

Html 九九乘法表

```

index.html M JS 99.js M X # table.css {} launch.json
JS 99.js > ...
1 var HTML = "<table border=1 width=100%>";
2 for (j=1;j<10;j++){
3     HTML += "<tr>";
4     for (i=1;i<10;i++){
5         HTML += "<td align=center>" + i*j + "</td>";
6     }
7     HTML += "</tr>";
8 }
9 HTML += "</table>";
10 document.getElementById("list").innerHTML = HTML;

```

這是寫出九九乘法表的程式

在javascript中，本程式透過將要寫入html的程式碼以字串的形式儲存，做出九九乘法表的表格

1.令HTML為一變數，將表格的html程式碼字串先存入

2.用雙重for迴圈逐次寫入1~9行，並在各個行中逐次寫入1~9列

3.在每列中放入第i個行*第j個列的數字

4.使用getElementById找到html中 <div>的id，配合innerHTML，使<div>內的內容替換成HTML中存的字串

5.最後用<script src=>將同資料夾的javascript引入html程式中

6.在html接受到javascript的字串後，會自動將裡面的程式碼編譯成html的形式

```

index.html M X JS 99.js M # table.css {} launch.json
index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>9*9乘法表</title>
7     <link rel="stylesheet" type="text/css" href="table.css">
8 </head>
9 <body>
10     <p>九九乘法表</p>
11     <div id="list"></div>
12     <script src="99.js"></script>
13 </body>
14 </html>

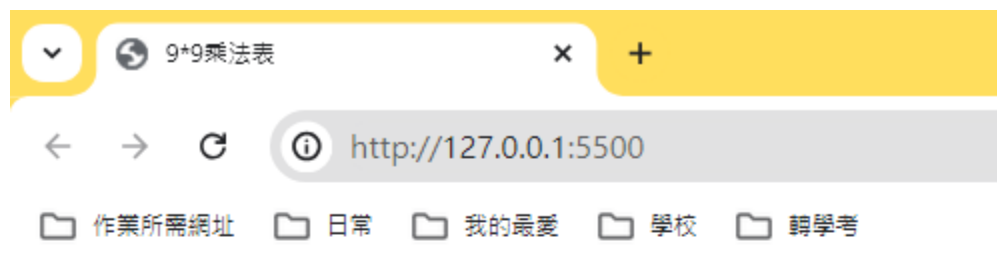
```

<div>中的替換的html編碼



```
<table border=1 width=100%><tr><td align=center>1</td><td align=center>2</td><td align=center>3</td><td align=center>4</td><td align=center>5</td><td align=center>6</td><td align=center>7</td><td align=center>8</td><td align=center>9</td></tr><tr><td align=center>2</td><td align=center>4</td><td align=center>6</td><td align=center>8</td><td align=center>10</td><td align=center>12</td><td align=center>14</td><td align=center>16</td><td align=center>18</td></tr>.....<tr><td align=center>9</td><td align=center>18</td><td align=center>27</td><td align=center>36</td><td align=center>45</td><td align=center>54</td><td align=center>63</td><td align=center>72</td><td align=center>81</td></tr></table>
```

這是表格的結果



九九乘法表

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

完整程式列表

```
<> index.html M JS 99.js M X # table.css {} launch.json

JS 99.js > ...
1  var HTML = "<table border=1 width=100%>";
2  for (j=1;j<10;j++){
3      HTML += "<tr>";
4      for (i=1;i<10;i++){
5          HTML += "<td align=center>" + i*j + "</td>";
6      }
7      HTML += "</tr>";
8  }
9  HTML += "</table>";
10 document.getElementById("list").innerHTML = HTML;

<> index.html M X JS 99.js M # table.css {} launch.json

<> index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>9*9乘法表</title>
7      <link rel="stylesheet" type="text/css" href="table.css">
8  </head>
9  <body>
10     <p>九九乘法表</p>
11     <div id="list"></div>
12     <script src="99.js"></script>
13 </body>
14 </html>
```

