

Promise

Promise 对象是一个构造函数，用来生成 Promise 实例

```
1  const promise = new Promise(function(resolve, reject) {
2    // ... some code
3
4    if (/* 异步操作成功 */){
5      resolve(value);
6    } else {
7      reject(error);
8    }
9  });
10
11
12  function f1(resolve, reject) {
13    // 异步代码...
14  }
15  var p1 = new Promise(f1);
```

Promise 实例生成以后，可以用 `then` 方法分别指定 `resolved` 状态和 `rejected` 状态的回调函数。

```
1  promise.then(function(value) {
2    // success
3  }, function(error) {
4    // failure
5  });
```

Promise 新建后就会立即执行。

```
1  let promise = new Promise(function(resolve, reject) {
2    console.log('Promise');
3    resolve();
4  });
5
6  promise.then(function() {
7    console.log('resolved. ');
8  });
9
10 console.log('Hi!');
11
```

```
12 // Promise
13 // Hi!
14 // resolved
```

异步模块同步执行

1. 如果Map里有异步，实际上将异步转同步执行。
2. 又希望让Map块在整个代码的顺序上同步执行，可以用Promise.all包一下。

比如这样

```
1  await Promise.all([5,6].map(async item => {
2    console.log(item, 'item3');
3    await getId();
4    console.log(item, 'item4');
5  }));
6  console.log('完成');
7  代码这个时候会输出这样的结果:
8  5 'item3'
9  6 'item3'
10 5 'item4'
11 6 'item4'
12 完成
```

```
// 处理map里有异步功能，转同步
await Promise.all([5,6].map(async item => {
  console.log(item, 'item3');
  await getId();
  console.log(item, 'item4');
}));
console.log('完成');
```

头条 @小郑搞码事

```

1 // 分组请求 请求返回错误不再执行剩余请求
2 async getMoney(requestPromises, num){
3     let flag= true;
4     let lineList = [];
5     for( let i=0; i<num; i++){
6         console.log(i);
7         await checkimportExcelByWeb(requestPromises[i]).then((res)=>{
8             if(!res.success) {
9                 showErrorMsg(res);
10                flag = false;
11            } else {
12                if(!isEmptyArray(res.data)) {
13                    const resList = res.data;
14                    lineList.push(...resList);
15                }
16            }
17        })
18        if(flag === false) {
19            break;
20        }
21    }
22    if(!isEmptyArray(lineList)) {
23        const errData = lineList.filter(item => {
24            return item.importResult === false;
25        });
26        if(!isEmptyArray(errData)) {
27            this.setState({importFilterData: errData, isError: true});
28        }
29        this.setState({sendBackImportData: lineList, progressStatus: 'success
30    }
31 }
32 // 分组请求
33 const groupArray = this.group(importDataNew,packageNum,num);
34 console.log(groupArray);
35 this.setState({progressStatus:'active', percentValue: 30});
36 const _that = this;
37 let requestPromises = [];
38 for (let index = 0; index < num; index++) {
39     const requestPromise = checkimportExcelByWeb({
40         ...defaultValue,

```

```

41         sourceType,
42         calibrationLineVos: groupArray[index],
43     })
44     requestPromises.push(requestPromise);
45 }
46 // this.getMoney(requestPromises, num);
47 Promise.all(requestPromises).then((res)=>{
48     if(!isEmptyArray(res)) {
49         const resPonseSuccess = res.map((item)=>{
50             return item.success;
51         });
52         if(resPonseSuccess.includes(false)) {
53             const messeges = res.map((item)=>{
54                 if(!item.success) {
55                     return item.message;
56                 }
57             });
58             const showError = {
59                 statusCode: 200,
60                 message: messeges.join(','),
61             }
62             showErrorMsg(showError);
63             _that.setState({progressStatus:'exception', isError: true
64         }
65         if(resPonseSuccess.includes(true)) {
66             let lineList = [];
67             let errData = [];
68             res.forEach((item)=>{
69                 if(item.success && !isEmptyArray(item.data)) {
70                     const list = item.data;
71                     lineList.push(...list);
72                 }
73                 // 失败行
74             });
75             if(!isEmptyArray(lineList)) {
76                 errData = lineList.filter(item => {
77                     return item.importResult === false;
78                 });
79                 if(!isEmptyArray(errData)) {
80                     this.setState({importFilterData: errData, isError

```

```

81         }
82     }
83     _that.setState({sendBackImportData: lineList, progressSta
84 }
85 }
86 })
87 }

```

```

1  // 校验物料+供应商+模具类型对应的条件价格是否用完
2  if(tableData.length) {
3      const requestPromises = tableData.map((item) => {
4          return {
5              supplierCode: record.biddSupplierCode,
6              materialCode: record.materialCode,
7              mouldType: item.mouldType,
8          };
9      });
10     const res = await this.fontCheck(requestPromises, tableData.length);
11     if (!res) {
12         return;
13     }
14 }
15
16 async fontCheck(requestPromises, num) {
17     let flag = true;
18     for (let i = 0; i < num; i++) {
19         await checkConditionType(requestPromises[i]).then((res)=> {
20             if(!res.success) {
21                 showErrorMsg(res);
22                 flag = false;
23             }
24         })
25         if(flag === false) {
26             break;
27         }
28     }
29     return flag;
30 }

