**任务三 使用箭头函数**  P34

箭头函数起因

Class Person={

‘username’:’小暖’，’age’:’18’,

‘sayHello’:function(){

setInterval(function(){

console.log(‘我叫’+this.username+’我今年’+this.age+’岁’!)},1000}}

Person.sayHello();

输出结果为“我叫 undefined 今年 undefined ” ？因 setlnterval 是在全局作用域下执行的，所以 this 指向的是全局 window ，而 window 没有 usemame age 所以输出的是 undefined 。

通常的写法是缓存 this ，然后在 setlnterval i:j=i 用缓存的 this 进行操作

，明明是写在对象里面町机还要使用缓存对象才能正确使用 EC |

官方觉得这确实是个问题，因此在之后 ES6 的新特性里添加了箭头函数

2.3.2 箭头函数定义

语法：

（参数1，参数2，…,参数N）=>{函数声明}

（参数1，参数2，…,参数N）=>表达式（单一）

//相当于（参数1，参数2，…,参数N）=>{return 表达式}

//当只有一个参数时，圆括号是可选的

(单一参数)=>{函数声明} 单一参数=>{函数声明}

//没有参数的函数应该写成一对圆括号

()=>{函数声明}

例子： 箭头函数： ()=>console.log(“hello”) 普通函数：function(){ console.log(’hello’)}

与普通函数的区别： 与普通函数相比，箭头函数优点： 不绑定this,arguments 更简化的代码语法

不绑定this 不绑定this 可以理解为箭头函数的this在定义的时候就确定了，以后不管如何调用箭头函数。箭头函数的this始终为定义的this,

改写例子：

Function Person(){

This.name=’小暖’，

This.age=20,

setInterval(()=>{

console.log(‘我叫’+this.name+’我今年’+this.age+’岁’)}，1000）}

运行结果：我叫小暖，今年20岁

2.不绑定arguments 箭头函数不绑定arguments,如果在箭头函数中使用arguments参数会出问题。

例子： let arowfunc=(…theArgs)=>console.log(theArgs.length)

Arrowfunc(1,2) 运行结果：2

箭头函数不适用场景： 1.对象的方法 2.不能作为构造函数 3.定义原型方法

**任务6 Promise对象**

Promise是异步编程的一种解决方案，比传统的解决方案—回调函数和事件更强大。

基本用法：Promise对象是一个构造函数，用来生成Promise实例。

Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject,它们是两个函数，不用自己部署。

Resolve函数将Promise对象的状态从”未完成”变为”成功”（即从pending变为resolved）,在异步操作成功时调用，并将异步操作的结果作为参数传递出去。

Reject 函数的作用是将Promise对象的状态从“未完成”变为“成功”（即从pending）变为rejected),在异步操作失败时调用，并将异步操作的报错作为参数传递出去。

Then方法可以接受两个回调函数作为参数。

第一个回调函数是在Promise对象的状态变为resolved时调用，第二个回调函数是在Promise对象的状态变为rejected时调用。

第二个回调函数是可选的，不一定提供。它们接受Promise对象传出的值作为参数。

实例36：

Const Promise=new Promise(function(resolve,reject){

//..some code

If(/\*异步操作成功\*/){

Resolve(value);}

Else{reject(error);}});

Promise.then(function(value)){//success

},function(error){//failture});

)

作业：Promise小例子 ，结合SetTimeout

Const promise=new Promise(function(resolve,reject)){

setTimeout(resolve(1),1000); //1s后状态变更为resolve,调用then方法的1第一个参数

promise.then(function(value){

console.log(‘resolve:成功回调函数’)

console.log(value)

) ,function(){

Console.log(‘reject:失败回调函数’)

**setTimeout**

1.JavaScript提供定时执行代码的功能，叫做定时器（timer），主要由settimeout()和setInterval()两个函数完成，它们向任务队列添加定时任务。

基础： settimeout函数用来指定某个函数或者某段代码，在多少毫秒之后执行，他返回一个整数，表示定时器编号，以后可以用来取消这个定时器。

2.Var timerId=setTimeout(func|code,delay) SetTimeout接受两个参数，第一个参数fucn|code是将要延迟执行的函数名或者一段代码，第二个参数是推迟执行的毫秒数。

Console.log(1); setTimeout(‘console.log(2),1000’) console.log(3)

上面代码的输出结果就是1,3,2 因为SetTimeout指定第二行语句推迟1000毫秒之后执行。

3.推迟延迟代码必须以字符串形式，放入setTimeout.

如果推迟的是函数，则可以直接将函数名，放入setTimeout.

Function func(){

Console.log(2)

}

Var timerId=setTimeout(func(),1000)

或者 setTimeout(function(){console.log(2)},1000)

4. SetTimeout传参数

5.SetTimeout()回调函数中的this

如果被SetTimeout推迟执行的回调函数是某个对象的方法，那么该方法中的this关键字将指向全局变量，而不是定义时所在的那个对象。

Var x=1;

Var o={

X:2,

Y:function(){

Console.log(this.x);

}

};

setTimeout(o.y,1000) //1

代码输出是1，不是2，这表示o.y的this指向的已经不是o,而是全局环境了。

Funtion User(login){

This.login=login

this.sayHi = function() {

console.log(this.login);

}

}

var user = new User('John');

setTimeout(user.sayHi, 1000); // 输出undefined，全局环境

为了防止在全局对象中执行，一种解决方案是将user.sayHi放在匿名函数中执行。

setTimeout(function(){

user.sayHi();},1000) 在user作用域中执行

另一种方法，使用bind方法 ，