Javascript 快速排序法+二分搜尋法

```

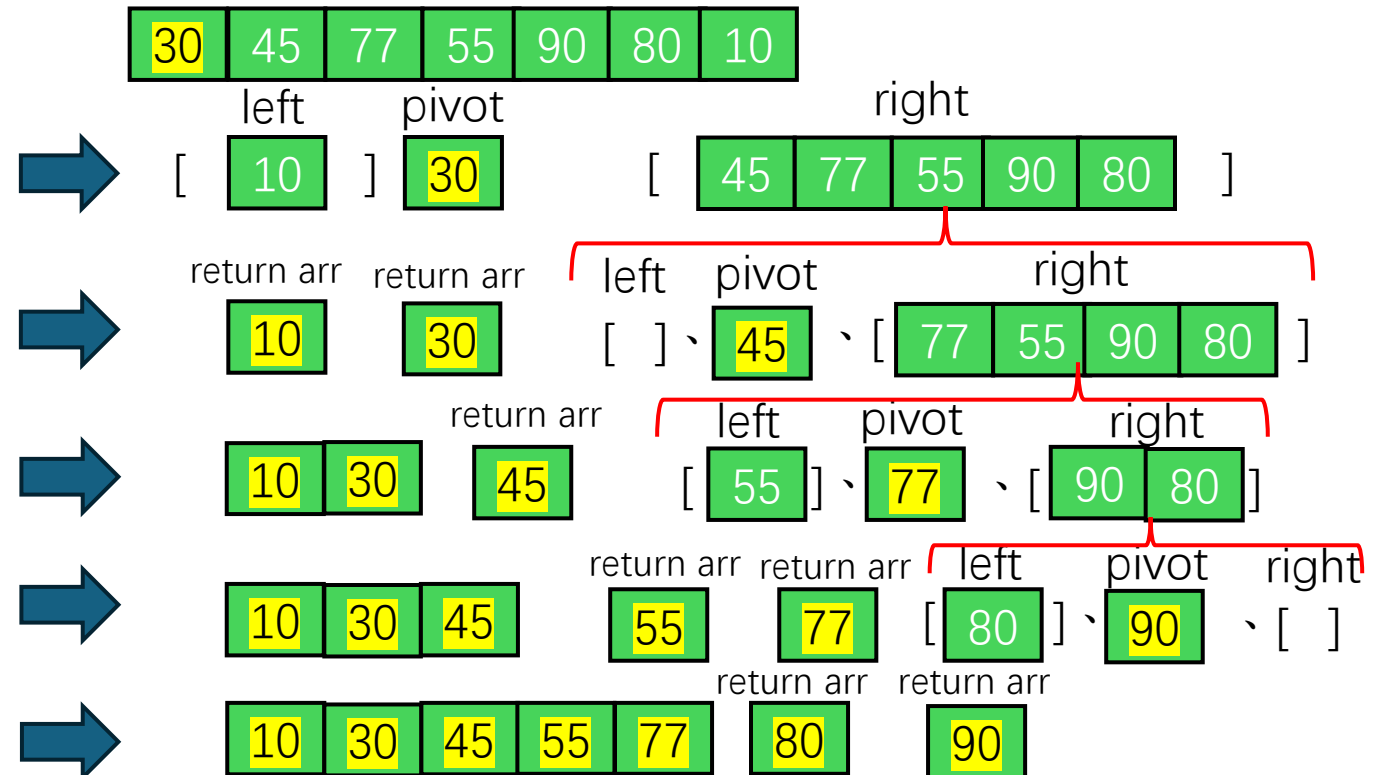
JS quicksort.js > prompt.get() callback
1  var prompt = require('prompt');
2
3  prompt.start();
4
5  let data = [30,45,77,55,90,80,10];
6
7  function quicksort(arr) {
8    if (arr.length <= 1) {
9      return arr;
10   }
11   let left = [];
12   let right = [];
13   let pivot = arr[0];
14   for (i = 1; i < arr.length; i++) {
15     let num = arr[i];
16     if (num < pivot){
17       left.push(num);
18     } else {
19       right.push(num);
20     }
21   }
22   return [...quicksort(left), pivot, ...quicksort(right)];
23 }
24 console.log(quicksort(data));
25
26 function binarysearch(arr,goal) {
27   let head = 0;
28   let end = arr.length - 1;
29   let mid;
30   while (head <= end) {
31     mid = ((head + end) / 2) | 0;
32     if (goal < arr[mid]) {
33       end = mid - 1;
34     } else if (goal > arr[mid]) {
35       head = mid + 1;
36     } else {
37       return ("搜尋選項在第" + (mid + 1) + "項");
38     }
39   }
40   return("無搜尋資料");
41 }
42
43 prompt.get(['number'],function(err,result){
44   console.log(binarysearch(quicksort(data),result.number));
45 });

