

第十三章 Promise对象

一样的在线教育，不一样的教学品质



目录 Contents

- ◆ Promise对象定义
- ◆ 模拟构建Promise对象
- ◆ Promise.prototype.then()方法
- ◆ Promise.prototype.catch()方法
- ◆ Promise.all()方法
- ◆ Promise.race()方法

小节导学

在开发中经常使用Ajax发送请求，可能出现如下的情况：

```
$.ajax(url, success() {  
    $.ajax(url2, success() {  
        $.ajax(url3, success() {  
  
        })  
    })  
})
```

在一个Ajax的回调中，又发送了另外一个Ajax请求；依次类推，导致多个回调函数的嵌套，且代码不够直观并且难以维护；这就是常说的**回调地狱**。

要解决回调地狱的问题，就要使用**Promise对象**。

1.Promise概念与基本使用

所谓的Promise就是一个对象，而Promise对象代表的是一个异步任务，即需要很长时间去执行的任务。

通过Promise对象，可将异步操作以同步操作的流程表达出来，避免层层嵌套的回调函数问题，即回调地狱的问题。

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(function() {
    let num = Math.random();
    if (num > 0.3) {
      resolve('成功了!')
    } else {
      reject('失败了')
    }
  }, 3000)
})
promise.then(function(value) {
  console.log(value);
}, function(reason) {
  console.log(reason);
})
```

使用Promise对象：

首先创建该对象，该对象的构造函数接受一个函数作为参数；

该函数有两个参数，分别是resolve和reject, resolve表示成功，reject表示失败。

■ 1.Promise概念与基本使用

Promise对象在处理异步操作时，如何取值？

◆ 通过 then方法进行取值

通过该方法可以获取Promise对象处理异步操作返回的结果；

◆ 原因

Promise对象在处理异步操作的时候，有可能返回成功的值，也有可能返回失败的原因；

then方法中的两个参数，皆属于回调函数；

第一个回调函数是Promise处理成功后调用的；

第二个回调函数是Promise处理失败后调用。

■ 2.使用Promise封装AJAX操作

使用Promise来封装真正的异步操作，即封装AJAX的操作，**AJAX是典型的异步操作。**

步骤：

- 创建一个getJSON方法；
- 在该方法中创建一个Promise对象，该对象中封装关于AJAX核心的对象XMLHttpRequest；

- **补充其他剩余步骤**



目录 Contents

- ◆ Promise对象定义
- ◆ 模拟构建Promise对象
- ◆ Promise.prototype.then()方法
- ◆ Promise.prototype.catch()方法
- ◆ Promise.all()方法
- ◆ Promise.race()方法

■ 模拟构建Promise对象

模拟编写一个Promise对象，深入理解Promise对象原理：

1.构建基本结构

2.异常处理

3.then方法的处理

4.基本测试

5.同步模式处理



目录 Contents

- ◆ Promise对象定义
- ◆ 模拟构建Promise对象
- ◆ Promise.prototype.then()方法
- ◆ Promise.prototype.catch()方法
- ◆ Promise.all()方法
- ◆ Promise.race()方法

■ Promise.prototype.then()方法

Promise实例具有then方法，即then方法定义在原型对象的Promise.prototype上；

作用：

为Promise实例添加状态改变时的回调函数。

then方法的第一个参数是Resolved状态的回调函数，第二个参数是Rejected状态的回调函数。

then方法返回的是一个新的Promise实例：

注意：

返回的不是原来的Promise实例；因此，可以采用链式写法，在then方法后面再调用另一个then方法。



目录 Contents

- ◆ Promise对象定义
- ◆ 模拟构建Promise对象
- ◆ Promise.prototype.then()方法
- ◆ Promise.prototype.catch()方法
- ◆ Promise.all()方法
- ◆ Promise.race()方法

■ Promise.prototype.catch()方法

应用场景：

- 用于指定发生错误的回调函数。
- 通过catch方法在then方法中添加第二个回调函数，来处理失败错误的情况。
- 使用catch方法来处理失败或者是错误。

采用catch写法的优点：

- then方法可以进行链式编程，可以在每个then方法中进行错误失败的处理，但过程复杂；使用catch方法处理，其中任何一个Promise对象抛出的错误都会被最后一个catch捕获；写法非常方便简单。
- 类似try..catch这种同步的写法，代码结构更加清晰，更容易理解。



目录 Contents

- ◆ Promise对象定义
- ◆ 模拟构建Promise对象
- ◆ Promise.prototype.then()方法
- ◆ Promise.prototype.catch()方法
- ◆ Promise.all()方法
- ◆ Promise.race()方法

■ Promise.all()方法

Promise.all() 方法可将多个Promise实例包装成一个新的Promise实例。

该方法可接受一个数组作为参数，数组中的成员都是Promise的实例。

语法如下：

```
let p=Promise.all([p1,p2,p3,...])
```

说明：

Promise实例都成功，则p的状态为成功状态，且数组中所有Promise实例的返回值组成一个数组，给p的回调函数。

若一个实例失败，则p的状态为失败状态；此时第一个失败的Promise实例的返回值会传递给p的回调函数。

■ Promise.all()方法

Promise.all()方法的应用场景：

同时异步请求多个数据操作时，且多个异步操作相关，即后续的操作步骤要依赖于这些异步操作。

例：一个支付操作需要用户的账户有余额，并且商品有库存，才能进行下一步的支付操作。

可使用Promise.all方法处理。

```
Promise.all([getUserBalance(), getInventory()])  
  .then(() => {  
    // 做一些下单和支付的操作  
  })  
  .catch(error => alert(`can't pay: ${error.message}`))
```



目录 Contents

- ◆ Promise对象定义
- ◆ 模拟构建Promise对象
- ◆ Promise.prototype.then()方法
- ◆ Promise.prototype.catch()方法
- ◆ Promise.all()方法
- ◆ Promise.race()方法

■ Promise.race()方法

Promise.race() 方法可将多个Promise实例包装成一个新的Promise实例。

该方法可接受一个数组作为参数，数组中的成员都是Promise的实例。

```
let p=Promise.race([p1,p2,p3,...])
```

说明：

以上Promise实例只要有一个成功，则p的状态就是成功状态，且将Promise实例中第一个成功的返回值传递给p的回调函数；同理，只要有一个失败，则p的状态为失败状态，且将Promise实例中第一个失败的返回值传递给p的回调函数。

总结：

Promise.race([p1, p2, p3])中最快获取哪个结果，则返回该结果，不论结果本身是成功状态还是失败状态。



一样的在线教育，不一样的教学品质