JSCORE01

小新老师

负责第三个阶段的课程

个人网站： www.xin88.top

三阶段的内容

核心 - DOM操作：利用 JS 操作网页上的元素

- JS高级：深入学习更多的 JS 编程技巧 -- 4天
- DOM操作： 3天
- jQuery -- DOM的框架， 类似 Bootstrap 和 css 的关系 -- 2天
- 项目实战 - 美食广场 - 8天
- HTML5高级 - 3天
- 工具链 - 1天

JSCORE - JS高级

学习技巧：本阶段的理论知识偏多，重在理解！

- 多数场景， 只要先看懂 明白原理，然后再自己尝试!!!! 千万别 盲目的跟着书写!!!

编程环境的更换 -- 编程的软件很多很多

- EditPlus ： 非正常工作时使用，缺少提示. 入门使用
- HBuilder：国产的，适合其 uni-app 框架的内容书写
- VSCode ：全球占有率最高的 轻量级编程工具，微软出品
  - 非必要，教程在 xin88.top 中提供
- WebStorm： 收费的框架，功能很强大

全局作用域

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>作用域相关 09:22</title>
</head>

<body>
  <!--
    什么是作用域?
    答：代码运行时的生效范围.  有作用的领域

    有哪些作用域?
    - 全局作用域：这里的代码在哪里都能用，公共的
    - 局部作用域：函数运行时产生的，函数私有的
    - 块级作用域：ES6提供的
   -->

  <script src="./01.a.js"></script>
  <script src="./01.b.js"></script>
  <script>
```

```
    // 直接在脚本中书写的代码，都属于全局作用域
    // 全局中 声明的变量 存储在 顶级对象 -- window
    // window是浏览器提供的，存储了 浏览器提供的所有 API
    // JS在浏览器中运行，就可以操作 window 对象中提供的功能

    // 宿主环境：JS代码运行时所在的平台
    // 1. node.js  -- 2阶段
    // 2. 浏览器  -- 3阶段 - 研究运行在浏览器上的JS 有什么特点.

    var a = 10

    function b() { }

    // clg : 快速生成打印的脚本代码
    console.log(window)
    // 运行方式：  右键当前页面内容 -> open with live server
    // 会自动在 默认浏览器 开启当前页面，自带热更新功能

    // alert: 浏览器提供的 弹窗功能 函数
    // window.alert("Hello Window!")

    // 使用全局对象 window 中的属性时，可以省略 window 前缀
    alert('Hello Window!')

    // 全局(对象)污染:
    // 全局对象指的是 window 对象，本质功能是存储系统级的内容
    // 我们声明的变量默认也会存储在 window 对象里，  这并非window的主要职责，所以说：  自定义
变量 污染了 全局对象

    // 造成的问题:
    // 1. 属性名和系统属性重复，会导致覆盖
    // 2. 多个外部脚本中的变量 都存放在全局，会导致冲突覆盖!

    // 利用函数作用域来解决全局污染!
  </script>
</body>

</html>
```

window对象

- window 对象哪里来的?

  浏览器提供的
- window对象里的内容是做什么的

  提供操作浏览器的各种功能函数
- 如何使用window对象中的内容

  两种方式可以使用
  - window.属性名
  - 属性名 ： 全局中没有书写前缀，默认到window对象中查找使用

函数作用域

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>作用域-局部 09:53</title>
</head>

<body>
  <script>
    // 函数运行时产生的作用域，其中的变量使用范围仅限函数内部!
    // 函数分两种状态:
    // 1. 静态 -- 声明时
    // 2. 动态 -- 触发函数/调用函数，例如 函数名()
    function show() {
      // 这里的 x 属于 函数的私有财产，外部无法使用
      var x = 10

      // 函数中声明的变量，存储在函数的作用域对象中
      // 没有存储在全局window里，不会造成污染!
    }

    show() //动态

    console.log('x:', x) // 能打印出10吗?

      // 快速生成函数作用域的语法:  匿名函数自调用
      // var z = function () { }
      // z()

      ; (function () {
        var a = 10 // 属于函数作用域，不会造成污染!
        var b = 20
      })()
  </script>
</body>

</html>
```

作用域链

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>作用域链 10:21</title>
</head>

<body>
  <!-- 作用域链: 多层作用域嵌套时，当在某个作用域中使用一个变量，按照作用域链的原则 向上层就近查找 -->

  <script>
    // var a = 10

    function x() {
      // var a = 20
```

```
      function y() {
        // var a = 30

        function z() {
          // var a = 40
          console.log('a:', a)
        }
        z()
      }
      y()
    }
    x()
  </script>
</body>