```

這是快速排序法+二分搜尋法的程式

快速排序法的程式是

- 1.先給予一資料陣列 data
- 2.在function quicksort中，本程式使用的是選定data第一個數字做為基準值，宣告兩個空陣列"左"和"右"
- 3.將接下來的各個數字小於基準值的放進左陣列，大於的則放進右陣列
- 4.之後用遞迴的方式使得左陣列、基準值、右陣列個別再跑一次function，前面的if有判斷陣列資料剩下小於等於1個數字時會直接回傳
- 5.在多次循環下數字就會由小到大排序傳回data中

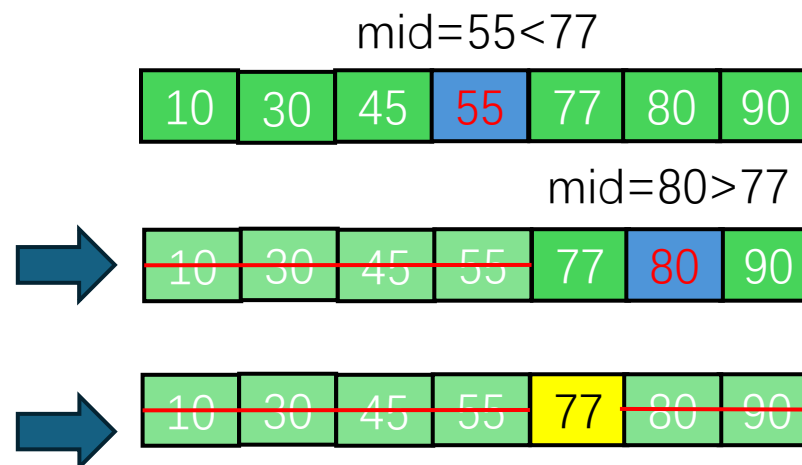


```
JS quicksort.js X
JS quicksort.js > prompt.get() callback
1 var prompt = require('prompt');
2
3 prompt.start();
4
5 let data = [30,45,77,55,90,80,10];
6
7 function quicksort(arr) {
8   if (arr.length <= 1) {
9     return arr;
10  }
11  let left = [];
12  let right = [];
13  let pivot = arr[0];
14  for (i = 1; i < arr.length; i++) {
15    let num = arr[i];
16    if (num < pivot){
17      left.push(num);
18    } else {
19      right.push(num);
20    }
21  }
22  return [...quicksort(left), pivot, ...quicksort(right)];
23 }
24 console.log(quicksort(data));
25
26 function binarysearch(arr,goal) {
27   let head = 0;
28   let end = arr.length - 1;
29   let mid;
30   while (head <= end) {
31     mid = ((head + end) / 2) | 0;
32     if (goal < arr[mid]) {
33       end = mid - 1;
34     } else if (goal > arr[mid]) {
35       head = mid + 1;
36     } else {
37       return ("搜尋選項在第" + (mid + 1) + "項");
38     }
39   }
40   return("無搜尋資料");
41 }
42
43 prompt.get(['number'],function(err,result){
44   console.log(binarysearch(quicksort(data),result.number));
45 });
```

