博学谷
www.boxuegu.com

第十四章 异步操作和async函数

一样的在线教育，不一样的教学品质

目录 Contents

- 常见的异步编程
- async函数应用
- async函数应用案例
- 并行处理问题

1.什么是异步编程？

什么是异步编程？

所谓“异步”：简单说就是一个任务分成两段，先执行第一段，然后转而执行其他任务，等做好准备，再回头执行第二段任务；

示例：

读取文件进行处理，异步编程如下所示：

1.什么是异步编程？

什么是异步编程？

这种不连续的执行，叫做异步。相应地，连续的执行，就叫做同步；

2.常见的异步编程方式

- **回调函数**

所谓回调函数，就是把任务的第二段单独写在一个函数内，等重新执行该任务时，就直接调用该函数。

其英文为： callback，直译过来就是"重新调用"。

```
fs.readFile('/etc/passwd', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

2.常见的异步编程方式

- **Promise对象**

回调函数的问题是：**可能会出现多个回调函数嵌套。**

假如读取A文件之后，再读取B文件，代码如下：

```
fs.readFile(fileA, function (err, data) {
  fs.readFile(fileB, function (err, data) {
    // ...
  });
});
```

2.常见的异步编程方式

Promise对象可**解决多个回调函数嵌套**的问题，如下所示：

```
readFile(fileA)
.then(function(data){
  console.log(data.toString());
})
.then(function(){
  return readFile(fileB);
})
.then(function(data){
  console.log(data.toString());
})
.catch(function(err) {
  console.log(err);
});
```

2.常见的异步编程方式

◆ **Generator函数**

Generator函数，就是一个封装的异步任务，或者说是异步任务的容器。

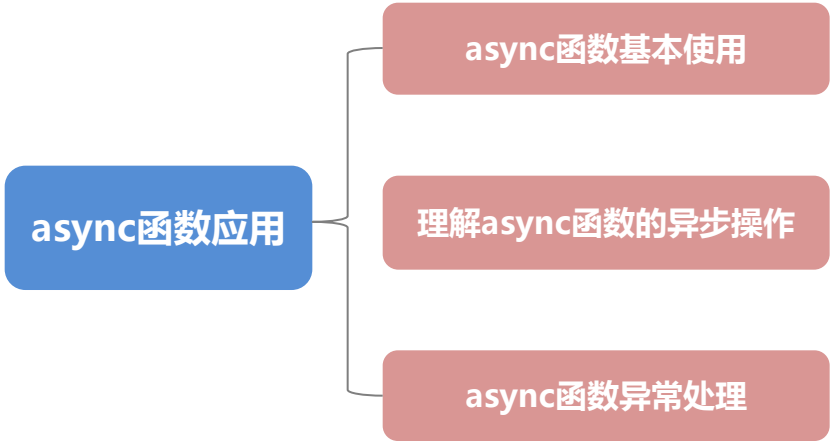
异步操作需要暂停的地方，都用到yield语句。

```
function* main() {
  let result = yield request("http://xxx.com/api");
  let resp = JSON.parse(result);
    console.log(resp.value);
}

function request(url) {
  makeAjaxCall(url, function(response){
    it.next(response);
  });
}

let it = main();
it.next();
```

目录 Contents

- 常见的异步编程
- async函数应用
- async函数应用案例
- 并行处理问题

async函数应用



1.基本用法

async函数可使异步操作更加简单；async函数是promise和generator的语法糖。

```
async function test() {
        let result = await Math.random();
        console.log(result);
    }
    test();
```

async：表示函数中有异步操作，await必须出现在 async 函数内部，不能单独使用。

await：表示紧跟其后的表达式需要等待结果。一般情况下，await后面是一个耗时的操作或者一个异步的操作。

2.理解async函数的异步操作

一般情况下await后面跟一个耗时的操作或一个异步的操作。

案例：

模拟一个耗时的操作；

步骤：

补充步骤

3.异常处理

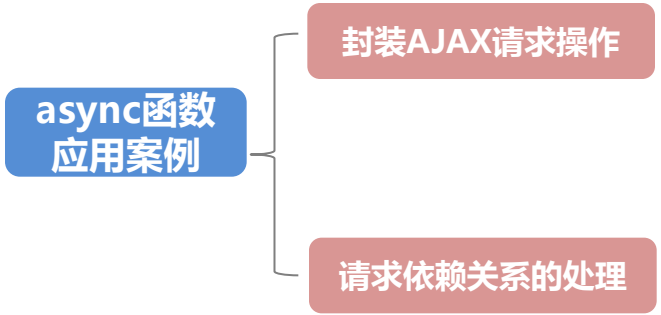
处理Promise中的异常：

- 通过try...catch捕获异常；
- 将结果返回后，通过catch方法捕获异常；

目录 Contents

- 常见的异步编程
- async函数应用
- async函数应用案例
- 并行处理问题

async函数应用案例



1.封装Ajax请求操作

使用async函数封装Ajax的操作：.

```
 async function getAJAX() {
            try {
                let result = await
getJSON('http://localhost:8081/api/getProductList');
                console.log(result);
            } catch (e) {
                console.log(e)
            }
        }
        getAJAX();
```

2.请求依赖关系的处理

在前面的案例中，只发送了一个请求，但是若需要发送三个请求，并且第三个请求依赖第二个请求返回的结果，第二个请求会依赖第一个请求的结果，应该如何处理？

```
function sleep(second, param) {
            return new Promise((resolve, reject) => {
                setTimeout(() => {
                    resolve(param);
                }, second);
            })
        }
        async function test() {
            let result1 = await sleep(2000, 'req01');
            let result2 = await sleep(1000, 'req02' + result1);
            let result3 = await sleep(500, 'req03' + result2);
            console.log(`
                            ${result3}
                        ${result2}
                        ${result1}
             `);
        }
        test();
```

2.请求依赖关系的处理

async函数的优势：

```
methods: {
     getLocation(phoneNum) {
         return axios.post('/xxxx/api', {
             phoneNum
         })
     },
     getFaceList(province, city) {
         return axios.post('/xxx/api/', {
             province,
             city
         })
     },
     getFaceResult () {
          this.getLocation(this.phoneNum).then(res => {
              if (res.status === 200 && res.data.success) {
              let province = res.data.obj.province;
              let city = res.data.obj.city;
                  this.getFaceList(province, city).then(res => {
                        if(res.status === 200 && res.data.success) {
                             this.faceList = res.data.obj
                        }
                  })
              }
         }).catch(err => {
             console.log(err)
         })
     }
}
```

2.请求依赖关系的处理

getFaceList方法嵌套的关系太过复杂，可通过async方法改造getFaceList方法：

```
async getFaceResult () {
                try {
                    let location = await
this.getLocation(this.phoneNum);
                    if (location.data.success) {
                        let province =
location.data.obj.province;
                        let city =
location.data.obj.city;
                        let result = await
this.getFaceList(province, city);
                        if (result.data.success) {
                            this.faceList =
result.data.obj;
                        }
                    }
                } catch(err) {
                    console.log(err);
                }
            }
```

目录 Contents

- 常见的异步编程
- async函数应用
- async函数应用案例
- 并行处理问题

并行处理问题

常见问题：

由于网速比较慢，用户访问的页面可能显示不出来，为了给用户一个友好的体验，一般都会显示一张loading图片，当页面展示出来后，将loading图片隐藏掉。

模拟：

假设用户访问某个页面，该页面有三个异步请求需要发送，且三个异步请求之间没有关联；

当三个请求都结束后才将页面中的loading图片隐藏。

一样的在线教育，不一样的教学品质