## 1 ES6 异步回调解决方案

三种：Promiss generator async

## 2 Promiss

### 2.1 Promiss的基础使用

let p = new Promise((resolve,reject)=>{  
 $.ajax({  
 url:'1.txt',  
 dataType:'json',  
 success(json){  
 console.log('正确');  
 resolve(json);  
 },  
 error(err){  
 console.log('错误');  
 reject(err);  
 }  
 });  
});  
  
*//其实这里第一个箭头函数就是resolve 第二个是reject*p.then(json=>{  
 console.log('成功');  
 console.log(json);  
},err=>{  
 console.log('失败');  
});

### 2.2. Promise.all()

let p1=new Promise((resolve, reject)=>{  
 $.ajax({  
 url: '1.txt',  
 dataType: 'json',  
 success(json){  
 resolve(json);  
 },  
 error(err){  
 reject(err);  
 }  
 });  
});  
  
let p2=new Promise((resolve, reject)=>{  
 $.ajax({  
 url: '2.txt',  
 dataType: 'json',  
 success(json){  
 resolve(json);  
 },  
 error(err){  
 reject(err);  
 }  
 });  
});  
  
Promise.all([p1, p2, p3]).then(arr=>{  
 let [j1, j2]=arr;  
  
 alert('成功');  
 console.log(j1, j2);  
}, err=>{  
 alert('失败');  
});

### 2.3 Promise与jQuery

jQuery本身返回的其实是一个Promise对象。

Promise.all([  
 $.ajax({url: '1.txt', dataType: 'json'}),  
 $.ajax({url: '2.txt', dataType: 'json'}),  
 $.ajax({url: '3.txt', dataType: 'json'}),  
]).then(arr=>{  
 let [j1, a, j2]=arr;  
  
 console.log(j1, a, j2);  
}, err=>{  
 alert('失败');  
});

### 2.4 Promise总结

Promise可以解除异步操作，但是具有局限性：如果异步操作中带了逻辑，Promise无法解决。

Promise.all() 所有都成功后才会执行后续；

Promise.race() 只要有一个完成就会执行后续操作。

## 3 generator

generator生成器可以用来暂停。

yield：后面一般跟一个Promise对象，将这个对象返回。

注意：generator 函数不能写成箭头函数。

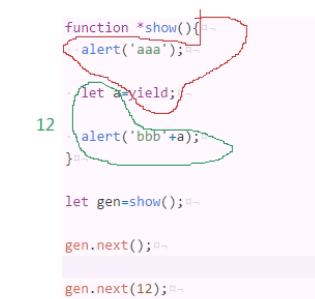
### 3.1 generator展示

function \*show() {  
console.log(11);  
yield;  
console.log(22);  
}  
*// show(); 直接输出没有结果*let gen = show(); *//执行生成器函数得到的是一个生成器对象*gen.next(); *//11 这样的停顿是为了给异步操作一定的时间*setTimeout(function () {  
 gen.next();  
},3000);

### 3.2 yield传参

function\* show() {  
 console.log(11);  
 let a = yield; *//yield传参* console.log(22 + a);  
}  
  
let gen = show();  
gen.next(); *//11*gen.next(10); *//32*

原理：



### 3.3 yield返回值

function \*show(){  
 alert('aaa');  
 yield 55;  
  
 alert('bbb');  
 return 89;  
}  
  
let gen=show();  
  
let res1=gen.next();  
console.log(res1); *//{value: 55, done: false}*let res2=gen.next();  
console.log(res2); *//{value: 89, done: true}*

### 3.4 generator与promise配合

function \*show(){  
 let data1=yield $.ajax({url: '1.txt', dataType: 'json'});  
  
 if(data1.a+data1.b<10){  
 let date2=yield $.ajax({url: '2.txt', dataType: 'json'});  
  
 alert('a'); *//12* }else{  
 let data3=yield $.ajax({url: '3.txt', dataType: 'json'});  
  
 alert('b'); *//blue* }  
}  
runner(show);

## 4 async

目前最优秀的异步解决方案。

语法：

async function(){

let 结果1 = await 异步操作(Promise generator async)

let 结果2 = await 异步操作(Promise generator async)

....

}

await后可以跟各种异步操作，当然也可以跟一个数值，这样就没有异步操作了，会直接跳过。

基本示例：

async function show(params) {

console.log(1);

await new Promise(function(resolve,rejec) {

setTimeout(function () {

resolve();

},3000);

});

console.log(2);

await new Promise(function(resolve,rejec) {

setTimeout(function () {

resolve();

},3000);

});

console.log(3);

}

show();

function sleep(sec){  
 return new Promise((resolve, reject)=>{  
 setTimeout(function (){  
 resolve();  
 }, sec\*1000);  
 })  
}

(async ()=>{  
 let data1 = await $.ajax({url: '1.txt', dataType: 'json'});  
 if(data1.a + data1.b < 10){  
 let data2 = await $.ajax({url: '2.txt', dataType: 'json'});  
 alert(data2[0]);  
 }else{  
 let data3 = await $.ajax({url: '3.txt', dataType: 'json'});  
 alert(data3.name);  
 }  
})();

async function show(){  
 try{  
 let data1=await $.ajax({url: '1.txt', dataType: 'json'});  
 let data2=await $.ajax({url: '33.txt', dataType: 'json'});  
 let data3=await $.ajax({url: '3.txt', dataType: 'json'});  
  
 console.log(data1, data2, data3);  
 }catch(e){  
 alert('有问题');  
 throw new Error('我错了....');  
 }  
}  
show();