二分搜尋法的程式是

- 1.輸入一個陣列與要在陣列中尋找的數字
- 2.數字用prompt javascript的寫法使得能輸入任意數字
- 3.在 function binarysearch 中，先宣告head=0、end=陣列長度-1和mid
- 4.本程式使用位元運算子的or方法配上0，使得(head+end)/2在存在小數點時，能捨去小數點後的數字變為整數
- 5.當mid位置的數字大於尋求的數字時，會使搜尋範圍從尾部往前縮至一半
- 6.反之，當mid位置的數字小於尋求的數字時，會使搜尋範圍從部頭往前縮至一半
- 7.反覆進行直到搜尋範圍縮至1個數字時，搜尋資料就在mid+1項的位置
- 8.若無此資料，最後就會回"無搜尋資料"

以10,30,45,55,80,77,90 搜尋77為例



這是執行結果



問題 輸出 偵錯主控台 終端機 連接埠

```
PS C:\Users\theo\Desktop\js\quicksort search> node quicksort.js  
[  
  10, 30, 45, 55,  
  77, 80, 90  
]  
prompt: number: 77  
搜尋選項在第5項  
PS C:\Users\theo\Desktop\js\quicksort search> |
```

完整程式列表

```
1  var prompt = require('prompt');
2
3  prompt.start();
4
5  let data = [30,45,77,55,90,80,10];
6
7  function quicksort(arr) {
8    if (arr.length <= 1) {
9      return arr;
10   }
11   let left = [];
12   let right = [];
13   let pivot = arr[0];
14   for (i = 1; i < arr.length; i++) {
15     let num = arr[i];
16     if (num < pivot){
17       left.push(num);
18     } else {
19       right.push(num);
20     }
21   }
22   return [...quicksort(left), pivot, ...quicksort(right)];
23 }
24 console.log(quicksort(data));
25
26 function binarysearch(arr,goal) {
27   let head = 0;
28   let end = arr.length - 1;
29   let mid;
30   while (head <= end) {
31     mid = ((head + end) / 2) | 0;
```

```
32     if (goal < arr[mid]) {
33         end = mid - 1;
34     } else if (goal > arr[mid]) {
35         head = mid + 1;
36     } else {
37         return ("搜尋選項在第" + (mid + 1) + "項");
38     }
39 }
40 return("無搜尋資料");
41 }
42
43 prompt.get(['number'],function(err,result){
44     console.log(binarysearch(quickSort(data),result.number));
45 });
```

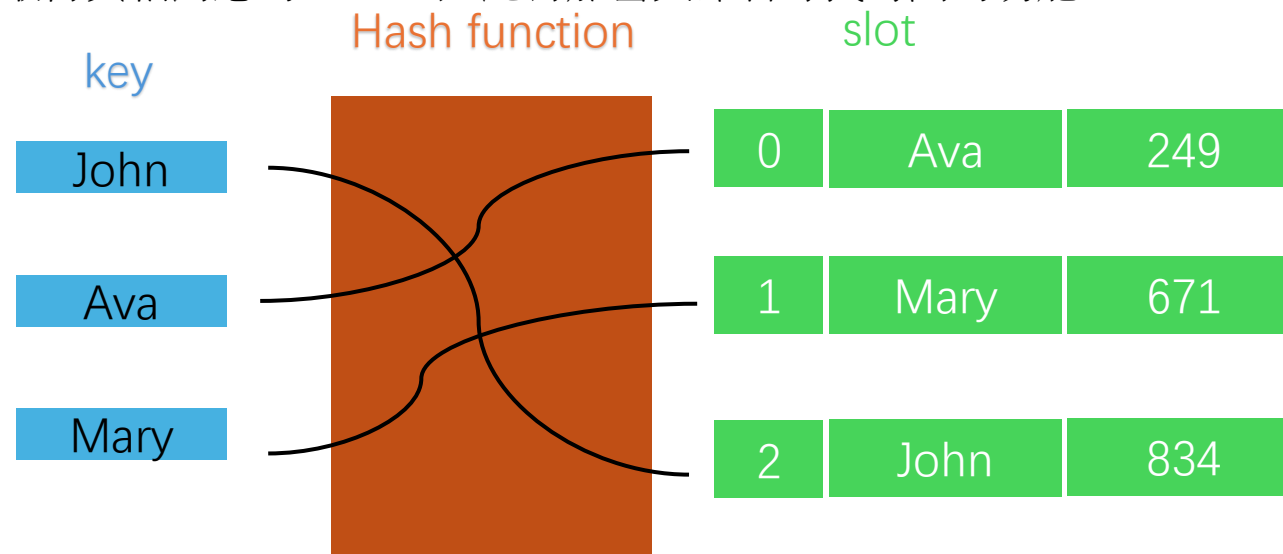
Javascript 雜湊表(HashTable)

