**Promise在各种场景的使用（Promise.all、Promise.allSettled、Promise.race、Promise.any）**

1、共有三个接口，接口二要根据接口一返回的数据判断是否执行，接口二需要接口一的返回值作为参数；接口三要根据接口二返回的数据判断是否执行，接口三需要接口二的返回值作为参数。

1. const getCatalogue = function (id) {
2. return new Promise((resolve, reject) => {
3. console.log('接口一');
4. resolve({result: true, text: 1});*// 在接口返回的值里加上 result=true,要确保result跟接口返回的字段不会重名*
5. })
6. }
7. const getWattHourMeter = function (id) {
8. return new Promise((resolve, reject) => {
9. console.log('接口二');
10. resolve({result: true, text: 2});*// 在接口返回的值里加上 result=true*
11. })
12. }
13. const getConsumption = function (id) {
14. return new Promise((resolve, reject) => {
15. console.log('接口三');
16. resolve({result: true, text: 3});*// 在接口返回的值里加上 result=true*
17. })
18. }
20. getCatalogue('one').then((res) => {
21. console.log(11, res)
22. *// 判断下一个接口是否运行*
23. if (false) {
24. console.log(11, 'if')
25. return Promise.resolve({result: false, id: 1});
26. *// 返回状态为已完成的Promise，这样会进入每一个then，在下一个步根据result判断then里面的代码要不要执行*
27. } else {
28. return getWattHourMeter('two')
29. }
30. }).then((res) => {
31. console.log(22, res)
32. if (!(typeof res == 'object' && res.result != null && res.result === false)) {*// result为false不执行，这个时候需要主要接口返回的res是不是对象*
33. *// 判断下一个接口是否运行*
34. if (false) {
35. console.log(22, 'if')
36. return Promise.resolve({result: false, id: 2});
37. } else {
38. return getConsumption('three')
39. }
40. } else {
41. *// 第二步不执行的时候还是得返回值，要不然第三步的res为会undefined，如果第三步判断了res是否为undefined也可以不写*
42. return Promise.resolve({result: false, id: 2});
43. }
44. }).then((res) => {
45. console.log(33, res)
46. *// 还是根据result判断是不是要继续执行*
47. if (!(typeof res == 'object' && res.result != null && res.result === false)) {
48. *//数据操作*
49. console.log(33, '执行数据操作');
50. }
51. }).catch((e) => {
52. console.log(e, 'catch');*//这样所有的报错信息都会进入catch*
53. })

2、共有两个接口，需要等两个接口都成功的返回数据之后再进行操作，其中任何一个接口报错都无法继续。

Promise.all 里面参数为一个数组，数组的每一项是一个返回promise 的函数调用。  
then 的第一个参数是所有promise都成功的调用，返回结果是一个数组，数组的每一项为函数promise 的返回结果。  
then 的第二个参数：返回结果有一个失败则执行失败的回调，拿到的是第一个失败的值。

Promise.all([fn(),fn()]) 都返回resolve(); 才能够拿到成功的返回值  
Promise.all([fn(),fn()]) 有一个返回reject(), 则进入catch(error), 拿到失败的返回值

1. const apiOne = function (id) {
2. return new Promise((resolve, reject) => {
3. resolve({result: true, text: 1});
4. })
5. }
7. const apiTwo = function (id) {
8. return new Promise((resolve, reject) => {
9. resolve({result: true, text: 2});
10. })
11. }
13. Promise.all([apiOne('one'), apiTwo('two')]).then(result => {
14. console.log(result, 'result');
15. }, error => {
16. *// 任何一个报错都会进入这里*
17. console.log(error, 'error');
18. })

3、共有两个接口，需要等两个接口都成功的返回数据之后再进行操作，但如果其中一个接口报错，不能影响另一个接口的数据。

3.1 使用Promise.all

1. const apiOne = function (id) {
2. return new Promise((resolve, reject) => {
3. resolve({result: true, text: 1});
4. }).catch(error => {
5. return error;*// 捕获error并返回error，这样即使报错，也还是会进入all的then方法*
6. })
7. }
9. const apiTwo = function (id) {
10. return new Promise((resolve, reject) => {
11. reject({result: true, text: 2});
12. }).catch(error => {
13. return error;*// 捕获error并返回error，这样即使报错，也还是会进入all的then方法*
14. })
15. }
17. Promise.all([apiOne('one'), apiTwo('two')]).then(result => {
18. console.log(result, 'result');
19. }, error => {
20. console.log(error, 'error');
21. })

3.2 使用Promise.allSettled

Promise.allSettled里面参数为一个数组，数组的每一项是一个返回promise 的函数调用。

Promise.allSettled()方法返回一个在所有给定的 promise 都已经fulfilled或rejected后的 promise，并带有一个对象数组，每个对象表示对应的 promise 结果。

1. const apiOne = function(id) {
2. return new Promise((resolve, reject) => {
3. resolve({result: true, text: 1});
4. });
5. }
7. const apiTwo = function(id) {
8. return new Promise((resolve, reject) => {
9. reject({result: true, text: 2});
10. });
11. }
13. Promise.allSettled([apiOne('one'), apiTwo('two')]).then(result => {
14. console.log(result, 'result');
15. }, error => {
16. console.log(error, 'error');
17. })

4、共有两个接口，任何一个接口先完成就可以，无论是成功还是报错。

Promise.race(iterable) 方法返回一个 promise，一旦迭代器中的某个 promise 解决或拒绝，返回的 promise 就会解决或拒绝。

Promise.race 里面参数为一个数组，数组的每一项是一个返回promise 的函数调用。

如果传的迭代是空的，则返回的 promise 将永远等待。

如果迭代包含一个或多个非承诺值和/或已解决/拒绝的承诺，则 Promise.race 将解析为迭代中找到的第一个值。

1. const apiOne = function (id) {
2. return new Promise((resolve, reject) => {
3. setTimeout(() => {
4. resolve({result: true, text: 1});
5. }, 1000)
6. })
7. }
9. const apiTwo = function (id) {
10. return new Promise((resolve, reject) => {
11. setTimeout(() => {
12. resolve({result: true, text: 2});
13. }, 2000)
14. })
15. }
17. Promise.race([apiOne('one'), apiTwo('two')]).then(result => {
18. console.log(result, 'result');
19. }, error => {
20. console.log(error, 'error');
21. })



5、共有两个接口，只处理第一个成功执行的函数。

Promise.any() 接收一个Promise可迭代对象，只要其中的一个 promise 成功，就返回那个已经成功的 promise 。如果可迭代对象中没有一个 promise 成功（即所有的 promises 都失败/拒绝），就返回一个失败的 promise 和AggregateError类型的实例，它是 Error 的一个子类，用于把单一的错误集合在一起。本质上，这个方法和 Promise.all() 是相反的。

1. const apiOne = function (id) {
2. return new Promise((resolve, reject) => {
3. setTimeout(() => {
4. reject({result: true, text: 1});
5. }, 1000)
6. })
7. }
9. const apiTwo = function (id) {
10. return new Promise((resolve, reject) => {
11. setTimeout(() => {
12. resolve({result: true, text: 2});
13. }, 2000)
14. })
15. }
17. Promise.any([apiOne('one'), apiTwo('two')]).then(result => {
18. console.log(result, 'result');
19. }, error => {
20. console.log(error, 'error');
21. })