继承

面向对象的特点

- 封装
- 继承
- 多态【抽象】

什么是继承

- 现实生活中的继承
- 程序中的继承

对象之间的继承

对象拷贝

for……in ：父对象的属性拷贝给子对象。

构造函数的属性继承

- 借用构造函数

构造函数的原型方法继承

- 拷贝继承（for-in）
- 原型继承

组合继承

函数定义方式

函数的定义方式

- 函数声明
- 函数表达式
- new Function

函数声明与函数表达式的区别

- 函数声明必须有名字
- 函数声明会函数提升，在预解析阶段就已创建，声明前后都可以调用
- 函数表达式类似于变量赋值
- 函数表达式可以没有名字，例如匿名函数
- 函数表达式没有函数提升，在执行阶段创建，必须在表达式执行之后才可以调用

函数也是对象

- 函数本身也是一种对象，可以调用属性和方法

函数的调用和 this

函数的调用方法

- 普通函数
- 构造函数
- 对象方法
- 事件函数
- 定时器、延时器的函数

函数内 this 指向的不同场景

- 函数的调用方式决定了 this 指向的不同：

调用方式	非严格模式	备注
普通函数调用	window	严格模式下是 undefined
构造函数调用	实例对象	原型方法中 this 也是实例对象
对象方法调用	该方法所属对象	紧挨着的对象
事件绑定方法	绑定事件对象	
定时器函数	window	

call、apply、bind

call

- call() 方法调用一个函数，其具有一个指定的 this 值和分别地提供的参数(参数的列表)。
- 注意：该方法的作用和 apply() 方法类似，只有一个区别，就是 call() 方法接受的是若干个参数的列表，而 apply() 方法接受的是一个包含多个参数的数组。
- 语法：

  fun.call(thisArg,arg1, arg2, arg3, ...)
- thisArg

  在 fun 函数运行时指定的 this 值

  如果指定了 null 或者 undefined 则内部 this 指向 window
- arg1, arg2, ...

  指定的参数列表

apply

- apply() 方法调用一个函数，第一个参数是一个指定的 this 值，第二个参数是以一个数组（或类似数组的对象）形式提供的参数。
- 注意：该方法的作用和 call() 方法类似，只有一个区别，就是 call() 方法接受的是若干个参数的列表，而 apply() 方法接受的是一个包含多个参数的数组。
- 语法：

  fun.apply(thisArg, [argsArray])

bind

- bind() 函数会创建一个新函数（称为绑定函数），新函数与被调函数（绑定函数的目标函数）具有相同的函数体（在 ECMAScript 5 规范中内置的 call 属性）。
- 当目标函数被调用时 this 值绑定到 bind() 的第一个参数，该参数不能被重写。绑定函数被调用时，bind() 也接受预设的参数提供给原函数。
- 一个绑定函数也能使用 new 操作符创建对象：这种行为就像把原函数当成构造器。提供的 this 值被忽略，同时调用时的参数被提供给模拟函数。
- 语法：

```
fun.bind(thisArg,arg1, arg2, arg3, ...)
```

bind

- 参数：
- thisArg：当绑定函数被调用时，该参数会作为原函数运行时的 this 指向。当使用 new 操作符调用绑定函数时，该参数无效。
- arg1, arg2, ...：当绑定函数被调用时，这些参数将置于实参之前传递给被绑定的方法。
- 返回值：

  返回由指定的 this 值和初始化参数改造的原函数拷贝。

总结

call 和 apply 特性一样

- 都是用来调用函数，而且是立即调用
- 但是可以在调用函数的同时，通过第一个参数指定函数内部 this 的指向
- call 调用的时候，参数必须以参数列表的形式进行传递，也就是以逗号分隔的方式依次传递即可
- apply 调用的时候，参数必须是一个数组，然后在执行的时候，会将数组内部的元素一个一个拿出来，与形参一一对应进行传递
- 如果第一个参数指定了 null 或者 undefined 则内部 this 指向 window