JS hashtable.js > HashTable > search > slot

```
1 function hash(key, size) {
2   let hashCode = 0;
3   for (let index in key) {
4     hashCode += key.charCodeAt(index);
5   }
6   return hashCode % size;
7 }
8
9 class Box{
10   constructor(key,value){
11     this.key=key;
12     this.value=value;
13   }
14 }
15
16 class HashTable {
17   constructor() {
18     this.size = 16;
19     this.slots = new Array(this.size).fill(null);
20     this.length = 0;
21   }
22 }
```

這個程式是雜湊表(HashTable)

概念是透過雜湊函數計算出key所對應的一個slot，將該組key與value存入該slot之中，進而建立雜湊表格，如果不同的key找到相同對應的slot時，則在該slot內依序存進該slot中的陣列裡面。當要尋找資料時，key能快速找到其相對位置的slot，如果不同的key找到相同對應的slot時，則在該slot內的陣列中線性搜尋該相同的key，並取得其相對應的value，如此有加密與節省尋找時間的功能。



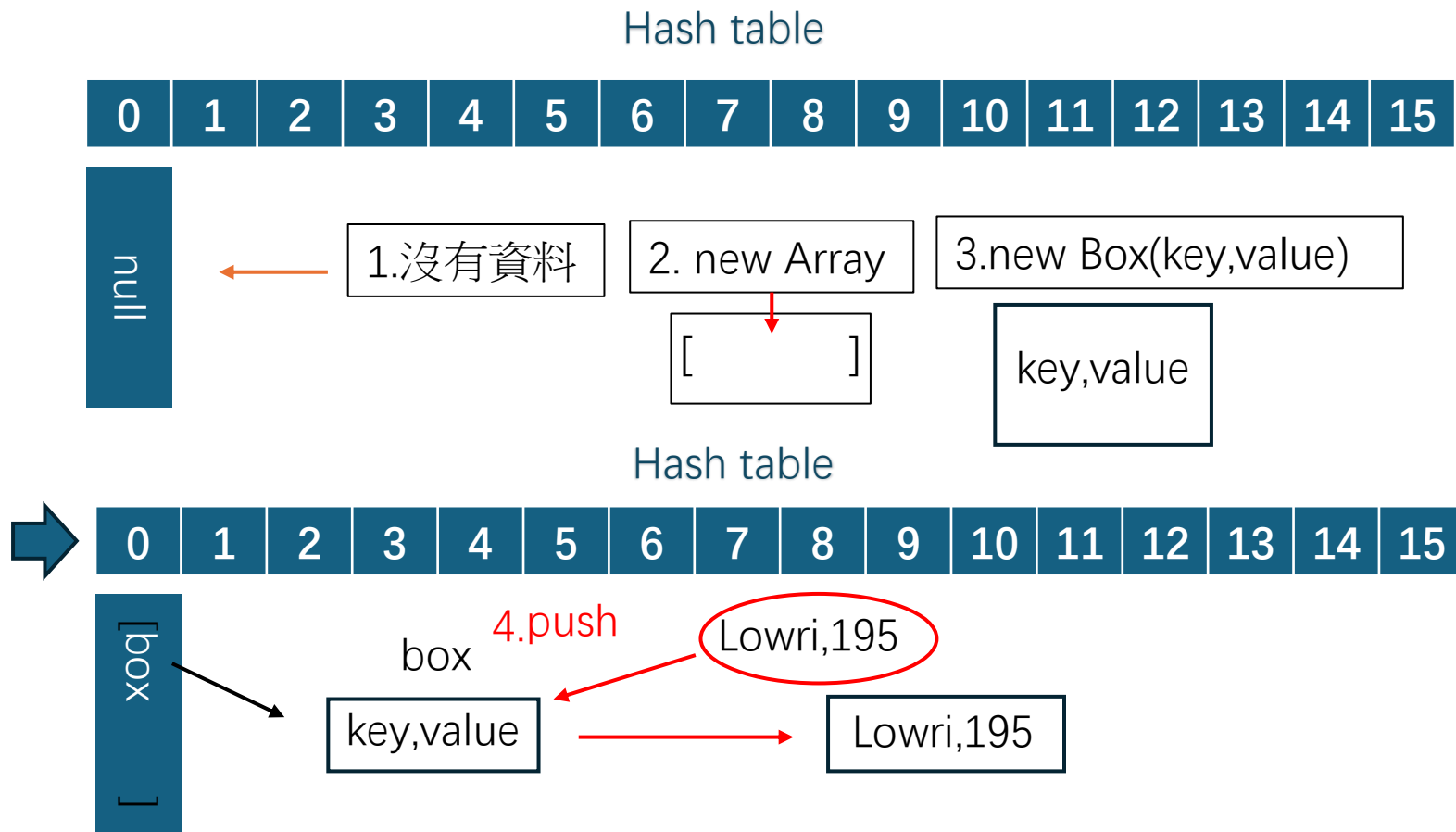
本程式使用的雜湊函數是以charCodeAt()，將key的字串先換成整數後除以slots的size取餘數來決定key與value放的slot位置，在本程式中的雜湊表size是16，所以就是餘16。本程式用class 定義box 來存放key與value，定義HashTable 有16個slots預先填入null，並用length存入資料的數量。

```

22 add(key,value){
23   const hashIndex = hash(key, this.size);
24   const slot = this.slots[hashIndex];
25   if (!slot){
26     this.slots[hashIndex] = new Array();
27     let box = new Box(key,value);
28     this.slots[hashIndex].push(box);
