Promise使用介紹

20190124 林郁翔

大綱

什麼是Promise

什麼是Promise

Promise 物件代表一個即將完成、或失敗的非同步操作，以及它所產生的值。

通常用來控制異步請求，使請求能照順序呼叫。

且他避免了callback造成的問題：

參考資料
：https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Promise

Callback

```
var floppy = require('floppy');

floppy.load('disk1', function (data1) {
    floppy.prompt('Please insert disk 2', function() {
        floppy.load('disk2', function (data2) {
            floppy.prompt('Please insert disk 3', function() {
                floppy.load('disk3', function (data3) {
                    floppy.prompt('Please insert disk 4', function() {
                        floppy.load('disk4', function (data4) {
                            floppy.prompt('Please insert disk 5', function() {
                                floppy.load('disk5', function (data5) {
                                    floppy.prompt('Please insert disk 6', function() {
                                        floppy.load('disk6', function (data6) {
                                            //if node.js would have existed in 1995
                                        });
                                    });
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});
```

Promise

```
fetch(TEST_URL, {
  method: 'GET',
  headers: {
    Accept: 'application/json'
  }
}).then(response => response.json())
  .then(responseJson => {
    console.log('1');
    return fetch(TEST_URL, {
      method: 'GET',
      headers: {
        Accept: 'application/json'
      }
    }) //將Promise拋出去
  })
  .then(response => response.json())
  .then(responseJson => {
    console.log('2');
    return fetch(TEST_URL, {
      method: 'GET',
      headers: {
        Accept: 'application/json'
      }
    }) //再將Promise拋出去
  })
  .then(responseJson => {
    console.log('3');
    console.log('載入完畢');
  })
```

Promise

consistency(一致性)：提供 resolve(),reject(),then(),done(),catch() 方法可以使用。

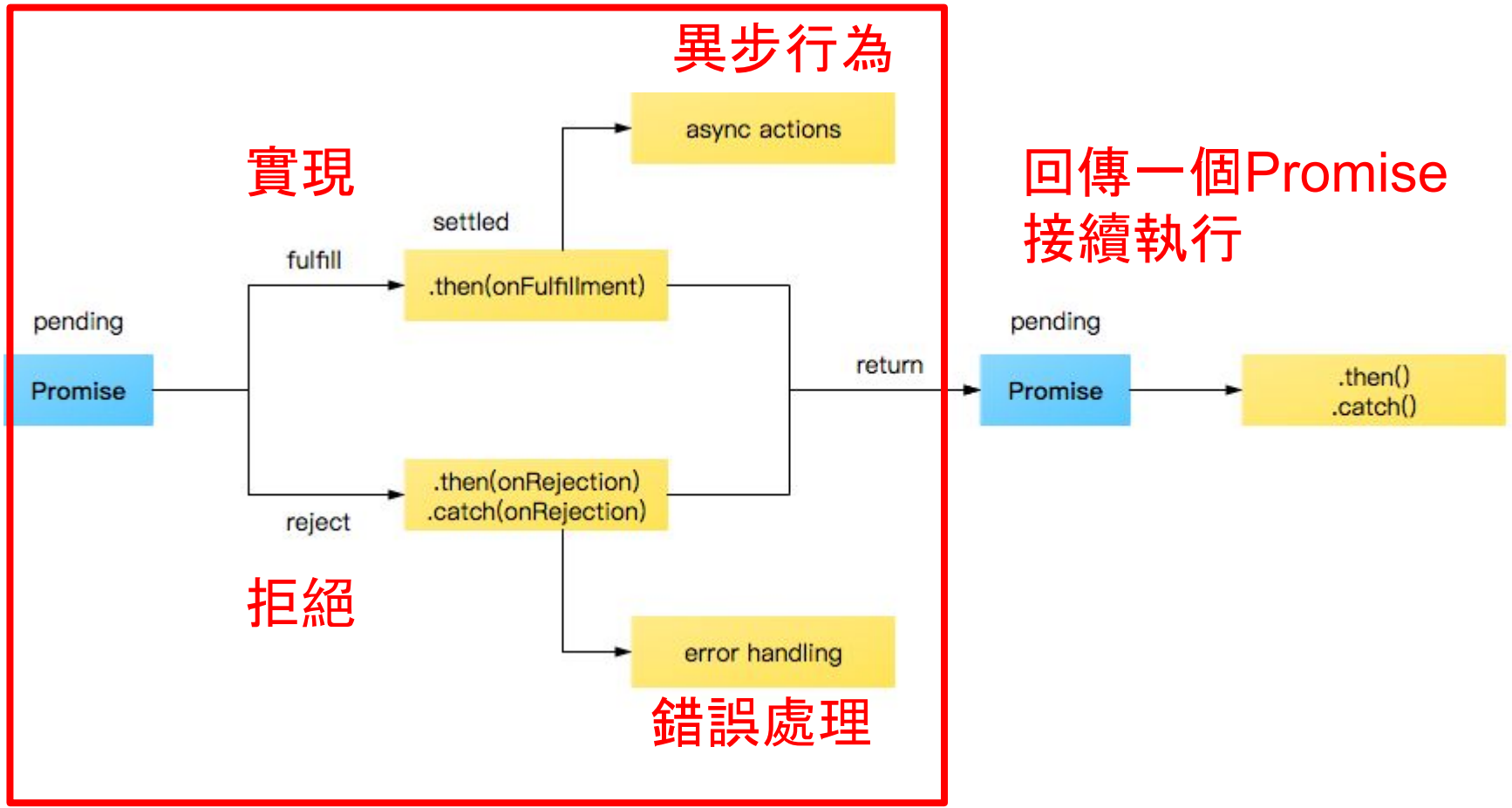
portability(移植性)：委派函式內只有完成、失敗，可以將一個Promise移植到另一個Promise上。

