(1)

document.addEventListener('click', function () {
   console.log(4)
})

console.log(2)

setTimeout(function () {
   console.log(3)
}, 3000)
```





Window对象

- BOM(浏览器对象模型)
- 定时器-延时函数
- JS执行机制
- · location对象
- navigator对象
- histroy对象

• 目标: 学习 window 对象的常见属性, 知道各个 BOM 对象的功能含义



- location 的数据类型是对象,它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法:
- ▶ href 属性获取完整的 URL 地址,对其赋值时用于地址的跳转
- > search 属性获取地址中携带的参数,符号?后面部分
- ▶ hash 属性获取地址中的啥希值,符号 # 后面部分
- > reload 方法用来刷新当前页面,传入参数 true 时表示强制刷新



一、Window对象

- location 的数据类型是对象,它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法:
- ▶ href 属性获取完整的 URL 地址,对其赋值时用于地址的跳转

```
// 可以得到当前文件URL地址
console.log(location.href)
// 可以通过js方式跳转到目标地址
location.href = 'http://www.itcast.cn'
```



富案例

5秒钟之后跳转的页面

需求:用户点击可以跳转,如果不点击,则5秒之后自动跳转,要求里面有秒数倒计时

分析:

①:目标元素是链接

②: 利用定时器设置数字倒计时

③:时间到了,自动跳转到新的页面

支付成功 /秒之后跳转回原网页



- location 的数据类型是对象,它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法:
- > search 属性获取地址中携带的参数,符号?后面部分

console.log(location.search)



- location 的数据类型是对象,它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法:
- ▶ hash 属性获取地址中的哈希值,符号 # 后面部分

console.log(location.hash)

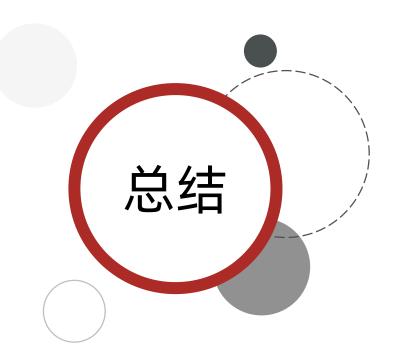
▶ 后期vue路由的铺垫,经常用于不刷新页面,显示不同页面,比如 网易云音乐



- location 的数据类型是对象,它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法:
- > reload 方法用来刷新当前页面,传入参数 true 时表示强制刷新

```
<button>点击刷新</button>
<script>
    let btn = document.querySelector('button')
    btn.addEventListener('click', function () {
        location.reload(true)
        // 强制刷新 类似 ctrl + f5
     })
</script>
</script>
```





- ▶ location.href 属性获取完整的 URL 地址,对其赋值时用于地址的跳转
- > search 属性获取地址中携带的参数,符号? 后面部分
- ▶ hash 属性获取地址中的啥希值,符号 # 后面部分
- > reload 方法用来刷新当前页面,传入参数 true 时表示强制刷新





Window对象

- BOM(浏览器对象模型)
- 定时器-延时函数
- JS执行机制
- location对象
- navigator对象
- histroy对象

• 目标: 学习 window 对象的常见属性, 知道各个 BOM 对象的功能含义



1.5 navigator对象

- navigator的数据类型是对象,该对象下记录了浏览器自身的相关信息
- 常用属性和方法:
- > 通过 userAgent 检测浏览器的版本及平台

```
// 检测 userAgent (浏览器信息)
!(function () {
    const userAgent = navigator.userAgent
    // 验证是否为Android或iPhone
    const android = userAgent.match(/(Android);?[\forall \sets\forall /] +([\forall \delta .] +)?/)
    const iphone = userAgent.match(/(iPhone\forall \sets OS)\forall \sets([\forall \delta .] +)/)

