Promise

Promise对象是一个构造函数，用来生成Promise实例

```
const promise = new Promise(function(resolve, reject) {
  // ... some code

  if (/* 异步操作成功 */){
    resolve(value);
  } else {
    reject(error);
  }
});


function f1(resolve, reject) {
  // 异步代码...
}
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。

```
promise.then(function(value) {
  // success
}, function(error) {
  // failure
});
```

Promise 新建后就会立即执行。

```
let promise = new Promise(function(resolve, reject) {
  console.log('Promise');
  resolve();
});

promise.then(function() {
  console.log('resolved.');
});

console.log('Hi!');

// Promise
// Hi!
// 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

```
await Promise.all([5,6].map(async item => {
console.log(item, 'item3');
await getId();
```

```
console.log(item, 'item4');
}));
console.log('完成');
代码这个时候会输出这样的结果：
5 'item3'
6 'item3'
 5 'item4'
 6 'item4'
 完成
```


```
// 分组请求  请求返回错误不再执行剩余请求
async getMoney(requestPromises, num){
        let flag= true;
        let lineList = [];
        for( let i=0; i<num; i++){
            console.log(i);
            await checkimportExcelByWeb(requestPromises[i]).then((res)=>{
                if(!res.success) {
                    showErrorMsg(res);
                    flag = false;
                } else {
                    if(!isEmptyArray(res.data)) {
                        const resList = res.data;
                        lineList.push(...resList);
                    }
                }
            })
            if(flag === false) {
                break;
            }
        }
        if(!isEmptyArray(lineList)) {
            const errData =  lineList.filter(item => {
                return item.importResult === false;
            });
            if(!isEmptyArray(errData)) {
                this.setState({importFilterData: errData, isError: true});
            }
```

```
            this.setState({sendBackImportData: lineList, progressStatus: 'success',percentValu
        }
    }
     // 分组请求
    const groupArry = this.group(importDataNew,pacageNum,num);
            console.log(groupArry);
            this.setState({progressStatus:'active', percentValue: 30});
            const _that = this;
            let requestPromises = [];
            for (let index = 0; index < num; index++) {
                 const  requestPromise = checkimportExcelByWeb({
                        ...defaultValue,
                        sourceType,
                        calibrationLineVos: groupArry[index],
                    })
                 requestPromises.push(requestPromise);
            }
            // this.getMoney(requestPromises, num);
            Promise.all(requestPromises).then((res)=>{
                if(!isEmptyArray(res)) {
                    const resPonseSuccess = res.map((item)=>{
                        return item.success;
                    });
                    if(resPonseSuccess.includes(false)) {
                        const messeges = res.map((item)=>{
                            if(!item.success) {
                                return item.message;
                            }
                        });
                        const showError = {
                            statusCode: 200,
                            message: messeges.join(','),
                        }
                        showErrorMsg(showError);
                        _that.setState({progressStatus:'exception', isError: true});
                    }
                    if(resPonseSuccess.includes(true)) {
                        let lineList = [];
                        let errData = [];
                        res.forEach((item)=>{
                            if(item.success && !isEmptyArray(item.data)) {
                                const list = item.data;
                                lineList.push(...list);
                            }
                            // 失败行
                        });
                        if(!isEmptyArray(lineList)) {
                            errData =  lineList.filter(item => {
                                return item.importResult === false;
                            });
                            if(!isEmptyArray(errData)) {
```

```
                                this.setState({importFilterData: errData, isError: true});
                            }
                        }
                        _that.setState({sendBackImportData: lineList, progressStatus: 'success
                    }
                }
            })
        }
```

多个请求并发执行

```
async function dbFuc(db) {
  let docs = [{}, {}, {}];
  let promises = docs.map((doc) => db.post(doc));

  let results = await Promise.all(promises);
  console.log(results);
}

// 或者使用下面的写法

async function dbFuc(db) {
  let docs = [{}, {}, {}];
  let promises = docs.map((doc) => db.post(doc));

  let results = [];
  for (let promise of promises) {
    results.push(await promise);
  }
  console.log(results);
}


await Promise.all(promises)
可改成:
await* promises

```

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

```
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，
这会导致页面中该三个区域数据全都无法渲染出来，
因为任何 reject 都会进入 catch 回调，很明显，这是无法接受的，如下:
Promise.all([
    Promise.reject({code: 500, msg: '服务异常'}),
    Promise.resolve({ code: 200, list: []}),
    Promise.resolve({code: 200, list: []})
])
.then((ret) => {
     // 如果其中一个任务是 reject，则不会执行到这个回调。
```

```
    RenderContent(ret);
})
.catch((error) => {
    // 本例中会执行到这个回调
    // error: {code: 500, msg: "服务异常"}
})

```

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。

```
Promise.allSettled([
    Promise.reject({code: 500, msg: '服务异常'}),
    Promise.resolve({ code: 200, list: []}),
    Promise.resolve({code: 200, list: []})
])
.then((ret) => {
    /*
        0: {status: "rejected", reason: {...}}
        1: {status: "fulfilled", value: {...}}
         2: {status: "fulfilled", value: {...}}
     */
    // 过滤掉 rejected 状态，尽可能多的保证页面区域数据渲染
    RenderContent(ret.filter((el) => {
        return el.status !== 'rejected';
    }));
});

```

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

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

如何在 JS 循环中正确使用 async 与 await