异步与 Promise

JS 异步编程模型

版权声明

联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03

什么是异步？什么是同步？

网上的解释经常混淆异步与回调

异步

- 如果能直接拿到结果
  - 那就是同步
  - 比如你在医院挂号，你拿到号才会离开窗口
  - 同步任务可能消耗 10 毫秒，也可能需要 3 秒
  - 总之不拿到结果你是不会离开的
- 如果不能直接拿到结果
  - 那就是异步
  - 比如你在餐厅门口等位，你拿到号可以去逛街
  - 什么时候才能真正吃饭呢?
  - 你可以每10分钟去餐厅问一下（轮询）
  - 你也可以扫码用微信接收通知（回调）

异步举例

- 以 AJAX 为例
  - request.send() 之后，并不能直接得到 response
  - 不信 console.log(request.response) 试试
  - 必须等到 readyState 变为 4 后，浏览器回头调用 request.onreadystatechange 函数
  - 我们才能得到 request.response
  - 这跟餐厅给你发送微信提醒的过程是类似的
- 回调 callback
  - 你写给自己用的函数，不是回调
  - 你写给别人用的函数，就是回调
  - request.onreadystatechange 就是我写给浏览器调用的
  - 意思就是你(浏览器)回头调一下这个函数
  - 在中文里，「回头」也有「将来」的意思，如「我回头请你吃饭」

回调

写了却不调用，给别人调用的函数，就是回调

「回头你调用一下呗」

大家意会

回调举例

- 把函数 1 给另一个函数 2

```
function f1(){}
function f2(fn){
  fn()
}
f2(f1)
```

- 分析

1. 我调用 f1 没有？答：没有调用
2. 我把 f1 传给 f2（别人）了没有？答：传了
3. f2 调用 f1 了没有？答：f2 调用了 f1

✓ 那么，f1 是不是我写给 f2 调用的函数？答：是

✓ 所以，f1 是回调。

抬杠 1

```
function f1(){}
function f2(fn){
  // fn()
}
f2(f1)
```

- 如果 f2 没有调用 f1 呢?
  - ✓ f2 有病啊? 它不调用 f1 那它为什么要接受 fn 参数

抬杠 2

```
function f1(){}
function f2(fn){
  fn()
}
f2('字符串')
```

- 如果我传给 f2 的参数不是函数呢
  - ✓ 你有病啊？用函数之前不看看函数的文档吗？
  - ✓ 会报错：fn 不是一个函数。看到报错你不就知错了？

抬杠 3

```
function f1(x){
  console.log(x)
}
function f2(fn){
  fn('你好')
}
f2(f1)
```

- f1 怎么会有一个 x 参数
- ✓ fn('你好') 中的 fn 就是 f1 对吧
- ✓ fn('你好') 中的'你好'会被赋值给参数 x 对吧
- ✓ 所以 x 就是 '你好' 啊！有什么难以理解的?
- ✓ x 可以改成任意其他名字，x 表示第一个参数而已

异步和回调的关系

- 关联
  - 异步任务需要在得到结果时通知 JS 来拿结果
  - 怎么通知呢?
  - 可以让 JS 写留一个函数地址(电话号码)给浏览器
  - 异步任务完成时浏览器调用该函数地址即可(拨打电话)
  - 同时把结果作为参数传给该函数(电话里说可以来吃了)
  - 这个函数是我写给浏览器调用的，所以是回调函数
- 区别 不一定非要用，也可以用轮询
  - 异步任务需要用到回调函数来通知结果
  - 但回调函数不一定只用在异步任务里
  - 回调可以用到同步任务里
  - array.forEach(n => console.log(n)) 就是同步回调

我特么怎么知道一个函数是同步还是异步？

很简单，根据特征或文档

判断同步异步

- 如果一个函数的返回值处于
  - ✓ setTimeout
  - ✓ AJAX（即 XMLHttpRequest）
  - ✓ AddEventListener
  - ✓ 这三个东西内部，那么这个函数就是异步函数
  - ✓ 如果还有其他 API 是异步的，我会另行说明
- 等下
  - ✓ 我听说 AJAX 可以设置为同步的
  - ✓ 答：傻 X 前端才把 AJAX 设置为同步的，这样做会使请求期间页面卡住。