</html>
```

闭包

为什么有闭包机制?

- 为了自身的正常运行，需要把用到的外部作用域存储在自身，防止其释放

Closure闭包:

- 一种称呼，代表函数作用域的一种状态： 被其他函数存储在 scopes 中,无法释放

官方描述:

- **闭包**（closure）是一个函数以及其捆绑的周边环境状态（lexical environment，**词法环境**）的引用的组合。换而言之，闭包让开发者可以从内部函数访问外部函数的作用域。在 JavaScript 中，闭包会随着函数的创建而被同时创建。

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>闭包 10:32</title>
</head>

<body>
  <script>
    // 设定: 函数运行时产生的内存会在函数执行完毕后自动销毁，节省内存
    // 除非 把变量存放在全局对象window，因为window永存!

    function displayName() {
      var name = 'mike'

      var x = function () {
        console.log('name:', name)
      }

      return x
    }

    // show 中存储的是 displayName() 函数中声明的变量x
    // 所以x函数不会释放!
```

```
    var show = displayName()
    // 这里能打印出 name，就说明 name 一定被存储了
    show()

    console.log(window)
    //scopes: 作用域 们  -- 存放函数声明时所在的作用域链 中的作用域
    //closure: 闭包 -- 封闭的包;  函数作用域生成的对象

    // 闭包 表达的是函数作用域的一种状态，被别人存储在 scopes 里了
    // 类似 妻子/丈夫 这种称呼，代表一个人的一种状态

    // 为什么要存闭包：防止其释放后，导致自身函数运行时，找不到对应的变量资源 而 运行失败!

    // 闭包的缺点：不可避免的消耗更多的内存
  </script>
</body>

</html>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>闭包 10:55</title>
</head>

<body>
  <script>
    // 闭包的原则：只存储用的到的值，杜绝内存的浪费
    // 存储顺序是 就近原则
    // 只存储外部作用域
    function x() {
      var a = 10, b = 20

      function y() {
        var b = 30, c = 40

        function z() {
          var d = 50
          // 使用了 a b c d 4个变量
          // 这些变量都属于哪些作用域?
          console.log(a, b, c, d);
        }
        // cdi:  direct直接输出. 以对象的方式打印函数
        // log: 按照一定的规则对输出内容进行美化，函数会以字符串模式打印
        console.dir(z)
      }
      y()
    }
    x()
  </script>
</body>

</html>
```

私有变量

公共变量：存储在全局中 能够被多个函数使用. 不安全 不可靠；-- 公共的电脑

私有变量：专门为 函数声明的变量 安全可靠，非共享 -- 个人笔记本电脑

声明的方式：

- 非全局变量： 局部变量，通过函数运行时产生
  - 利用 匿名函数自调用语法 快速生成作用域
- 利用 闭包 这个被动技能，把这个局部作用域保存在自身，后续随时使用.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>私有属性 11:26</title>
</head>

<body>
  <script>
    // 公共属性
    var a = 10  //全局区

    function x() {
      a++
      console.log('a:', a)
    }

    function y() {
      console.log('a:', a)
    }

    x()
    y()

    // 私有属性：只给某个函数使用的属性，其他函数无法使用
    // 案例：记录某个函数调用的次数

    // 利用匿名函数自调用语法 快速生成函数作用域/局部作用
    var show = (function () {
      // 这里的count 是函数作用域中的，不在全局中，所以非公有
      var count = 0  // alt + 上/下  可以调整代码位置

      function show() {
        count++
        console.log('调用次数:', count)
      }
      // show函数发动 被动技能 - 闭包，把当前的匿名函数作用域保存在自身的闭包中，属于show函数私有的财产
      console.dir(show)
      // 需要通过return的方式，把show函数暴露到全局中
      return show
    })()

    console.log(window)
    // 在全局中使用show函数
    show()
```

```
    show()
    show()


    ////////////////////////////////////
    // 标准的私有属性声明方式:
    var suibian = (function () {
      // 命名规范: 以 _ 作为私有属性的开头, 可以达到 见名知意的效果
      var _x = 10
      var _y = 20
      // var z = function () {
      //   console.log(_y, _x)
      // }
      // return z
      return function () {
        console.log(_y, _x)
      }
    })()

    suibian()
    suibian()
  </script>
</body>

</html>
```

上午知识点回顾

作用域

- 全局作用域
  - 顶级对象: window, 由浏览器提供的, 存放系统方法 (API) 的对象
    - 使用window中的属性, 可以直接调用

