函数this的指向,call,apply,bind

call,apply,bind

为什么要用this?

this本来是以class为基石的面向对象语言设计类的时候，内部指向实例。 javascript是用函数和原型的方式模拟类,this存在函数里，根据执行环境不一样有不同的指向。

```
//普通函数及回调
let name = 'will00';
let age = 0;

//var  name = 'will00';
//var  age = 0;
function sayHello(){
    console.log(`My name is ${this.name}, I am ${this.age}`);
}

let obj01 = {
    name:'will01',
    age:11,
    sayHello(){
        console.log(`My name is ${this.name}, I am ${this.age}`);
    }
}

let obj02 = {
    name:'will02',
    age:12,
    _sayHello(cb){
        if(typeof cb === 'function'){
            cb(this);
        }
    },
    sayHello(cb){
        if(typeof cb === 'function'){
            cb.call(this);
        }
    }
}

let newSayHello = obj01.sayHello;
sayHello();
newSayHello();
obj01.sayHello();


obj02.sayHello(obj01.sayHello);
obj02._sayHello(obj01.sayHello);
obj02.sayHello(sayHello);
```

```
console.log(obj02);


//类函数及回调
let obj03 = {
    name:'will03',
    age:13,
    sayHello(cb){
        if(typeof cb === 'function'){
            cb.call(this);
        }
    }
}
function Person(name,age){
    this.name = name;
    this.age = age;
}

Person.prototype = {
    sayHello(){
        console.log(`My name is ${this.name}, I am ${this.age}`);
    }
}

let one = new Person('onePerson',25);
one.sayHello();
one.sayHello.call(obj03);
obj03.sayHello(one.sayHello);


//闭包及回调
function fun(){
    this.name = 'will';
    this.age = 100;
    this.obj = {
        name:'will0002',
        age:12
    }
    return function(){
        console.log(`My name is ${this.name}, I am ${this.age}`);
    }
}

fun()();
fun().call(obj);
```

ES5，在全局作用域下用var声明的变量,定义在window下，用let声明的变量则不是。 由以上代码可得，函数的内部this的指向,由函数【执行时的调用方】确定的。 确定方法，（call、apply,bind除外），【找到函数执行的小括号】，从右向左，看， 执行小括号 <= 函数名 <= 函数的调用方 执行小括号 <= (匿名函数)函数名 <= window/global

如何改变this指向

函数的灵活性也在于此。

```
function sayHello(){
    if(arguments[0]&&arguments[1]){
        this.name = arguments[0];
        this.age = arguments[1];
    }
    console.log(`My name is ${this.name}, I am ${this.age}`);
}

let obj01 = {
    name:'will01',
    age:11
}

let obj02 = {
    name:'will02',
    age:12,
    sayHello(){
        console.log(`My name is ${this.name}, I am ${this.age}`);
    }
}

let obj03 = {
    name:'will03',
    age:33
}

sayHello('namegogogo',111);
sayHello.call(obj01,'namegogogo',111);
sayHello.apply(obj01,['namegogogo',111]);
sayHello.bind(obj02)();
sayHello.bind(obj02)('namegogogo',111);


obj02.sayHello.call(obj03);
obj02.sayHello.apply(obj03);
obj02.sayHello.bind(obj03)();
```

bind和call/apply谁优先级更高

```
function sayHello(){
    if(arguments[0]&&arguments[1]){
        this.name = arguments[0];
        this.age = arguments[1];
    }
```

```
    console.log(`My name is ${this.name}, I am ${this.age}`);
}

let obj01 = {
    name:'will01',
    age:11
}

let obj02 = {
    name:'will02',
    age:12,
    sayHello(){
        console.log(`My name is ${this.name}, I am ${this.age}`);
    }
}

let obj03 = {
    name:'will03',
    age:33
}

let sayHelloNew = obj02.sayHello.bind(obj03);
sayHelloNew.call(obj01);
```

如何手动实现call\apply\bind

call执行时，改变一个函数的this指向,如何才能改变一个函数的this指向??? 看上文，函数this执行时如何确定的?

函数调用方和函数名之间是用.或[]连接，我们通过特定处理使得调用方和函数之间是通过 .或[]连接并执行的，即可完成改变this的指向。

```
/*
确定this的方法，（call、apply,bind除外）,【找到函数执行的小括号】，从右向左，看，
执行小括号 <= 函数名 <= 函数的调用方
执行小括号 <= 匿名函数名 <= window/global
*/
Function.prototype.myCall = function(obj){
    //这里的this是指向 function自己，是Function的一个实例
    //arguments for循环
    //Array.from
    //let myFunParams = Array.prototype.slice.apply(arguments,[1]);
    //还有哪些方法切割Array-like对象 arguments
    let myFunParams = [...arguments].slice(1);
    let funName = Symbol('funName');
    obj[funName] = this;
    obj[funName](...myFunParams);
    //防止污染obj对象
    delete obj[funName];
}
```

```
function one(height,gendar){
    console.log(`name = ${this.name}-- age = ${this.age}--
height=${height}--- gendar = ${gendar}`);
}
var obj01 = {
    name:'will',
    age:18
}
one.myCall(obj01,180,'male');
one(180,'male');
```

```
Function.prototype.myApply = function(obj,params=[]){
    //这里的this是指向 function自己，是Function的一个实例
    let funName = Symbol('funName');
    obj[funName] = this;
    obj[funName](...params);
    //防止污染obj对象
    delete obj[funName];
}
function one(height,gendar){
    console.log(`name = ${this.name}-- age = ${this.age}--
height=${height}--- gendar = ${gendar}`);
}
var obj01 = {
    name:'will',
    age:18
}
one.myApply(obj01,[180,'male']);
one(180,'male');
```

bind的实现，如果作为new 执行则，不做改变

```
Function.prototype.bind = function(obj){
    if (typeof this !== "function") {
      throw new Error("Function.prototype.bind - what is trying to be
bound is not callable");
    };
    let params = Array.from(arguments).slice(1);
    let exeFun = this;
    return function(){
        let funName = Symbol('funName');
        obj[funName] = exeFun;
        obj[funName](...params);
        delete obj[funName];
    }
}
function one(height,gendar){
    console.log(`name = ${this.name}-- age = ${this.age}--
height=${height}--- gendar = ${gendar}`);
```

```
}
var obj01 = {
    name:'will',
    age:18
}
one.bind(obj01,180,'male')();
one(180,'male');
```