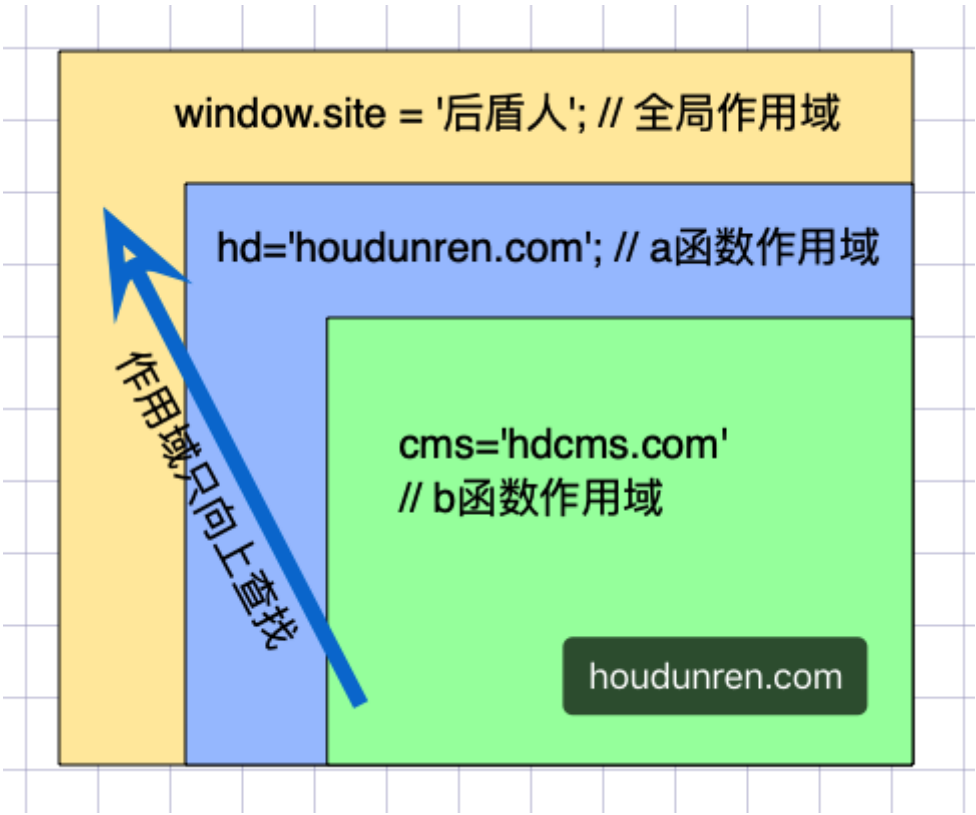
作用域

全局作用域只有一个，每个函数又都有作用域（环境）。

- 编译器运行时会将变量定义在所在作用域
- 使用变量时会从当前作用域开始向上查找变量
- 作用域就像攀亲亲一样，晚辈总是可以向上辈要些东西

#使用规范

作用域链只向上查找，找到全局 window 即终止，应该尽量不要在全局作用域中添加变量。



函数被执行后其环境变量将从内存中删除。下面函数在每次执行后将删除函数内部的 total 变量。

```
function count() {
  let total = 0;
}
count();

```

函数每次调用都会创建一个新作用域

```
let site = '后盾人';
```

```
function a() {
  let hd = 'houdunren.com';

  function b() {
      let cms = 'hdcms.com';
      console.log(hd);
      console.log(site);
  }
  b();
}
a();

```

如果子函数被使用时父级环境将被保留

```
function hd() {
  let n = 1;
  return function() {
    let b = 1;
    return function() {
      console.log(++n);
      console.log(++b);
    };
  };
}
let a = hd()();
a(); //2,2
a(); //3,3

```

构造函数也是很好的环境例子，子函数被外部使用父级环境将被保留

```
function User() {
  let a = 1;
  this.show = function() {
    console.log(a++);
  };
}
```

```
let a = new User();
a.show(); //1
a.show(); //2
let b = new User();
b.show(); //1

```

#let/const

使用 `let/const` 可以将变量声明在块作用域中（放在新的环境中，而不是全局中）

```
{
 let a = 9;
}
console.log(a); //ReferenceError: a is not defined
if (true) {
 var i = 1;
}
console.log(i);//1