摇骰子

- 举例1

```
function 摇骰子(){
  setTimeout(()=>{ // 箭头函数
    return parseInt(Math.random() * 6) + 1
  },1000)
  // return undefined
}
```

- 分析

- ✓ 摇骰子() 没有写 return，那就是 return undefined
- ✓ 箭头函数里有 return，返回真正的结果
- ✓ 所以这是一个异步函数/异步任务
- ✓ 很多傻子居然分不清这两个 return 属于不同的函数
- ✓ 在座的各位有不少这样的傻子

摇骰子续

```
const n = 摇骰子()
console.log(n) // undefined
```

- 那么怎么拿到异步结果?
  - ✓ 答：可以用回调。写个函数，然后把函数地址给它

```
function f1(x){ console.log(x) }
摇骰子(f1)
```

  - ✓ 然后我要求摇骰子函数得到结果后把结果作为参数传给 f1

```
function 摇骰子(fn){
  setTimeout(()=>{ // 箭头函数
    fn(parseInt(Math.random() * 6) + 1)
  },1000)
}
```

摇骰子续

- 摇骰子函数不调用 fn 怎么办?
  - 答：不调？不调我neng死写代码的人（我自己）
- 简化为箭头函数
  - 由于 f1 声明之后只用了一次，所以可以删掉 f1

```
function f1(x){ console.log(x) }
摇骰子(f1)
```

  - 改为

```
摇骰子(x => {
  console.log(x)
})
```

  - 再简化为

```
摇骰子(console.log)
// 如果参数个数不一致就不能这样简化，有个面试题
```

总结

- 异步任务不能拿到结果
- 于是我们传一个回调给异步任务
- 异步任务完成时调用回调
- 调用的时候把结果作为参数
- 希望你已经理解上面过程

如果异步任务有两个结果成功或失败，怎么办

进一步思考

两个结果

- 方法一：回调接受两个参数呗

```
fs.readFile('./1.txt', (error, data)=>{
  if(error){ console.log('失败'); return }
  console.log(data.toString()) // 成功
})
```

- 方法二：搞两个回调呗

```
ajax('get','/1.json', data=>{}, error=>{})
// 前面函数是成功回调，后面函数是失败回调
ajax('get', '/1.json', {
  success: ()=>{}, fail: ()=>{}
})
// 接受一个对象，对象有两个 key 表示成功和失败
```

这些方法的不足

面试：为什么要用promise

- 不管方法一还是方法二，都有问题

1. 不规范，名称五花八门，有人用 success + error，有人用 success + fail，有人用 done + fail
2. 容易出现回调地域，代码变得看不懂
3. 很难进行错误处理

- 回调地域举例

```
getUser( user => {
  getGroups(user, (groups)=>{
    groups.forEach( (g)=>{
      g.filter(x => x.ownerId === user.id)
       .forEach(x => console.log(x))
    })
  })
})
```

这还只是四层回调，你能想象20层回调吗?

波动拳（Hadoken）

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

↓ ↘ → + 拳

怎么解决回调问题

- 有什么办法能解决这三个问题
  - ✓ 规范回调的名字或顺序
  - ✓ 拒绝回调地狱，让代码可读性更强
  - ✓ 很方便地捕获错误
- 前端程序员开始翻书了
  - ✓ 1976 年，Daniel P. Friedman 和 David Wise
  - ✓ 俩人提出 Promise 思想
  - ✓ 后人基于此发明了 Future、Delay、Deferred 等
  - ✓ 前端结合 Promise 和 JS，制订了 Promise/A+ 规范
  - ✓ 该规范详细描述了 Promise 的原理和使用方法
  - ✓ 我将由浅入深慢慢在课程中讲解 Promise

以 AJAX 的封装为例

来解释 Promise 的用法

```
ajax = (method, url, options)=>{
  const {success, fail} = options // 析构赋值，别再问
  const request = new XMLHttpRequest()
  request.open(method, url)
  request.onreadystatechange = ()=>{
    if(request.readyState === 4){
      // 成功就调用 success，失败就调用 fail
      if(request.status < 400){
        success.call(null, request.response)
      }else if(request.status >= 400){
        fail.call(null, request, request.status)
      }
    }
  }
  request.send()
}

ajax('get', '/xxx', {
  success(response){}, fail: (request, status)=>{}
}) // 左边是 function 缩写，右边是箭头函数，记下来别再问
```