29     this.length++;
30   } else {
31     let found=false;
32     for(let i=0;i<slot.length;i++){
33       let box = slot[i];
34       if(box.key==key){
35         box.value=value;
36         found=true;
37         break;
38       }
39     }
40     if(!found){
41       let box = new Box(key,value);
42       this.slots[hashIndex].push(box);
43       this.length++;
44     }
45   }
46 }

```

add(key,value)是將key與value存入slot的方法，會先用if確認該slot內是否已經有陣列可以存放box，如果沒有就先創建一個陣列，再創建1個box放入陣列中，然後將box用push填入box陣列中，



```

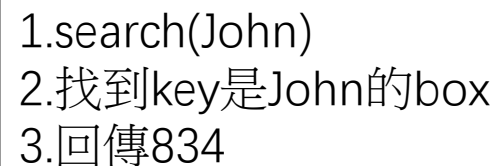
22 add(key,value){
23   const hashIndex = hash(key, this.size);
24   const slot = this.slots[hashIndex];
25   if (!slot){
26     this.slots[hashIndex] = new Array();
27     let box = new Box(key,value);
28     this.slots[hashIndex].push(box);
29     this.length++;
30   } else {
31     let found=false;
32     for(let i=0;i<slot.length;i++){
33       let box = slot[i];
34       if(box.key==key){
35         box.value=value;
36         found=true;
37         break;
38       }
39     }
40     if(!found){
41       let box = new Box(key,value);
42       this.slots[hashIndex].push(box);
43       this.length++;
44     }
45   }
46 }