```
window.alert()
alert()
```

- 局部作用域
  - 函数在运行时产生的作用域, 属于一个私有的空间
  - 作用: 避免全局(变量/对象)污染
    - 在脚本中 自定义的属性, 默认存储在 window 对象里, 污染
    - 解决: 用匿名函数自调用语法, 快速生成局部作用域, 保存自定义的变量
- 块级作用域

作用域链:

- 当多层作用域嵌套时, 当在某个作用域下使用变量, 如果自身没有, 则按照就近原则 到上层作用域查找

闭包: 对于函数作用域的 一种状态的描述

- 状态: 被其他函数存储在 scopes 中
- 为什么: 为了确保函数的顺利执行, 需要把用到的外部作用域存储在自身scopes中, 防止其释放

闭包的使用场景: 私有属性

- 利用闭包的存储特性, 把 存放在函数作用域中的属性, 存储在自身使用

```
var 函数名 = (function(){
    var _私有属性 = 值

    return function(){}
})()
```

arguments

函数自带的属性:arguments

- 其中存储函数收到的所有实参

使用场景:

- 实参个数不固定的函数，例如 max min
- 函数重载：通过判断 实参的个数 或 类型 执行不同的代码逻辑，实现不同的效果!

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>arguments 14:10</title>
</head>

<body>
  <script>
    function show(x, y) {
      console.log(x, y)
      // 函数内部自带 arguments 变量，存储收到的所有实参
      console.log('arguments:', arguments)
    }

    show(11, 22, 'mike', 'lucy', 'lily')

    // 来自 Math 的 max 函数，求多个实参中的最大值
    console.log(
      Math.max(11, 2, 3, 43, 54),
      Math.max(11, 2, 3, 43, 54, 11, 23, 345, 6786),
    );

    // 仿写 max 函数
    function max() {
      // 利用 arguments 来查看收到的所有实参，进行进一步处理
      console.log('max:', arguments)
      // 思路:
      // 1. 先假设数组的第一个值是最大的
      // 2. 然后从第二个值开始比较，是否有更大的
      // 3. 如果有更大的，就替换成 找到的最大值，一直到最后
      var tmp_max = arguments[0] //临时最大值

      for (var i = 1; i < arguments.length; i++) {
        // 如果临时最大值 比 遍历出来的值 小
        if (tmp_max < arguments[i]) {
          // 替换临时最大值
          tmp_max = arguments[i]
        }
```

```
      }
      // 遍历结束后，返回最终的结果
      return tmp_max
    }

    console.log(
      max(11, 22),
      max(11, 22, 342, 43, 564, 76),
    )

    // 什么是编程?
    // 把人类解决问题的思维,转换成代码的方式表达出来.


    // 练习1: 制作一个min函数，找到实参中的最小值
    function min() {
      var tmp_min = arguments[0]

      for (var i = 1; i > arguments.length; i++) {
        if (tmp_min > arguments[i]) {
          tmp_min = arguments[i]
        }
      }

      return tmp_min
    }

    console.log(
      min(11, 22, 33),
      min(11, 22, 33, 34, 54, 65, 766),
    )

    // 练习2: 制作一个sum函数，求出实参的总和
    function sum() {
      var total = 0
      for (var i = 0; i < arguments.length; i++) {
        total += arguments[i]
      }
      return total
    }

    console.log(
      sum(11, 232, 43, 56),
      sum(54, 54, 23, 3, 5, 4, 10),
    );
  </script>
</body>

</html>
```

函数重载

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>函数的重载 15:07</title>
</head>

<body>
  <!-- 函数重载：根据实参的数量或类型不同，执行不同的逻辑操作；可以让函数变成多功能函数 -->
  <script>
    function zhekou(money) {
      // 判断实参的个数， 3个 就是满减
      if (arguments.length == 3) {
        if (money >= arguments[1]) { //超过满减条件
          money -= arguments[2]
        }
      }