Promise 说这代码太傻了

我们改成 Promise 写法

```
// 先改一下调用的姿势
ajax('get', '/xxx', {
  success(response){}, fail: (request, status)=>{}
})
// 上面用到了两个回调，还使用了 success 和 fail

// 改成 Promise 写法
ajax('get', '/xxx')
  .then((response)=>{}, (request, status)=>{})

// 虽然也是回调
// 但是不需要记 success 和 fail 了
// then 的第一个参数就是 success
// then 的第二个参数就是 fail

// 请问 ajax() 返回了个啥?
// 返回了一个含有 .then() 方法的对象呗
// 那么再请问如何得到这个含有 .then() 的对象呢?
// 那就要改造 ajax 的源码了
```

```
ajax = (method, url, options)=>{
  return new Promise((resolve, reject)=>{
    const {success, fail} = options
    const request = new XMLHttpRequest()
    request.open(method, url)
    request.onreadystatechange = ()=>{
      if(request.readyState === 4){
        // 成功就调用 resolve，失败就调用 reject
        if(request.status < 400){
          resolve.call(null, request.response)
        }else if(request.status >= 400){
          reject.call(null, request)
        }
      }
    }
    request.send()
  })
}
```

return new Promise((resolve, reject)=>{})

背下这五个单词即可，等你用熟了……

小结

如何让一个回调的异步函数变成promise的异步函数：

- 第一步
  - return new Promise((resolve,reject)=>{...})
  - 任务成功则调用 resolve(result)
  - 任务失败则调用 reject(error)
  - resolve 和 reject 会再去调用成功和失败函数
- 第二步
  - 使用 .then(success, fail) 传入成功和失败函数
- 点到为止
  - 先讲到这里，Promise 还有高级用法，以后说

我们封装的 ajax 的缺点

- post 无法上传数据
  - request.send(这里可以上传数据)
- 不能设置请求头
  - request.setRequestHeader(key, value)
- 怎么解决呢?
  - 花时间把 ajax 写到完美（有时间可以做）
  - 使用 jQuery.ajax（这个可以）
  - 使用 axios（这个库比 jQuery 逼格高）

jQuery.ajax

- 已经非常完美
  - ✓ 进入 jQuery 的文档，搜索 ajax，找到 jQuery.ajax
  - ✓ 看看参数说明，然后直接看代码示例
  - ✓ 看看 jQuery 的封装，就知道自己的封装是辣鸡了
- 封装优点
  - ✓ 支持更多形式的参数
  - ✓ 支持 Promise
  - ✓ 支持的功能超多
- 我们需要掌握 jQuery.ajax 吗?
  - ✓ 不用，现在的专业前端都在用 axios
  - ✓ 写篇博客罗列一下功能，就可以忘掉 jQuery 了

axios

- 目前最新的 AJAX 库
  - ✓ 显然它抄袭了 jQuery 的封装思路
  - ✓ 方方记得 axios 的 API 吗?
  - ✓ 不记得，但是我写了博客
  - ✓ 通过这个博客我们可以快速了解 axios 的用法
  - ✓ 推荐大家也可以通过写博客来学习一个库
- 代码示例

```
axios.get('/5.json')
  .then( response =>
    console.log(response)
  )
```

axios 高级用法

- JSON 自动处理
  - ✓ axios 如何发现响应的 Content-Type 是 json
  - ✓ 就会自动调用 JSON.parse
  - ✓ 所以说正确设置 Content-Type 是好习惯
- 请求拦截器
  - ✓ 你可以在所有请求里加些东西，比如加查询参数
- 响应拦截器
  - ✓ 你可以在所有响应里加些东西，甚至改内容
- 可以生成不同实例(对象)
  - ✓ 不同的实例可以设置不同的配置，用于复杂场景

封装！封装！封装！

初级程序员学习 API（包括 Vue / React 的 API）

中级程序员学习如何封装

高级程序员造轮子

课后作业

- Promise 问答题
  - ✓ 你需要自行查看 Promise 的 MDN 文档才能答对
- Axios 问答题
  - ✓ 你需要自行查看 axios 的文档才能答对
- 我们的项目会经常用到这俩玩意
  - ✓ 所以忘了也没关系

下节课讲 JSONP 与跨域

又是一个面试必考点，Promise 也是必考点哦