chaining(鏈)： .then()使有順序的任務更容易執行，避免巢狀。

straightforward(直接)： .then()使程式碼更容易閱讀。

用法簡介

Promise運行流程



用法簡介:

程式碼: http://codepad.org/D7k5JTTz

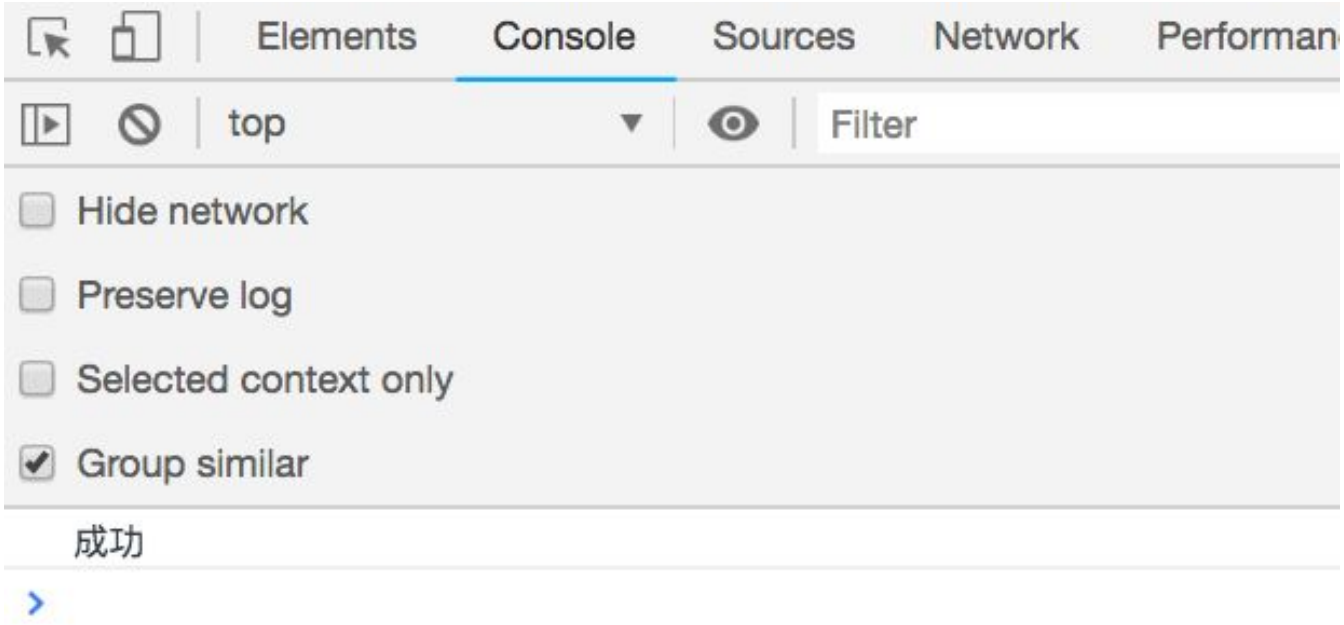
```
1   <script>
2
3     new Promise((resolve, reject) => {
4       if (1 === 1) {
5         console.log('成功')
6         resolve('成功')
7       } else {
8         console.log('失敗')
9         reject('失敗')
10      }
11    })
12
13  </script>
```