// 如果是Android或iPhone,则跳转至移动站点
    if (android || iphone) {
        location.href = 'http://m.itcast.cn'
        }
})()
```





Window对象

- BOM(浏览器对象模型)
- 定时器-延时函数
- JS执行机制
- · location对象
- navigator对象
- histroy对象

• 目标: 学习 window 对象的常见属性, 知道各个 BOM 对象的功能含义



1.6 histroy对象

- history 的数据类型是对象,主要管理历史记录,该对象与浏览器地址栏的操作相对应,如前进、后退、历史记录等
- 常用属性和方法:

history对象方法	作用
back()	可以后退功能
forward()	前进功能
go(参数)	前进后退功能 参数如果是 1 前进1个页面 如果是-1 后退1个页面

history 对象一般在实际开发中比较少用,但是会在一些 OA 办公系统中见到。







- ◆ Window对象
- ◆ 本地存储
- ◆ 综合案例





本地存储

- 本地存储介绍
- 本地存储分类
- 存储复杂数据类型

• 目标: 学习 window 对象的常见属性,知道各个 BOM 对象的功能含义



2.1 本地存储介绍

- 以前我们页面写的数据一刷新页面就没有了,是不是?
- 随着互联网的快速发展,基于网页的应用越来越普遍,同时也变的越来越复杂,为了满足各种各样的需求,会经常性在本地存储大量的数据,HTML5规范提出了相关解决方案。
- 1、数据存储在用户浏览器中
- 2、设置、读取方便、甚至页面刷新不丢失数据
- 3、容量较大, sessionStorage和localStorage约 5M 左右
- 常见的使用场景:
- https://todomvc.com/examples/vanilla-es6/ 页面刷新数据不丢失





本地存储

- 本地存储介绍
- 本地存储分类
- 存储复杂数据类型



2.2 本地存储分类- localStorage

目标: 能够使用localStorage 把数据存储的浏览器中

● **作用:** 可以将数据永久存储在本地(用户的电脑), 除非手动删除, 否则关闭页面也会存在

● 特性:

▶ 可以多窗口(页面)共享(同一浏览器可以共享)

> 以键值对的形式存储使用



2.2 本地存储分类- localStorage

● 语法:

存储数据:

localStorage.setItem(key, value)

获取数据:

localStorage.getItem(key)

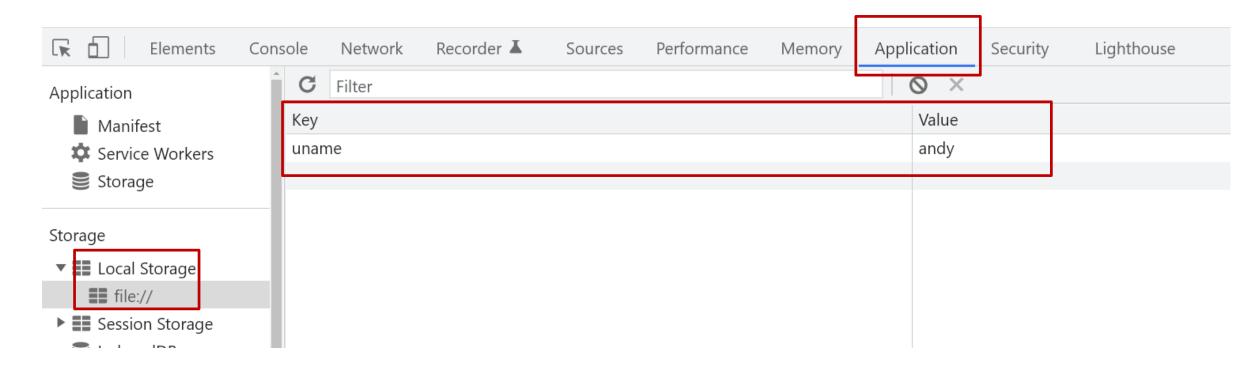
删除数据:

localStorage.removeItem(key)



2.2 本地存储分类- localStorage

● 浏览器查看本地数据:





2.2 本地存储分类- sessionStorage

- 特性:
- ▶ 生命周期为关闭浏览器窗口
- 在同一个窗口(页面)下数据可以共享
- > 以键值对的形式存储使用
- 用法跟localStorage 基本相同





- 1.localStorage 作用是什么?
 - ▶ 可以将数据永久存储在本地(用户的电脑),除非手动删除,否则关闭页面也会存在
- 2. localStorage 存储,获取,删除的语法是什么?
 - 存储: localStorage.setItem(key, value)
 - 获取: localStorage.getItem(key)
 - ▶ 删除: localStorage.removeItem(key)





本地存储

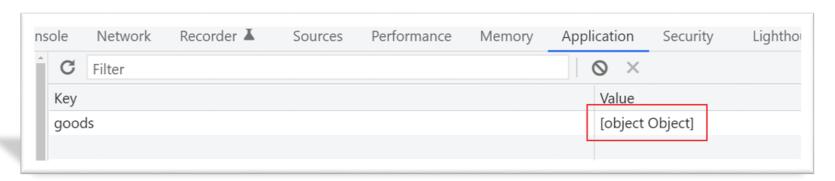
- 本地存储介绍
- 本地存储分类
- 存储复杂数据类型



目标: 能够存储复杂数据类型以及取出数据

● 本地只能存储字符串,无法存储复杂数据类型.

