

等风来

博客园

首页

新随笔

联系

订阅

管理

随笔 - 664 文章 - 0 评论 - 19 阅读 - 118万

昵称：看风景就
园龄：7年5个月
粉丝：47
关注：18
[+加关注](#)

<	2021年11月						>
日	一	二	三	四	五	六	
31	1	2	3	4	5	6	
7	8	9	10	11	12	13	
14	15	16	17	18	19	20	
21	22	23	24	25	26	27	
28	29	30	1	2	3	4	
5	6	7	8	9	10	11	

搜索

常用链接


- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

我的标签

- JavaScript(175)
- css(84)
- react(56)
- 算法(38)
- nodejs(26)
- 移动端(25)
- vuejs(24)
- webpack(24)
- git(24)
- http(18)

async的基本用法

1. async函数的基本形式



```
//函数声明
async function foo() {}


//函数表达式
const foo = async function () {}

//对象的方法
let obj = { async foo() {} };
obj.foo().then(...)

//Class 的方法
class Storage {
  constructor() {
    this.cachePromise = caches.open('avatars');
  }
  async getAvatar(name) {
    const cache = await this.cachePromise;
    return cache.match(`/avatars/${name}.jpg`);
  }
}

const storage = new Storage();
storage.getAvatar('jake').then(...);

//箭头函数
const foo = async () => {};
```



2. async函数的返回值总是一个Promise

无论async函数有无await操作，其总是返回一个Promise。

1. 没有显式return，相当于return Promise.resolve(undefined);
2. return非Promise的数据data，相当于return Promise.resolve(data);
3. return Promise, 会得到Promise对象本身

async总是返回Promise，因此，其后面可以直接调用then方法，函数内部return返回的值，会成为then回调函数的参数
函数内部抛出的错误，会被then的第二个函数或catch方法捕获到



```
//正常返回值
async function f(){
  retrun 'hello world';
}

f().then(v => console.log(v)); //hello world

//抛出错误
async function f(){
  throw new Error('出错了');
}

f().then(
  v => console.log(v),
  e => console.log(e) //Error: 出错了
)
```

更多
随笔档案
2021年11月(6)
2021年10月(6)
2021年9月(3)
2021年8月(2)
2021年7月(4)
2021年6月(10)
2021年5月(1)
2021年4月(2)
2020年12月(1)
2020年11月(2)
2020年9月(10)
2020年8月(16)
2020年7月(9)
2020年6月(4)
2020年5月(94)
更多

阅读排行榜
1. ES6箭头函数总结(51029)
2. async的基本用法(46820)
3. React ref的用法(42035)
4. touch事件中的touches、targetTouches和changedTouches详解(40665)
5. vue数组对象修改触发视图更新(36842)

评论排行榜
1. ES6箭头函数总结(3)
2. React context基本用法(2)
3. rematch的基本用法(2)
4. phaser常用API总结(2)
5. websocket基本知识(1)


--



3. await操作符的值

[rv] = await expression (expression可以是任何值, 通常是一个promise)
--

expression是Promise, rv等于Promise兑现的值, 若Promise被拒绝, 则抛出异常, 由catch捕获 expression是非Promise, 会被转换为立即resolve的Promise, rv等于expression

await操作只能用在async函数中, 否则会报错。

4. async就是generator和promise的语法糖


<pre>//generator写法 var gen = function* () { var f1 = yield readFile('/etc/fstab'); var f2 = yield readFile('/etc/shells'); console.log(f1.toString()); console.log(f2.toString()); }; //async写法 var asyncReadFile = async function () { var f1 = await readFile('/etc/fstab'); var f2 = await readFile('/etc/shells'); console.log(f1.toString()); console.log(f2.toString()); };</pre>


async就是将 generator的 * 换成 async, 将 yield 换成 await。

5. async对generator的改进

1. 内置执行器

Generator必须依靠执行器调用next方法来自动执行, 例如co模块。而async函数自带执行器, 可以自动执行。

2. 更好的语义

async和await分别表示异步和等待, 语义更加明确

3. 适用性更强

co模块后面只能是Thunk函数或Promise对象, 而await后面可以是Promise或基本数据类型(如: 数字, 字符串, 布尔等)

4. 返回Promise, 可以继续操作

async函数总是返回一个Promise对象, 可以对其进行then调用, 继续操作后面的数据, 因此, async函数完全可以看作是多个Promise合成一个Promise对象, 而await命令就是内部的then调用。

6. async内部的并行调用

async配合await都是串行调用, 但是若有并行调用, 则应按照以下方式来写:

1. 变量分别接收Promise

<pre>let fooPromise = getFoo(); let barPromise = getBar(); let foo = await fooPromise(); let bar = await barPromise();</pre>
--

2. 使用Promise.all

<pre>let [foo,bar] = await Promise.all([getFoo(),getBar()]);</pre>
--

Promise.all这种写法有缺陷, 一个调用报错, 会终止, 这个不太符合并行调用的初衷。

3. 使用多个async函数

实际上, 一个async函数内部包含的调用应该是强相关的, 没有依赖关系的函数调用不应该放在一个async函数中, 分开来逻辑更清晰。

4. 并行执行的一些写法

1. 不能再内部非async function中使用await

推荐排行榜
1. touch事件中的touches、targetTouches和changedTouches详解(9)
2. ES6箭头函数总结(4)
3. React ref的用法(3)
4. git中HEAD^和HEAD~区别(2)
5. npm link用法总结(2)