解決、拒絕方法

被解決

被拒絕

結果:


Elements
Console
Sources
Network
Performan
top
Filter
Hide network
Preserve log
Selected context only
Group similar
成功

串接下一個Promise

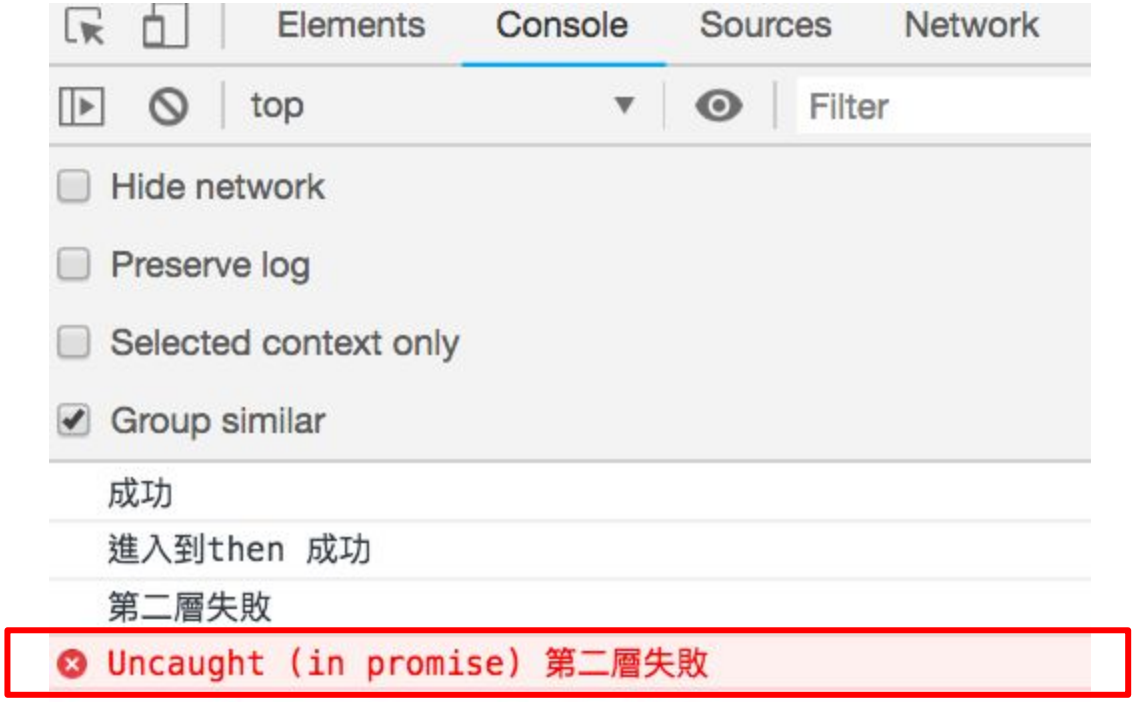
程式碼:http://codepad.org/B2VKMCwY

```
<script>

  new Promise((resolve, reject) => {
    if (1 === 1) {
      console.log('成功')
      resolve('成功')
    } else {
      console.log('失敗')
      reject('失敗')
    }
  }).then(res => {
    console.log("進入到then", res)
    return new Promise((resolve, reject) => {
      if (2 === 1) {
        console.log('第二層成功')
        resolve('第二層成功')
      } else {
        console.log('第二層失敗')
        reject('第二層失敗')
      }
    })
  })

</script>
```

1.用then串接

2.做一個Promise回傳

結果：


Elements
Console
Sources
Network
top
Filter
Hide network
Preserve log
Selected context only
Group similar
成功
進入到then 成功
第二層失敗
Uncaught (in promise) 第二層失敗