总结

bind

- 可以用来指定内部 this 的指向，然后生成一个改变了 this 指向的新的函数
- 它和 call、apply 最大的区别是：bind 不会调用
- bind 支持传递参数，它的传参方式比较特殊，一共有两个位置可以传递
  1. 在 bind 的同时，以参数列表的形式进行传递
  2. 在调用的时候，以参数列表的形式进行传递
- 那到底以谁 bind 的时候传递的参数为准呢还是以调用的时候传递的参数为准

  两者合并：bind 的时候传递的参数和调用的时候传递的参数会合并到一起，传递到函数内部

函数的其他成员

函数的其他成员

- arguments　　实参集合
- arguments.callee　　函数本身，arguments的一个属性
- fn.caller　　函数的调用者.如果在全局调用，返回的调用者为 null。
- fn.length　　形参的个数
- fn.name　　函数的名称

高阶函数

高阶函数

- 函数可以作为参数
- 函数可以作为返回值

函数闭包

回顾作用域、作用域链、预解析

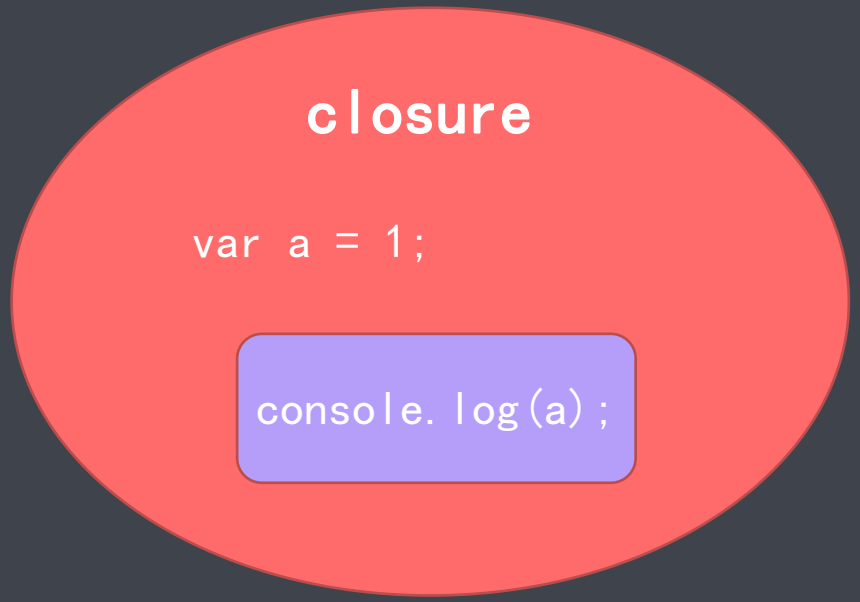
- 全局作用域
- 函数作用域
- 没有块级作用域
- 内层作用域可以访问外层作用域，反之不行

什么是闭包

- 一个函数和对其周围状态（lexical environment，词法环境）的引用捆绑在一起（或者说函数被引用包围），这样的组合就是闭包（closure）。也就是说，闭包让你可以在一个内层函数中访问到其外层函数的作用域。在 JavaScript 中，每当创建一个函数，闭包就会在函数创建的同时被创建出来。

闭包

- 函数定义时天生就能记住自己生成的作用域环境和函数自己，将它们形成一个密闭的环境，这就是闭包。不论函数以任何方式在任何地方进行调用，都会回到自己定义时的密闭环境进行执行。



观察闭包

- 从广义上来说，定义在全局的函数也是一个闭包，只是我们没办法将这样的函数拿到更外面的作用域进行调用，从而观察闭包的特点。
- 闭包是天生存在的，不需要额外的结构，但是我们为了方便观察闭包的特点，需要利用一些特殊结构将一个父函数内部的子函数拿到父函数外部进行调用，从而观察闭包的存在。

闭包的用途

- 可以在函数外部读取函数内部成员
- 让函数内成员始终存活在内存中

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容