      // 两个参数
      if (arguments.length == 2) {
        // 参数2：字符串类型 - 代表 vip
        // 参数2：数字类型 - 折扣
        if (typeof arguments[1] == 'number') {
          money *= arguments[1]
        }
        if (typeof arguments[1] == 'string') {
          if (arguments[1] == 'vip1') {
            money *= 0.95
          }
          if (arguments[1] == 'vip2') {
            money *= 0.8
          }
        }
      }

      return money
    }

    // zhekou ： 计算折扣价格的函数
    // 使用方式：
    console.log(zhekou(3000, 0.8)) // 消费3000 打八折
    console.log(zhekou(3000, 0.6)) // 6折

    console.log(zhekou(3000, 2000, 500)) // 满2000 减500
    console.log(zhekou(3000, 3000, 700)) // 满3000 减700

    console.log(zhekou(3000, 'vip1')) // 95折
    console.log(zhekou(3000, 'vip2')) // 8折

  </script>
</body>

</html>
```

练习

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>练习 15:30</title>
</head>

<body>
  <script>
    function calc() {
      if (arguments.length == 1) {
        if (typeof arguments[0] == 'number') {
          return arguments[0] * arguments[0]
        }
        // instanceof: 判断对象是否是指定构造函数创建的
        // 是 Array 创建的，则说明是数组类型
        if (arguments[0] instanceof Array) {
          var total = 0
          // arguments[0] 是 数组
          for (var i = 0; i < arguments[0].length; i++) {
            total += arguments[0][i]
          }
          return total
        }
      }

      if (arguments.length == 2) {
        return arguments[0] + arguments[1]
      }

      if (arguments.length == 3) {
        return arguments[0] * arguments[1] * arguments[2]
      }
    }

    // 实现一个 calc 函数，预期效果如下
    console.log(calc(10)) //算出数字的平方，即 10 * 10
    console.log(calc(20)) //算出数字的平方，即 20 * 20
    console.log(calc(10, 20)) //算出和值，即 10 + 20
    console.log(calc(10, 20, 30)) //算出乘积，即 10 * 20 * 30

    // 难
    console.log(calc([11, 22, 33])) //数组类型，则计算出数组内容的和
  </script>
</body>

</html>
```

this

是什么?

- 是函数中自带的一个变量，指向函数运行时所在的对象 -- **非常灵活**

使用场景?

- 为函数提供了另一种执行方式
  - 传统方案：通过传递参数的方式　把对象传入
  - 当前方案：把函数作为参数传递到对象里执行 -- 反过来

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>this 16:13</title>
</head>

<body>
  <!--
    this关键词：代表函数运行时所在的对象
    从格式上分辨：  对象.方法名()  方法中的this 就是 前面的对象
   -->

  <script>
    function show() {
      console.log('this:', this) // window
    }

    show() // 简写
    window.show() // 全局的show函数，在window对象里

    var emp = {
      ename: "亮亮",
      show: function () {
        console.log('this:', this)
      }
    }

    emp.show()
  </script>
</body>

</html>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>this使用场景 16:22</title>
</head>

<body>
  <script>
    // 矩形对象：
    var r1 = { length: 10, width: 20 }
    var r2 = { length: 100, width: 120 }

    // area函数：用于计算 矩形的面积  长 * 宽

    // 传统做法：
    function area(rect) {
      return rect.length * rect.width
    }

    console.log(
      area(r1), area(r2)
```

```
    );


    // this模式的函数
    function area1() {
      // this: 运行时服务的对象，所在的
      return this.length * this.width
    }

    // 1. 把 area1 存储到对象中 :  上门服务
    r1.area1 = area1
    console.log(r1)
    // 2. 执行 r1 中的 area1 函数
    console.log(
      r1.area1()
    )
    // 3. 执行完毕后，从对象中删除.
    delete r1.area1

    // 练习: 把 area1 放到 r2 执行
    // 1. 上门服务:  函数存入r2中
    r2.area1 = area1
    // 2. 在 r2 中执行
    console.log(r2.area1())
    // 3. 删除
    delete r2.area1
  </script>
</body>

