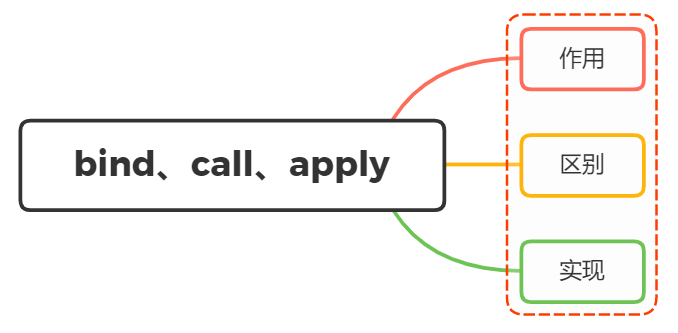
# 面试官：bind、call、apply 区别？如何实现一个bind?



## 一、作用

call、apply、bind作用是改变函数执行时的上下文，简而言之就是改变函数运行时的this指向

那么什么情况下需要改变this的指向呢？下面举个例子

var name = "lucy";  
var obj = {  
 name: "martin",  
 say: function () {  
 console.log(this.name);  
 }  
};  
obj.say(); // martin，this 指向 obj 对象  
setTimeout(obj.say,0); // lucy，this 指向 window 对象

从上面可以看到，正常情况say方法输出martin

但是我们把say放在setTimeout方法中，在定时器中是作为回调函数来执行的，因此回到主栈执行时是在全局执行上下文的环境中执行的，这时候this指向window，所以输出lucy

我们实际需要的是this指向obj对象，这时候就需要该改变this指向了

setTimeout(obj.say.bind(obj),0); //martin，this指向obj对象

## 二、区别

下面再来看看apply、call、bind的使用

### apply

apply接受两个参数，第一个参数是this的指向，第二个参数是函数接受的参数，以数组的形式传入

改变this指向后原函数会立即执行，且此方法只是临时改变this指向一次

function fn(...args){  
 console.log(this,args);  
}  
let obj = {  
 myname:"张三"  
}  
  
fn.apply(obj,[1,2]); // this会变成传入的obj，传入的参数必须是一个数组；  
fn(1,2) // this指向window

当第一个参数为null、undefined的时候，默认指向window(在浏览器中)

fn.apply(null,[1,2]); // this指向window  
fn.apply(undefined,[1,2]); // this指向window

### call

call方法的第一个参数也是this的指向，后面传入的是一个参数列表

跟apply一样，改变this指向后原函数会立即执行，且此方法只是临时改变this指向一次

function fn(...args){  
 console.log(this,args);  
}  
let obj = {  
 myname:"张三"  
}  
  
fn.call(obj,1,2); // this会变成传入的obj，传入的参数必须是一个数组；  
fn(1,2) // this指向window

同样的，当第一个参数为null、undefined的时候，默认指向window(在浏览器中)

fn.call(null,[1,2]); // this指向window  
fn.call(undefined,[1,2]); // this指向window

### bind

bind方法和call很相似，第一参数也是this的指向，后面传入的也是一个参数列表(但是这个参数列表可以分多次传入)

改变this指向后不会立即执行，而是返回一个永久改变this指向的函数

function fn(...args){  
 console.log(this,args);  
}  
let obj = {  
 myname:"张三"  
}  
  
const bindFn = fn.bind(obj); // this 也会变成传入的obj ，bind不是立即执行需要执行一次  
bindFn(1,2) // this指向obj  
fn(1,2) // this指向window

### 小结

从上面可以看到，apply、call、bind三者的区别在于：

* 三者都可以改变函数的this对象指向
* 三者第一个参数都是this要指向的对象，如果如果没有这个参数或参数为undefined或null，则默认指向全局window
* 三者都可以传参，但是apply是数组，而call是参数列表，且apply和call是一次性传入参数，而bind可以分为多次传入
* bind是返回绑定this之后的函数，apply、call 则是立即执行

## 三、实现

实现bind的步骤，我们可以分解成为三部分：

* 修改this指向
* 动态传递参数

// 方式一：只在bind中传递函数参数  
fn.bind(obj,1,2)()  
  
// 方式二：在bind中传递函数参数，也在返回函数中传递参数  
fn.bind(obj,1)(2)

* 兼容new关键字

整体实现代码如下：

Function.prototype.myBind = function (context) {  
 // 判断调用对象是否为函数  
 if (typeof this !== "function") {  
 throw new TypeError("Error");  
 }  
  
 // 获取参数  
 const args = [...arguments].slice(1),  
 fn = this;  
  
 return function Fn() {  
  
 // 根据调用方式，传入不同绑定值  
 return fn.apply(this instanceof Fn ? new fn(...arguments) : context, args.concat(...arguments));   
 }  
}