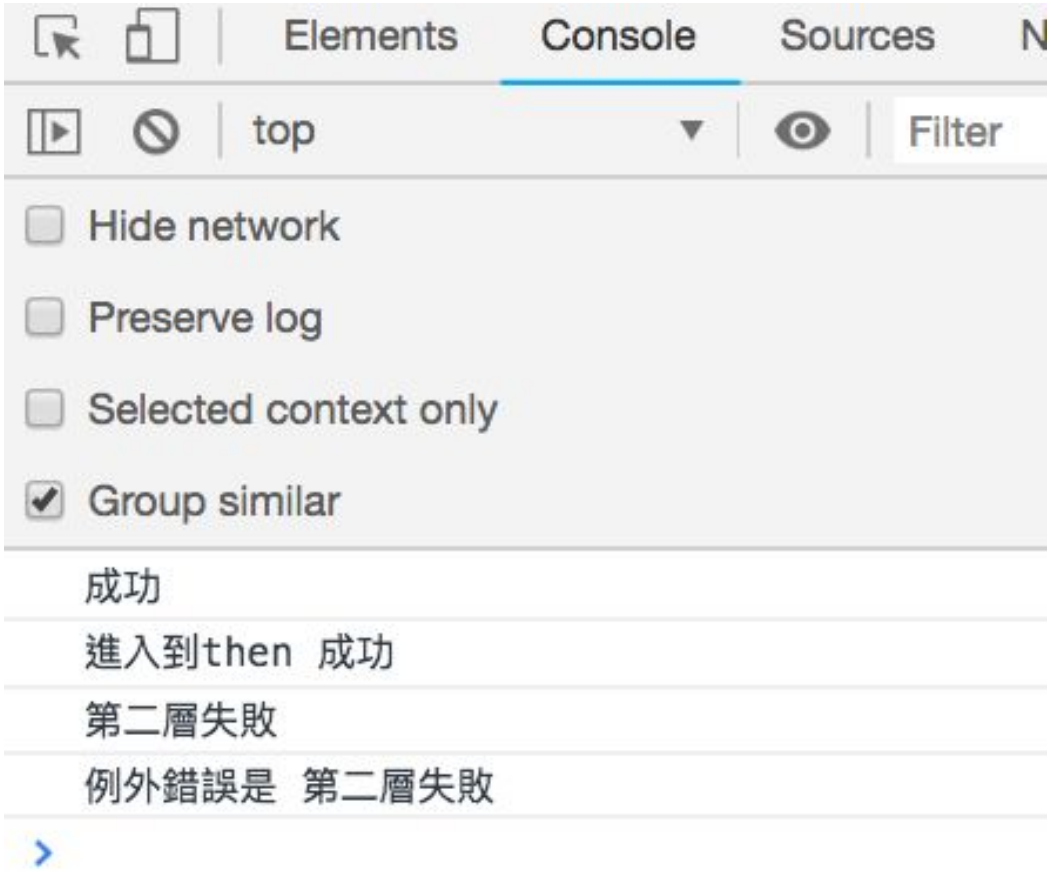
有例外但未捕捉

```
<script>

  new Promise((resolve, reject) => {
    if (1 === 1) {
      console.log('成功')
      resolve('成功')
    } else {
      console.log('失敗')
      reject('失敗')
    }
  }).then(res => {
    console.log("進入到then", res)
    return new Promise((resolve, reject) => {
      if (2 === 1) {
        console.log('第二層成功')
        resolve('第二層成功')
      } else {
        console.log('第二層失敗')
        reject('第二層失敗')
      }
    })
  }).catch(err => {
    console.log('例外錯誤是', err)
  })

</script>
```

加上catch

程式碼 http://codepad.org/bvJvSDUj

結果


Elements
Console
Sources
N
top
Filter
Hide network
Preserve log
Selected context only
Group similar
成功
進入到then 成功
第二層失敗
例外錯誤是 第二層失敗

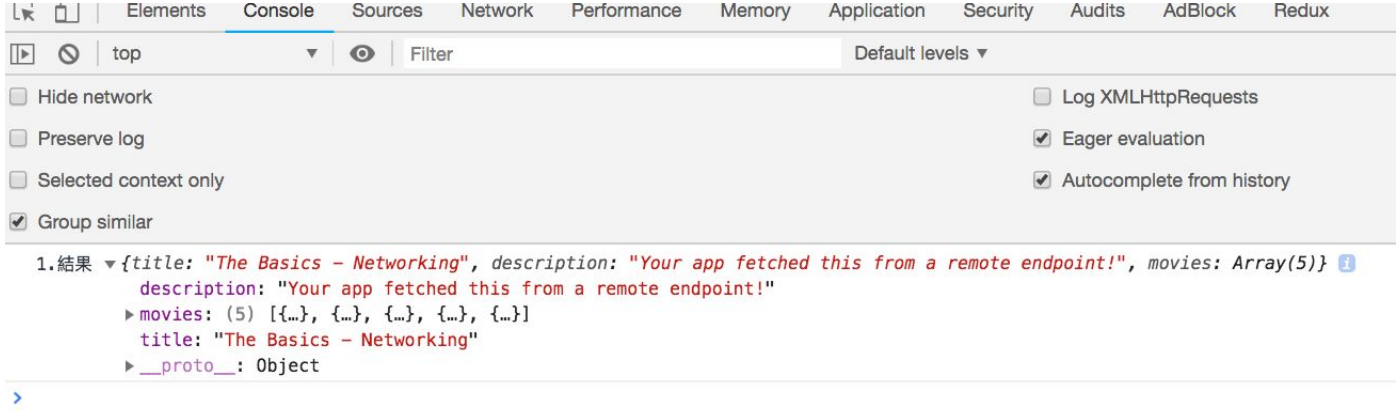
現實中遇到的Promise

fetch

fetch請求

程式碼:http://codepad.org/f8SCJen3

```
1  <script>
2
3    fetch('https://facebook.github.io/react-native/movies.json')
4    .then(res=>res.json())
5    .then(res=>{
6      console.log('1.結果',res)
7    })
8
9  </script>
```