</html>
```

练习

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>练习 16:45</title>
</head>

<body>
  <script>
    // 立方体: 长 宽 高
    var c1 = { length: 10, width: 20, height: 30 }

    // 制作一个函数 volume 计算体积 = 长 * 宽 * 高
    function volume() {
      // 在制作包含this变量的函数时，要提前预判 这个函数运行时的环境，才能知道 this 是什么
      return this.length * this.width * this.height
    }

    // 要求: 把 volume 传递到 c1 对象里执行，返回其体积
    // 1. 函数上门服务: 到c1对象里
    c1.volume = volume
    // 2. 调用
    console.log(c1.volume())
    // 3. 删除
    delete c1.volume
```

```
  </script>
</body>

</html>
```

call

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>call 17:11</title>
</head>

<body>
  <script>
    // call: 短暂拜访
    // 函数 具有一个短暂拜访对象的方法 -- call
    function area() {
      return this.length * this.width
    }

    var r1 = { length: 10, width: 20 }

    // 任务: 把 area 函数临时放到 r1 对象中执行
    console.log(
      area.call(r1) // area函数短暂拜访 r1 对象
    )

    console.dir(area) // dir: 直接输出函数对象

    // 练习: 立方体
    var c1 = { length: 10, width: 20, height: 30 }

    // 1. 制作 volume 函数, 计算 体积=长*宽*高
    // 把此函数放到 c1 中执行, 获取其体积
    function volume() {
      return this.length * this.width * this.height
    }
    console.log(volume.call(c1))

    // 2. 制作 area 函数, 计算 面积=(长x宽 + 长x高 + 宽x高)*2
    // 把此函数放到 c1 中执行, 获取其面积
    function area() {
      return (this.width * this.length + this.length * this.height + this.width *
this.height) * 2
    }

    console.log(area.call(c1))
  </script>
</body>

</html>
```

练习

```
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>练习 17:30</title>
</head>

<body>
  <script>
    var emp1 = { name: "亮亮", salary: 30000 } //月薪
    var emp2 = { name: "凯凯", salary: 10000 }
    var emp3 = { name: "泡泡", salary: 20000 }

    // 所得税汇算函数  shui
    // 年薪 10~ 15w   扣税 5%
    // 年薪 >15 ~ 20  扣税 10%
    // 年薪 >20 ~ 30  扣税 15%
    // 年薪 >30 ~ 40  扣税 20%
    // >40  扣税40%;   <10  不扣税
    function shui() {
      var total = 12 * this.salary // 年薪
      // 语法糖：代码在满足一些特定条件时，有简化写法．让程序员能够偷懒，像吃糖一样，令人愉
悦!
      // if判断的 {} 里只有一行代码，可以省略{}
      if (total < 10) return 0
      if (total >= 10 && total < 15) return total * 0.05
      if (total >= 15 && total < 20) return total * 0.1
      if (total >= 20 && total < 30) return total * 0.15
      if (total >= 30 && total < 40) return total * 0.2
      if (total >= 40) return total * 0.4
    }

    // 把shui函数，放到 emp1 2 3 中，分别计算扣税的金额
    console.log(shui.call(emp1))
    console.log(shui.call(emp2))
    console.log(shui.call(emp3))
  </script>
</body>

</html>
```

call的实参

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>call的参数 17:48</title>
</head>

<body>
  <script>
    var emp = { ename: '亮亮', salary: 20000 }
```

```
    // 制作一个函数，用于获取员工  n 个月的总薪资
    function total(n) {
      return this.salary * n
    }

    console.log(
      // 6个月
      // 参数1: 指定函数执行时所在的对象，即 this 的指向
      // 参数2 之后 : 传递给函数的实参列表
      total.call(emp, 6)
    )

    ////////////////////////
    var x = { a: 10 }

    function show(b, c, d) {
      return this.a + b + c + d
    }

    console.log(
      show.call(x, 20, 30, 40)
    );
  </script>
</body>

</html>
```

今日内容回顾

- 作用域
  - 全局
    - 顶级对象window - 浏览器提供，存放系统方法
  - 局部
    - 利用局部的函数作用域，避免全局污染
  - 块级
- 作用域链
  - 多层作用域嵌套，按照 就近原则 到上级作用域查找变量来使用
- 闭包 Closure
  - 是一种状态： 函数作用域被 别的函数保存在自身的scopes中，无法释放
- 函数中隐藏的变量
  - arguments: 保存所有实参
    - 实参个数不固定的函数
    - 函数重载: 多功能函数
  - this: 指向函数运行时所在的对象，特别灵活
    - 把函数 传递到 对象中执行
    - call: 短暂访问； 更加便捷的把 函数放到对象中执行

```
函数.call(对象，实参，实参...)
```

作业

- 复习 之前 JS基础
- 预习 明天的内容 -- FTP下载上个班的代码和笔记!

- 今天的内容 多加练习

> 不会的 看不懂的代码，直接AI问即可