```
const goods = {
  name: '小米10',
  price: 1999
}
localStorage.setItem('goods', goods)
```





● 解决:需要将复杂数据类型转换成JSON字符串,在存储到本地

● 语法: JSON.stringify(复杂数据类型)

```
localStorage.setItem('goods', JSON.stringify(goods))

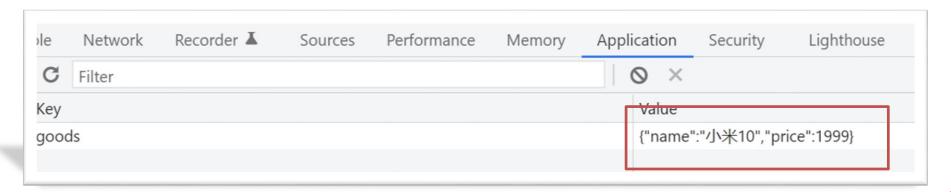
localStorage.setItem('goods', JSON.stringify(goods))

localStorage.setItem('goods', JSON.stringify(goods))

localStorage.setItem('goods', JSON.stringify(goods))

localStorage.setItem('goods', JSON.stringify(goods))
```

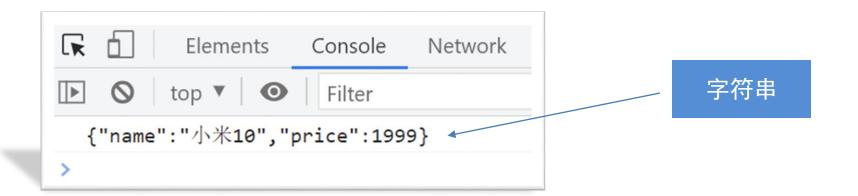
> 将复杂数据转换成JSON字符串 存储 本地存储中





● **问题:** 因为本地存储里面取出来的是字符串,不是对象,无法直接使用

```
const obj = localStorage.getItem('goods')
console.log(obj)
```





● **解决:** 把取出来的字符串转换为对象

● 语法: JSON.parse(JSON字符串)

```
const obj = JSON.parse(localStorage.getItem('goods'))
console.log(obj)
```

➢ 将JSON字符串转换成对象 取出 时候使用





- ◆ Window对象
- ◆ 本地存储
- ◆ 综合案例





学生就业信息表

需求: 录入学生信息,页面刷新数据不丢失

新增学生信息

姓名:	年龄:	性别: 男 > 薪资:	就业城市: 北京 🗸	录入
Company of the last	0.5653800			

就业榜

学号	姓名	年龄	性别	薪资	就业城市	操作
1001	欧阳霸天	19	男	20000	上海	删除
1002	令狐霸天	29	男	30000	北京	删除
1003	诸葛霸天	39	男	2000	北京	删除



国 案例

学生就业信息表

需求: 录入学生信息, 页面刷新数据不丢失

模块分析:

①:新增模块,输入学生信息,数据会存储到本地存储中

②: 渲染模块,数据会渲染到页面中

③:删除模块,点击删除按钮,会删除对应的数据



1 案例

学生就业信息表

需求: 录入学生信息,页面刷新数据不丢失

思路分析:

①:因为页面刷新不丢失数据,所以可能存在已有数据,所以第一步,我们先找本地存储里面查找是否有数据,如果有数据先进行渲染页面,如果没有数据,我们放一个空数组,用来存放数据

②: 渲染模块,数据会渲染到页面中

③:新增模块,输入学生信息,数据会存储到本地存储中,然后渲染页面

④:删除模块,点击删除按钮,会删除对应的数据,然后渲染页面



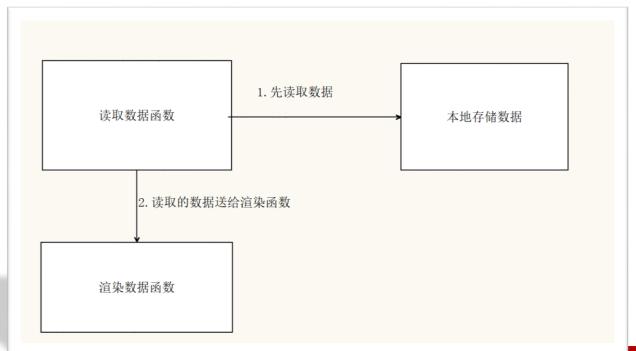


步骤:

需求①:读取本地存储数据

如果本地存储有数据,则返回 JSON.parse() 之后的对象

如果本地存储没有数据,则声明一个空的数组







步骤:

需求②: 渲染模块

- (1) 遍历数组,td里面填写对应td数据,并追加给tbody
- (2) 尽量减少dom操作,所以此处我们不在使用创建节点,追加节点方式(后面vue的做法)



数组中map方法 迭代数组

● **作用:** map 迭代数组

● 语法:

```
const arr = ['pink', 'red', 'blue']
arr.map(function (item, index) {
  console.log(item) // item 得到 数组元素 'pink' 'red' 'blue'
  console.log(index) // index 得到 索引号 0 1 2
})
```



数组中map方法 迭代数组

● 使用场景:

map 可以处理数据,并且**返回新的数组**

```
map([∰, ♠, ♠, ♣], cook)
=> [♠, ♠, ♠, ▮]
```

```
const arr = ['pink', 'red', 'blue']
const newArr = arr.map(function (item, index) {
  console.log(item) // item 得到 数组元素 'pink' 'red' 'blue'
  console.log(index) // index 得到 索引号 0 1 2
  return item + '老师'
})
console.log(newArr) // ['pink老师', 'red老师', 'blue老师']
```



数组中join方法

● 作用:

join() 方法用于把数组中的所有元素转换一个字符串

● 语法:

```
const arr = ['pink老师', 'red老师', 'blue老师']
console.log(arr.join('')) // pink老师red老师blue老师
```

参数:

数组元素是通过参数里面指定的分隔符进行分隔的





步骤:

需求②: 渲染模块

- (1) 遍历数组, td里面填写对应td数据, 并追加给 tbody
- (2) 尽量减少dom操作,所以此处我们不在使用创建节点,追加节点方式(后面vue的做法)
- (3) 我们使用map方法遍历数组,直接返回整个tr,里面包含所有修改后的 tr 标签,里面更换数据
- (4) 但是map方法返回的是一个修改后的数组,怎么办?

所以我们通过join方法转换为字符串

(5) 把返回的结果, 通过 innerHTML 赋值给 tbody

```
        >1001
```





步骤:

需求③: 录入模块

- (1) 事件是提交事件,同样阻止默认提交事件
- (2) 非空判断。
 - 获取所有需要填写的表单, 他们共同特点都有 name属性
 - 遍历这些表单,如果有一个值为空,则return 返回提示输入为空中断程序





步骤:

需求③: 录入模块

- (3) 创建新的对象, 里面存储 表单获取过来的数据, 格式如右图
 - 创建一个空的对象
 - 给对象追加一个 stuld
 - 利用刚才非空判断的循环, 采取对象追加 属性 = 值的方式, 顺便 把表单值 赋值给相应的对象

```
uname: uname.value,
age: age.value,
gender: gender.value,
salary: salary.value,
city: city.value
```





步骤:

需求③: 录入模块

- (4) 追加给数组
- (5) 放入本地存储里面, 记得一定要把数组 利用 JSON.stringify()存储为字符串
- (6) 渲染页面
- (7) 重置表单



ョ 步骤

学生信息表案例

核心思路:

需求④: 点击删除模块

- (1) 采用事件委托形式,给 tbody 注册点击事件
- (2) 点击链接,要删除的是对应数组里面的这个数据,而不是删除dom节点,如何找到这个数据?
- (3) 前面渲染数据的时候,动态给a链接添加 自定义属性 data-id= "0",这样点击当前对象就知道索引号了
- (4) 根据索引号,利用 splice 删除这条数据
- (5) 写入本地存储, 记得一定要把数组 利用 JSON.stringify()存储为字符串
- (6) 重新渲染





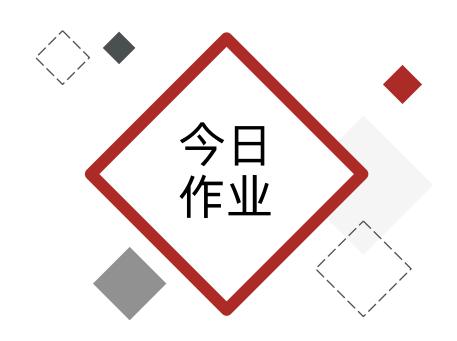
学生信息表案例

核心思路:

需求⑤: 关于stuld 处理的问题

- (1) 最好的做法: 新增加序号应该是最后一条数据的序号 + 1 数组[数组的长度-1].stuld + 1
- (2) 但是要判断, 如果没有数据则是直接赋值为1, 否则就采用上面的做法





- 1. 整理今天笔记
- 2. 必须写出学生信息表案例
- 3. 作业就是 梳理学生信息表 本地存储版本

今天多一份拼搏,明日多一份欢笑



传智教育旗下高端IT教育品牌