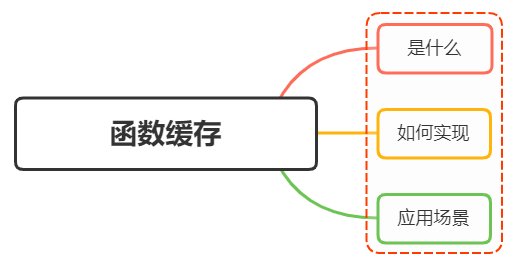
# 面试官：Javascript中如何实现函数缓存？函数缓存有哪些应用场景？



## 一、是什么

函数缓存，就是将函数运算过的结果进行缓存

本质上就是用空间（缓存存储）换时间（计算过程）

常用于缓存数据计算结果和缓存对象

const add = (a,b) => a+b;  
const calc = memoize(add); // 函数缓存  
calc(10,20);// 30  
calc(10,20);// 30 缓存

缓存只是一个临时的数据存储，它保存数据，以便将来对该数据的请求能够更快地得到处理

## 二、如何实现

实现函数缓存主要依靠闭包、柯里化、高阶函数，这里再简单复习下：

### 闭包

闭包可以理解成，函数 + 函数体内可访问的变量总和

(function() {  
 var a = 1;  
 function add() {  
 const b = 2  
 let sum = b + a  
 console.log(sum); // 3  
 }  
 add()  
})()

add函数本身，以及其内部可访问的变量，即 a = 1，这两个组合在⼀起就形成了闭包

### 柯里化

把接受多个参数的函数转换成接受一个单一参数的函数

// 非函数柯里化  
var add = function (x,y) {  
 return x+y;  
}  
add(3,4) //7  
  
// 函数柯里化  
var add2 = function (x) {  
 //\*\*返回函数\*\*  
 return function (y) {  
 return x+y;  
 }  
}  
add2(3)(4) //7

将一个二元函数拆分成两个一元函数

### 高阶函数

通过接收其他函数作为参数或返回其他函数的函数

function foo(){  
 var a = 2;  
  
 function bar() {  
 console.log(a);  
 }  
 return bar;  
}  
var baz = foo();  
baz();//2

函数 foo 如何返回另一个函数 bar，baz 现在持有对 foo 中定义的bar 函数的引用。由于闭包特性，a的值能够得到

下面再看看如何实现函数缓存，实现原理也很简单，把参数和对应的结果数据存在一个对象中，调用时判断参数对应的数据是否存在，存在就返回对应的结果数据，否则就返回计算结果

如下所示

const memoize = function (func, content) {  
 let cache = Object.create(null)  
 content = content || this  
 return (...key) => {  
 if (!cache[key]) {  
 cache[key] = func.apply(content, key)  
 }  
 return cache[key]  
 }  
}

调用方式也很简单

const calc = memoize(add);  
const num1 = calc(100,200)  
const num2 = calc(100,200) // 缓存得到的结果

过程分析：

* 在当前函数作用域定义了一个空对象，用于缓存运行结果
* 运用柯里化返回一个函数，返回的函数由于闭包特性，可以访问到cache
* 然后判断输入参数是不是在cache的中。如果已经存在，直接返回cache的内容，如果没有存在，使用函数func对输入参数求值，然后把结果存储在cache中

## 三、应用场景

虽然使用缓存效率是非常高的，但并不是所有场景都适用，因此千万不要极端的将所有函数都添加缓存

以下几种情况下，适合使用缓存：

* 对于昂贵的函数调用，执行复杂计算的函数
* 对于具有有限且高度重复输入范围的函数
* 对于具有重复输入值的递归函数
* 对于纯函数，即每次使用特定输入调用时返回相同输出的函数

## 参考文献

* https://zhuanlan.zhihu.com/p/112505577