```

多个请求并发执行

```
1  async function dbFuc(db) {
2    let docs = [{}, {}, {}];
3    let promises = docs.map((doc) => db.post(doc));
4
5    let results = await Promise.all(promises);
6    console.log(results);
7  }
8
9  // 或者使用下面的写法
10
11 async function dbFuc(db) {
12   let docs = [{}, {}, {}];
13   let promises = docs.map((doc) => db.post(doc));
14
15   let results = [];
16   for (let promise of promises) {
17     results.push(await promise);
18   }
19   console.log(results);
20 }
21
22
23 await Promise.all(promises)
24 可改成：
25 await* promises
26
```

Promise.all 具有并发执行异步任务的能力。但它的最大问题就是如果其中某个任务出现异常(**reject**)，所有任务都会挂掉，**Promise** 直接进入 **reject** 状态

ES2020新特性**Promise.allSettled**

- 1 使用 **Promise.all** 来并发三个接口，如果其中任意一个接口服务异常，状态是 **reject**，
- 2 这会导致页面中该三个区域数据全都无法渲染出来，
- 3 因为任何 **reject** 都会进入 **catch** 回调，很明显，这是无法接受的，如下：

```

4 Promise.all([
5     Promise.reject({code: 500, msg: '服务异常'}),
6     Promise.resolve({ code: 200, list: []}),
7     Promise.resolve({code: 200, list: []})
8 ])
9 .then((ret) => {
10     // 如果其中一个任务是 reject, 则不会执行到这个回调。
11     RenderContent(ret);
12 })
13 .catch((error) => {
14     // 本例中会执行到这个回调
15     // error: {code: 500, msg: "服务异常"}
16 })
17

```

Promise.allSettled 的优势

我们需要一种机制，如果并发任务中，无论一个任务正常或者异常，都会返回对应的的状态（fulfilled 或者 rejected）与结果（业务 value 或者 拒因 reason），在 then 里面通过 filter 来过滤出想要的业务逻辑结果，这就能最大限度的保障业务当前状态的可访问性，而 Promise.allSettled 就是解决这问题的。

```

1 Promise.allSettled([
2     Promise.reject({code: 500, msg: '服务异常'}),
3     Promise.resolve({ code: 200, list: []}),
4     Promise.resolve({code: 200, list: []})
5 ])
6 .then((ret) => {
7     /*
8         0: {status: "rejected", reason: {...}}
9         1: {status: "fulfilled", value: {...}}
10        2: {status: "fulfilled", value: {...}}
11    */
12    // 过滤掉 rejected 状态, 尽可能多的保证页面区域数据渲染
13    RenderContent(ret.filter((el) => {
14        return el.status !== 'rejected';
15    }));
16 });
17

```

不能在 forEach 里面 循环 异步请求,改为 for 循环语句

for循环中进行Promise异步操作的问题总结

```
1 // 联系方式
2 for ( let i=0; i<tableData.length; i++ ) {
3     //await的必须是个Promise
4     await GetSupplierContact({
5         supplierId: tableData[i].supplier.id,
6     }).then((res)=>{
7         console.log(res);
8         const contractObj = res.data && !isEmptyArray(res.data) && res.data[0];
9         tableData[i].contactUserName = contractObj.name;
10        tableData[i].contactUserTel = contractObj.mobile;
11    });
12 }
```

如何在 JS 循环中正确使用 async 与 await

Promise.all 对比

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <title>11_Promise相关面试题1</title>
7 </head>
8 <script>
9     const a = () => {
10         return new Promise((resolve, reject) => {
11             setTimeout(() => {
12                 resolve(1);
13             })
14         });
15     }
16     const b = () => {
17         return new Promise((resolve, reject) => {
18             setTimeout(() => {
19                 resolve(2);
20             })
21         });
22     }
```



```

21     });
22 }
23 const c = () => {
24     return new Promise((resolve, reject) => {
25         setTimeout(() => {
26             reject(3);
27         })
28     });
29 }
30 // a().then((res) => {
31 //     console.log(res);
32 // })
33 // Promise.all([a(), b(), c()]).then((res) => {
34 //     console.log(res);
35 // })
36 Promise.allSettled([a(), b(), c()]).then((res) => {
37     console.log(res);
38 })
39 </script>
40
41 <body>
42     <mark>测试 Promise.all 与 Promise.allSettled</mark>
43 </body>
44
45 </html>

```

for await ... of

```

1  function getTime(seconds){
2      return new Promise(resolve=>{
3          setTimeout(() => {
4              resolve(seconds)
5          }, seconds);
6      })
7  }
8
9  async function test(){
10     let arr = [getTime(2000),getTime(300),getTime(1000)]
11     for await (let x of arr){
12         console.log(x); // 2000  300 1000 按顺序的
13     }
14 }

```

```
13     }  
14     test()  
15 VM137:14 2000  
16 VM137:14 300  
17 VM137:14 1000
```