利用鏈的方式串接下個請求繼續執行

程式碼:http://codepad.org/nErEdozK

```
<script>

  fetch('https://facebook.github.io/react-native/movies.json')
    .then(res => res.json())
    .then(res => {
      console.log('1.結果', res)
      return fetch('https://facebook.github.io/react-native/movies.json')
    })
    .then(res => res.json())
    .then(res => {
      console.log('2.結果', res)
    })

</script>
```

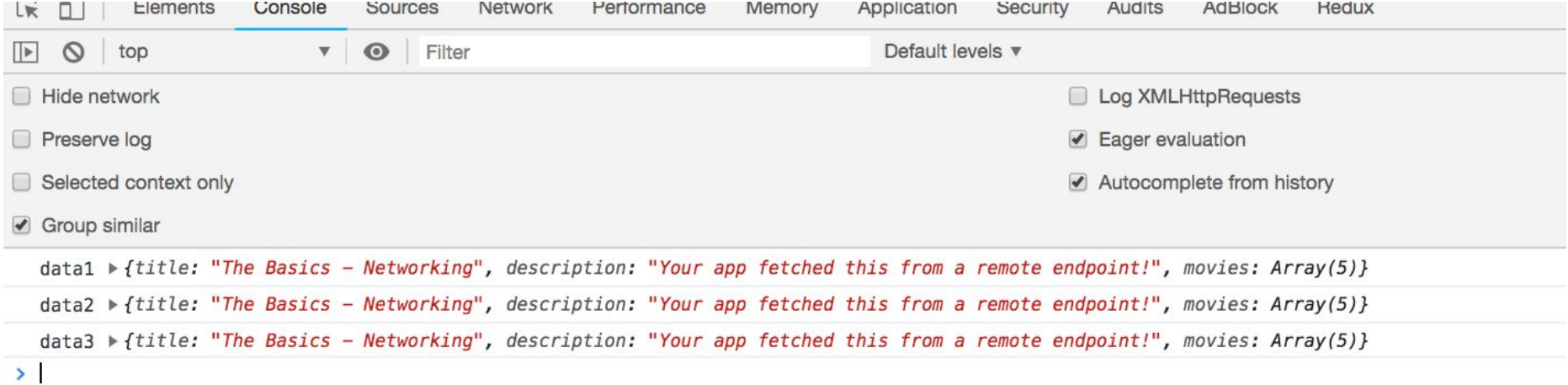
```
Group similar
1.結果 ▶{title: "The Basics - Networking", description: "Your app fetched this from a remote endpoint!", movies: Array(5)}
2.結果 ▶{title: "The Basics - Networking", description: "Your app fetched this from a remote endpoint!", movies: Array(5)}
>
```

Promise.all 一次等待資料都呼叫完再繼續

程式碼:http://codepad.org/onWNayLI

```
<script>

  Promise.all([
    fetch('https://facebook.github.io/react-native/movies.json'),
    fetch('https://facebook.github.io/react-native/movies.json'),
    fetch('https://facebook.github.io/react-native/movies.json')
  ]).then(res => Promise.all(res.map(res => res.json())))
    .then(([data1, data2,data3]) => {
      console.log('data1', data1)
      console.log('data2', data2)
      console.log('data3', data2)
    })
    .catch(e => console.log(e))

</script>
```

結果


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
AdBlock
Redux
top
Filter
Default levels
Hide network
Preserve log
Selected context only
Group similar
Log XMLHttpRequests
Eager evaluation
Autocomplete from history
data1 {title: "The Basics - Networking", description: "Your app fetched this from a remote endpoint!", movies: Array(5)}
data2 {title: "The Basics - Networking", description: "Your app fetched this from a remote endpoint!", movies: Array(5)}
data3 {title: "The Basics - Networking", description: "Your app fetched this from a remote endpoint!", movies: Array(5)}

除了ES6的 Promise，異步處理還可以利用
ES7的 Async/Await 達到！

END