最新评论
1. Re:git中HEAD^和HEAD~区别
请问以下， 母亲 和 父亲 有什么区别？
--ZRHW菜鸟
2. Re:svg的viewport和viewbox
图片例子很形象 感谢
--幻世无双520
3. Re:vue数组对象修改触发视图更新
this.\$set(this,arr[...newArr]) 可以解决你们的问题
--_Raymond
4. Re:React高阶组件总结
牛逼啊
--yuyu168
5. Re:ES6箭头函数总结
@闹太套 花括号作为作用域（块级作用域）是 ES6 才引入的，必须使用 let 声明，之前都只函数才有单独作用域；这个例子里面 a 没有处于块里面所以并不在块级作用域里，而箭头函数的特性使得 this...
--Ahacad




```
async function dbFuc(db) {
  let docs = [{}, {}, {}];
  // 报错，forEach的function是非async，不能使用await
  docs.forEach(function (doc) {
    await db.post(doc);
  });
}

//这里不需要 async
function dbFuc(db) {
  let docs = [{}, {}, {}];
  // 可能得到错误结果，这样调用也不能得到正确的结果
  docs.forEach(async function (doc) {
    await db.post(doc);
  });
}
```



2. 循环调用await可以使用for循环或for of循环



```
//for of
async function dbFuc(db) {
  let docs = [{}, {}, {}];

  for (let doc of docs) {
    await db.post(doc);
  }
}

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

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


//map + for of
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);
}

//for循环中去请求网页，若await操作成功，会break退出；若失败，会catch捕获，进入下一轮循环
const superagent = require('superagent');
const NUM_RETRIES = 3;

async function test() {
  let i;
  for (i = 0; i < NUM_RETRIES; ++i) {
    try {
      await superagent.get('http://google.com/this-throws-an-error!');
      break;
    } catch(err) {}
  }
  console.log(i); // 3
}

test();
```



7. async的错误处理

使用try...catch进行包裹，例如：



```

async function myFunction() {
  try {
    await somethingThatReturnsAPromise();
  } catch (err) {
    console.log(err);
  }
}

```



如果仅仅对一部分错误进行处理或者忽略，可以局部的进行包裹，或者对单独的promise进行catch，例如：



```

async function myFunction() {
  await somethingThatReturnsAPromise().catch((err)=> {
    console.log(err);
  })
}

async function myFunction() {
  try{
    await somethingThatReturnsAPromise();
  }
  catch(e){}
  await somethingElse();
}

```



Promise的错误处理，推荐用async + await来写:



```

// 存值
createData(title, successBack, errorBack) {
  // 使用key保存数据
  storage.save({
    key: title,
    data: 'true',
  }).then(successBack(), errorBack());
}

```



改写为



```

//存值
async createData1(title, successBack, errorBack) {
  try {
    // 使用key保存数据
    await storage.save({
      key: title,
      data: 'true',
    });
    successBack()
  } catch (e) {
    errorBack()
  }
}

```



形式上更加清晰一些。

8. async函数的实现原理

async函数就是将执行器和Generator做为一个整体返回。



```

async function fn(){}
//等同于
function fn(){
  return spawn(function* (){

```

```
    })  
  }  
}
```

spawn的实现

```
function spawn(genF) {  
  /****  
  * 返回的是一个promise  
  */  
  return new Promise(function(resolve, reject) {  
    var gen=genF(); //运行Generator这个方法;  
    /****  
    * 执行下一步的方法  
    * @param fn 一个调用Generator方法的next方法  
    */  
    function step(fn) {  
      //如果有错误, 则直接返回, 不执行下面的await  
      try {  
        var next=fn();  
      } catch (e) {  
        return reject(e)  
      }  
      //如果下面没有yield语句, 即Generator的done是true  
      if(next.done) {  
        return resolve(next.value);  
      }  
      Promise.resolve(next.value).then((val)=>{  
        step(function(){ return gen.next(val) })  
      }).catch((e)=>{  
        step(function(){ return gen.throw(e) })  
      })  
    }  
    step(function () {  
      return gen.next();  
    })  
  });  
}
```

参考: <https://segmentfault.com/a/1190000008677697>
<https://juejin.im/post/5b56837c6fb9a04fe0181555>
<https://www.cnblogs.com/goloving/p/8013119.html>
<https://blog.csdn.net/u011272795/article/details/80197481>

标签: [ES6](#) [JavaScript](#)

好文要顶

关注我

收藏该文



看风景就

关注 - 18

粉丝 - 47

+加关注

« 上一篇: [js判断设备类型](#)

» 下一篇: [ES6箭头函数总结](#)

1

0

posted @ 2018-09-15 17:18 看风景就 阅读(46820) 评论(0) 编辑 收藏 举报

[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论


编辑

预览

B



支持 Markdown

 自动补全

提交评论

退出

订阅评论

我的博客

[Ctrl+Enter快捷键提交]

编辑推荐：

- 巧用滤镜实现高级感拉满的文字快闪切换效果
- 记一次 .NET 某公交卡扣费系统 程序卡死分析
- 设计系统，设计和开发之间的 “DevOps”
- 3D 穿梭效果？使用 UWP 也能搞定
- 理解ASP.NET Core - 日志(Logging)

最新新闻：

- 比亚迪员工猝死在出租屋：生前一个月曾连续夜班 每班12小时 (2021-11-17 22:25)
- 给“浪潮们” 提个醒：提倡加班不是健康的企业文化 (2021-11-17 22:19)
- 千亿市值腰斩、增速放缓，泡泡玛特让资本买账有多难？ (2021-11-17 22:12)
- 市值超苹果，但属于微软的时代没有到来 (2021-11-17 22:00)
- 前有小鹏理想、后有特斯拉，减配交付会成为潮流吗？ (2021-11-17 21:19)
- » 更多新闻...