\*\*\*\*

**#### 闭包:摆脱作用域的束缚**

**#### 闭包：函数和他周围状态的引用捆绑在一起的组合**

**#### 场景1 函数作为返回值**

```js

function mail() {

    const name = "";

    return function() {

        return name;

    }

}

const getName = mail();

console.log(getName());

```

**#### 场景2 函数作为参数**

```js

function ev(fn) {

    const content = 1;

    fn(content)

}

function mail(content) {

    console.log(content)

}

```

**#### 场景3函数嵌套**

```js

let num = 0;

function super() {

    function inner() {

        num++;

        console.log(num)

    }

    return inner;

}

```

**#### 场景4 事件处理（异步）的闭包**

```js

const lis = document.querySelector("li")

for(let i = 0; i < lis.length; i++) {

    (function(i) {

        lis[i].onClick = function() {}

    })(i)

    // (setTimeout(i) {}, 200) (i)

}

```

**#### 立即执行嵌套---按照作用域链查找**

```js

(function(a) {

    return (function(b) {

        console.log(a) //0

    })(2)

}, 200)(1)

```

**#### 立即执行遇上块级作用域**

```JS

const num = 0;

(function() {

  if(num = 0) {

      let num = 1;

      console.log(num)

  }

  console.log(num)

})()

```

**#### 拆分执行**

```js

function supers() {

    let num = 0;

    fucntion child() {

        num ++;

        console.log(num)

    }

    function son() {

        const message = `这里输出的是${num}`

    }

    return [child, son]

}

const [child, son] = supers();

child()

child()

child()

son() //这里输出的是0

```

**#### 实现私有变量**

```JS

function create() {

    return {

        arr: [],

        pushArr: function(item) {

            this.arr.push(item)

        }

    }

}

const obj = create()

obj.pushArr("0")

console.log(obj.arr)//["0"]

```

```JS

function createStack() {

    const arr = [];

    return{

        pushArr: function(item) {

            arr.push(item)

            return arr;

        }

    }

}

const obj = createStack()

const arr = obj.pushArr("8")

console.log(arr) //["8"]

```