链滴

关于 Promise.all, map, reduce, mapSeries, each 方法

作者：XPPA

原文链接：https://ld246.com/article/1538126239274

来源网站：链滴

原文 https://www.mgenware.com/blog/?p=2980

前言

之前在写nodejs有按顺序执行的需求，在网上找了很多的资料，最后发现上文中的帖子，后来发现原无法访问了，就自己在整理一篇,留作备用。

首先加入bluebird的 Promise 对象，然后写一个辅助方法 makePromise 用来生成 Promise ，下面的 data 变量是演示用的数据，代表 setTimeout 的延时。

```
const Promise = require('bluebird');

function makePromise(name, delay) {
    return new Promise((resolve) => {
        console.log(`${name} started`);
        setTimeout(() => {
            console.log(`${name} completed`);
            resolve(name);
        }, delay);
    });
}

const data = [2000, 1, 1000];
```

OK，首先看 Promise.all 方法，很简单，他会等所有 Promise 执行完毕后，把结果放在数组里，注这里 Promise 的执行不存在什么顺序还是并发问题，因为 Promise 本身就代表一个已经执行的过程所以 Promise.all 就是在等所有过程结束，代码：

```
Promise.all(data.map((v, k) => makePromise(k, v))).then(res => {
    console.log(res);
});
```

输出：

```
0 started
1 started
2 started
1 completed
2 completed
0 completed
[ 0, 1, 2 ]
```

然后是 Promise.map 方法，这个相当于把常见的 Array.map 创建 Promise 的过程和 Promise.all 合起来，所以用 Promise.map 来实现上例的代码是：

```
Promise.map(data, (v, k) => makePromise(k, v)).then(res => {
    console.log(res)
});
```

输出：

```
0 started
1 started
2 started
```

```
1 completed
2 completed
0 completed
[ 0, 1, 2 ]
```

输出和上面的 Promise.all 例子一样。

另外， Promise.map 还可以加入一个 concurrency 参数，注意这个 concurrency 参数是控制同时发创建的 Promise 个数，并且是不保证顺序的，所以当 {concurrency: 1} 时，只会有一个 Promise 运行，但是整个数组不保证是从左到右顺序执行的。

Promise.reduce 方法才是一个顺序执行 Promise 的方法，所谓顺序执行，其实就是从左到右按顺序创建 Promise ，并且始终只有一个 Promise 在运行。看bluebird的 源码 ，可以发现，一些顺序执的方法，比如 Promise.mapSeries 和 Promise.each ，都是基于 Promise.reduce 来实现的，这里的educe 和 Array.reduce 一样，测试代码：

```
Promise.reduce(data, (total, v, k) => {
    return makePromise(k, v).then(res => {
        return total + res;
    });
}, 0).then(res => {
    console.log(res);
})
```

输出:

```
0 started
0 completed
1 started
1 completed
2 started
2 completed
3
```

最后的 3 是所有 Promise 的结果总和，即 0+1+2 .

接下来是 Promise.mapSeries 方法，这个方法和 Promise.map 类似，只不过是顺序执行的，代码：

```
Promise.mapSeries(data, (v, k) => makePromise(k, v), 0).then(res => {
    console.log(res);
})
```

输出：

```
0 started
0 completed
1 started
1 completed
2 started
2 completed
[ 0, 1, 2 ]
```

最后是 Promise.each 方法，和 Promise.mapSeries 方法类似，都是顺序执行，区别是，返回的数不是所有 Promise 的结果，而是传入 Promise.each 方法的原始数据数组，如下代码：

```
Promise.each(data, (item, index) => makePromise(index, item), 0).then(res => {
    console.log(res);
});
```

输出:

```
0 started
0 completed
1 started
1 completed
2 started
2 completed
[ 0, 1, 2 ]
```