```

也可以通过下面的定时器函数来体验

```
for (let i = 0; i < 10; i++) {
  setTimeout(() => {
    console.log(i);
  }, 500);
}

```

在 `for` 循环中使用`let/const` 会在每一次迭代中重新生成不同的变量

```
let arr = [];
for (let i = 0; i < 10; i++) {
        arr.push(() => i));
}
console.log(arr[3]()); //3 如果使用var声明将是10

```

在没有`let/const`的历史中使用以下方式产生作用域

```
//自行构建闭包
var arr = [];
for (var i = 0; i < 10; i++) {
  (function (a) {
      arr.push(()=>a);
  })(i);
}
console.log(arr[3]()); //3

```

#闭包使用

闭包指子函数可以访问外部作用域变量的函数特性，即使在子函数作用域外也可以访问。如果没有闭包那么在处理事件绑定，异步请求时都会变得困难。

- JS 中的所有函数都是闭包
- 闭包一般在子函数本身作用域以外执行，即延伸作用域

#基本示例

前面在讲作用域时已经在使用闭包特性了，下面再次重温一下闭包。

```
function hd() {
  let name = '后盾人';
  return function () {
        return name;
  }
}
let hdcms = hd();
console.log(hdcms()); //后盾人

```

使用闭包返回数组区间元素

```
let arr = [3, 2, 4, 1, 5, 6];
function between(a, b) {
  return function(v) {
```

```
    return v >= a && v <= b;
  };
}
console.log(arr.filter(between(3, 5)));

```

下面是在回调函数中使用闭包，当点击按钮时显示当前点击的是第几个按钮。

```
<body>
  <button message="后盾人">button</button>
  <button message="hdcms">button</button>
</body>
<script>
  var btns = document.querySelectorAll("button");
  for (let i = 0; i < btns.length; i++) {
    btns[i].onclick = (function(i) {
      return function() {
        alert(`点击了第${i + 1}个按钮`);
      };
    })(i);
  }
</script>

```

#移动动画

计时器中使用闭包来获取独有变量

```
<body>
  <style>
    button {
      position: absolute;
    }
  </style>
  <button message="后盾人">houdunren</button>
  <!-- <button message="hdcms">hdcms</button> -->
</body>
<script>
  let btns = document.querySelectorAll("button");
```

```
  btns.forEach(function(item) {
    let bind = false;
    item.addEventListener("click", function() {
      if (!bind) {
        let left = 1;
        bind = setInterval(function() {
          item.style.left = left++ + "px";
        }, 100);
      }
    });
  });
</script>

```

#闭包排序

下例使用闭包按指定字段排序

```
let lessons = [
  {
    title: "媒体查询响应式布局",
    click: 89,
    price: 12
  },
  {
    title: "FLEX 弹性盒模型",
    click: 45,
    price: 120
  },
  {
    title: "GRID 栅格系统",
    click: 19,
    price: 67
  },
  {
    title: "盒子模型详解",
    click: 29,
    price: 300
  }
```

```
];
function order(field) {
  return (a, b) => (a[field] > b[field] ? 1 : -1);
}
console.table(lessons.sort(order("price")));

```

#闭包问题

内存泄漏

闭包特性中上级作用域会为函数保存数据，从而造成的如下所示的内存泄漏问题

```
<body>
  <div desc="houdunren">在线学习</div>
  <div desc="hdcms">开源产品</div>
</body>
<script>
  let divs = document.querySelectorAll("div");
  divs.forEach(function(item) {
    item.addEventListener("click", function() {
      console.log(item.getAttribute("desc"));
    });
  });
</script>

```

下面通过清除不需要的数据解决内存泄漏问题

```
let divs = document.querySelectorAll("div");
divs.forEach(function(item) {
  let desc = item.getAttribute("desc");
  item.addEventListener("click", function() {
    console.log(desc);
  });
  item = null;
});

```

this 指向

this 总是指向调用该函数的对象，即函数在搜索 this 时只会搜索到当前活动对象。
下面是函数因为是在全局环境下调用的，所以 this 指向 window，这不是我们想要的。

```
let hd = {
  user: "后盾人",
  get: function() {
    return function() {
      return this.user;
    };
  }
};
console.log(hd.get()()); //undefined

```

使用箭头函数解决这个问题

```
let hd = {
  user: "后盾人",
  get: function() {
    return () => this.user;
  }
};
console.log(hd.get()()); //undefined
```