```

如果slot中已經有box陣列的情況下，如果有相同key不同value的資料加入時，本程式會用新資料替換掉舊資料，用一個found變數當flag，預設為false，然後用for迴圈在陣列中線性尋找是否有相同的key的box。如果有找到相同key的box，則將新的value替換進該box中，將found設為true，然後離開迴圈。這樣就可以將舊value換成新value了。而如果没有找到相同的key，則該key為新的key，則創建1個新的box放入box陣列中，然後將資料用push填入box陣列中。

唯有替換資料的部分沒有放入`this.length++`，因為資料總數不變。

search(key)是用key找到已存入資料的方法，透過hash函數找到slot，再利用for迴圈在box陣列內線性搜尋是否有相同的key的box，若有相同，則回傳該box中的value，否則回傳"查無此資料!"。



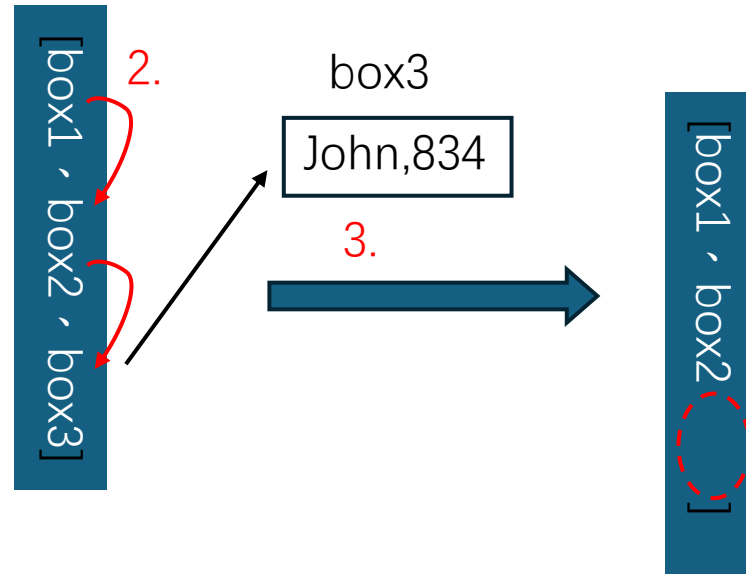

```

58 remove(key){
59     const hashIndex = hash(key, this.size);
60     const slot = this.slots[hashIndex];
61     for(let i=0;i<slot.length;i++){
62         let box = slot[i];
63         if(box.key==key){
64             slot.splice(i,1);
65             this.length--;
66         }
67     }
68     return "資料已清除!";
69 }
70 }
71
72 const ht = new HashTable();
73
74 ht.add("Louis", "767");
75 ht.add("Lowri", "195");
76 ht.add("Michelle", "162");
77 ht.add("Julie", "934");
78 ht.add("John", "329");
79 ht.add("Theresa", "331");
80 ht.add("John", "834");
81 ht.add("Eddie", "378");
82 ht.add("Robin", "77");
83 ht.add("Alfie", "532");
84 ht.add("Ava", "249");
85 ht.add("Owen", "208");
86 ht.add("Theodore", "539");
87 ht.add("Savannah", "54");

```

remove(key)是用來移除資料的方法，同樣透過透過hash函數找到slot，再利用for迴圈在box陣列內線性搜尋是否有相同的key的box。如果有，就使用splice將該位置的box移除。最後回報"資料已清除!"。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----



- 1.remove(John)
- 2.在box陣列中找到key是John的box
- 3.移除box

```
88 ht.add("Marc", "938");
89 ht.add("Sara", "420");
90 ht.add("Carmen", "599");
91 ht.add("Christine", "415");
92 ht.add("Mary", "671");
93 ht.add("Melissa", "171");
94 ht.add("Gethin", "483");
95 ht.add("Vanessa", "396");
96 ht.add("Eden", "638");
97 ht.add("Zoya", "72");
98 ht.add("Mae", "23");
99 ht.add("Edith", "609");
100 ht.add("Elizabeth", "660");
101 ht.add("Abdul", "619");
102 ht.add("Tabitha", "745");
103 ht.add("Jon", "957");
104 ht.add("Francis", "621");
105 ht.add("Millie", "767");
106
107 console.log(ht.search("John"));
108 console.log(ht.search("Ava"));
109 console.log(ht.remove("John"));
110 console.log(ht.search("John"));
```

