JavaScript基础--闭包和作用域

执行上下文和变量提升

- 范围:一段 `<script>` 或者一个函数
- 全局:变量声明、函数声明
- 函数:变量定义、函数声明、this、arguments

```
// 函数声明
sum(10,10); // 20，可以执行，函数声明提升
function sum(num1,num2){
  return num1 + num2;
}
```

```
// 函数表达式
sum(10,10);
// 报错：Uncaught TypeError: sum is not a function
// 函数在一个初始化语句中，不会出现变量提升
// 可以理解为 var sum = undefined;
var sum = function(num1,num2){
  return num1 + num2;
}
```

```
// 变量声明
console.log(a); //undefined
var a = 100;
// 全局函数声明
fn('张三'); // 张三 20
function fn(name){
  console.log(this); // Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}
  console.log(arguments); // Arguments ["张三", callee: ƒ, Symbol(Symbol.iterator): ƒ]

  age = 20;
  console.log(name,age);
  var age;
}
```

this

- `this` 在执行时才能确认值，定义时无法确认

```
var a = {
  name: 'A',
  fn: function() {
    console.log(this.name);
  }
}
a.fn(); // this ===a
a.fn.call({name: 'B'}); // this === {name:'B'}
var fn1 = a.fn;
```

```
fn1(); // this===window
```

- 作为构造函数执行

```
function Foo(name){
  this.name = name;
}
var f = new Foo('张三');
console.log(f.name); // 张三
```

- 作为对象属性执行

```
var obj = {
    name:'A',
    prientname:function(){
      console.log(this.name);
    }
}
obk.prientname(); // 张三
```

- 作为普通函数执行

```
function fn(){
  console.log(this);
}
fn() // Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}
```

- 作为 `call` 、 `apply` 、 `bind` 执行

```
function fn1(name,age){
  console.log(name,age);
  console.log(this);
}
fn1.call({x:100},'李四',25) //李四 25  {x: 100}
fn1.apply({x:100},['李四',25]) //李四 25  {x: 100}
var fn2 =function(name,age){
    console.log(name,age);
    console.log(this);
  }.bind({x:1000})
fn2('王五',123); // 王五 123 {x: 1000}
```

作用域

- js没有块级作用域
- 只有函数和全局作用域

```
// 无块级作用域
if(true){
  var name = 'xxxx';
}
console.log(name); // xxxx
```

```
// 函数和全局作用域
var a = 1000;
function fn() {
```

```
    var a = 200;
    console.log('fn:', a);
  }
  console.log('global', a); // global 1000
  fn(); // fn: 200
```

作用域链

- 执行时，一个自由变量一直往父级作用域中寻找直到找到，形成作用域链

```
// 函数和全局作用域
var a = 1000;
function fn() {
  var b = 200;
  // 当前作用域没有定义的变量，即 `自由变量`
  // 执行的时候，去父级作用域找变量 `a`
  console.log(a);
  console.log(b);
}
fn();
```

闭包

- `闭包` 是指有权访问另一个函数作用域中的变量的函数。创建闭包的常见方式，就是再一个函数内部创建另一个函数。

```
  function Fn() {
    var a = 1000;
    return function() {
      console.log(a);
    }
  }
  var a = 2000;
  var f = Fn();
  f(); // 1000
```

1. 函数作为返回值
2. 函数作为参数传递

```
  function Fn() {
    var a = 1000;
    return function() {
      console.log(a);
    }
  }

  var f = Fn();

  function f2(fn) {
    var a = 2000;
    fn();
  }
  f2(f); // 1000
```