這是實際程式表現出來的結果



問題	輸出	偵錯主控台	終端機	連接埠
			C:\Program Files\node.exe .\hashtable.js	
			834	
			249	
			資料已清除!	
			查無此資料!	

完整程式列表

```
1  function hash(key, size) {
2      let hashCode = 0;
3      for (let index in key) {
4          hashCode += key.charCodeAt(index);
5      }
6      return hashCode % size;
7  }
8
9  class Box{
10     constructor(key,value){
11         this.key=key;
12         this.value=value;
13     }
14 }
15
16 class HashTable {
17     constructor() {
18         this.size = 16;
19         this.slots = new Array(this.size).fill(null);
20         this.length = 0;
21     }
22     add(key,value){
23         const hashIndex = hash(key, this.size);
24         const slot = this.slots[hashIndex];
25         if (!slot){
26             this.slots[hashIndex] = new Array();
27             let box = new Box(key,value);
28             this.slots[hashIndex].push(box);
29             this.length++;
30         } else {
31             let found=false;
```

```

32         for(let i=0;i<slot.length;i++){
33             let box = slot[i];
34             if(box.key==key){
35                 box.value=value;
36                 found=true;
37                 break;
38             }
39         }
40         if(!found){
41             let box = new Box(key,value);
42             this.slots[hashIndex].push(box);
43             this.length++;
44         }
45     }
46 }
47 search(key){
48     const hashIndex = hash(key, this.size);
49     const slot = this.slots[hashIndex];
50     for(let i=0;i<slot.length;i++){
51         let box = slot[i];
52         if(box.key==key){
53             return box.value;
54         }
55     }
56     return "查無此資料!";
57 }
58 remove(key){
59     const hashIndex = hash(key, this.size);
60     const slot = this.slots[hashIndex];
61     for(let i=0;i<slot.length;i++){
62         let box = slot[i];
63         if(box.key==key){
64             slot.splice(i,1);
65             this.length--;
66         }

```

```
67     }  
68     return "資料已清除!";  
69 }  
70 }  
71  
72 const ht = new HashTable();  
73  
74 ht.add("Louis", "767");  
75 ht.add("Lowri", "195");  
76 ht.add("Michelle", "162");  
77 ht.add("Julie", "934");  
78 ht.add("John", "329");  
79 ht.add("Theresa", "331");  
80 ht.add("John", "834");  
81 ht.add("Eddie", "378");  
82 ht.add("Robin", "77");  
83 ht.add("Alfie", "532");  
84 ht.add("Ava", "249");  
85 ht.add("Owen", "208");  
86 ht.add("Theodore", "539");  
87 ht.add("Savannah", "54");  
88 ht.add("Marc", "938");  
89 ht.add("Sara", "420");  
90 ht.add("Carmen", "599");  
91 ht.add("Christine", "415");  
92 ht.add("Mary", "671");  
93 ht.add("Melissa", "171");  
94 ht.add("Gethin", "483");  
95 ht.add("Vanessa", "396");  
96 ht.add("Eden", "638");  
97 ht.add("Zoya", "72");  
98 ht.add("Mae", "23");  
99 ht.add("Edith", "609");  
100 ht.add("Elizabeth", "660");
```

```
101 ht.add("Abdul", "619");
102 ht.add("Tabitha", "745");
103 ht.add("Jon", "957");
104 ht.add("Francis", "621");
105 ht.add("Millie", "767");
106
107 console.log(ht.search("John"));
108 console.log(ht.search("Ava"));
109 console.log(ht.remove("John"));
110 console.log(ht.search("John"));|
```