

题目描述

Maven 版本号定义，<主版本>.<次版本>.<增量版本>-<里程碑版本>，举例3.1.4-beta

其中，主版本和次版本都是必须的，主版本，次版本，增量版本由多位数字组成，可能包含前导零，里程碑版本由字符串组成。

<主版本>.<次版本>.<增量版本>：基于数字比较；例如

1.5 > 1.4 > 1.3.11 > 1.3.9

里程碑版本：基于字符串比较，采用字典序；例如

1.2-beta-3 > 1.2-beta-11

比较版本号时，按从左到右的顺序依次比较。

基于数字比较，只需比较忽略任何前导零后的整数值。

输入2个版本号，输出最大版本号。

输入描述

输入2个版本号，换行分割，每个版本的最大长度小于50

输出描述

版本号相同时输出第一个输入版本号

用例

输入	2.5.1-C 1.4.2-D
输出	2.5.1-C
说明	无

输入	1.05.1 1.5.01
输出	1.05.1
说明	版本号相同，输出第一个版本号

输入	1.5 1.5.0
输出	1.5.0
说明	主次相同，存在增量版本大于不存在

输入	1.5.1-A 1.5.1-a
输出	1.5.1-a
说明	里程碑版本号，基于字典序比较，a 大于 A

题目解析

简单排序 题，考察数组排序

本题和[华为机试 - 比较两个版本号的大小_伏城之外的博客-CSDN博客](#)

类似，做完本题，可以再试试上面这个真题

2022.12.20 补充

这题看走眼了。

题目中说：“主版本和次版本都是必须的”，也就说后面几个版本可能是没有的，因此如下正则表达式：const

题目解析

简单排序^Q题，考察数组排序

本题和[华为机试 - 比较两个版本号的大小](#) - 伏城之外的博客 - CSDN博客

类似，做完本题，可以再试试上面这个真题

2022.12.20 补充

这题看走眼了。

题目中说：“主版本和次版本都是必须的”，也就说后面几个版本可能是没有的，因此如下正则表达式：const regExp = /[^](\d+)(\d+)(\d+)(-)?\$/;

获取缺失了增量版本和里程碑版本的字符串时，是存在问题的。

这里更新了下正则

const reg = /[^](\d+)(\d+)(\d+)?(-)?\$/;

在增量版本捕获组后面加了?，以及在里程碑版本后面加了?，表示这两个版本可以没有。

同时为了直到缺失的增量版本，还是里程碑版本，我们需要将"-"和“-”加入捕获组中，来帮助我们辨别。

最终效果如下：

```
> const reg = /^(\d+)(\d+)(\d+)?(-)?$/;
< undefined
> reg.exec('1.5')
< (7) ['1.5', '1', '.5', '5', undefined, undefined, undefined, index: 0, input: '1.5', groups: undefined]
> reg.exec('1.5.0')
< (7) ['1.5.0', '1', '.5', '5', '.0', '0', undefined, index: 0, input: '1.5.0', groups: undefined]
> reg.exec('1.5-a')
< (7) ['1.5-a', '1', '.5', '5', undefined, undefined, '-a', index: 0, input: '1.5-a', groups: undefined]
> |
CSDN @伏城之外
```

需要注意的是捕获组的顺序就是正则中括号从左到右的顺序，比如

```
> const reg = /^(\d+)(\d+)(\d+)?(-)?$/;
< undefined
> reg.exec('1.5.5-A')
< (7) ['1.5.5-A', '1', '.5', '5', '.5', '5', '-A', index: 0, input: '1.5.5-A', groups: undefined]
> |
CSDN @伏城之外
```

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12 });
13 if (lines.length === 2) {
14   console.log(getResult(lines));
15   lines.length = 0;
16 }
17 });
18
19 function getResult(versions) {
20   const reg = /^(\d+)(\d+)(\d+)?(-)?$/;
21   return versions.sort((v1, v2) => {
22     let [_1, major1, _11, minor1, _111, patch1, mile1] = reg.exec(v1);
23     let [_2, major2, _22, minor2, _222, patch2, mile2] = reg.exec(v2);
24
25     major1 = Number(major1);
26     major2 = Number(major2);
27
28     if (major1 !== major2) return major2 - major1;
29
30     minor1 = Number(minor1);
31     minor2 = Number(minor2);
32
33     if (minor1 !== minor2) return minor2 - minor1;
34
35     if (patch1 !== undefined && patch2 !== undefined) {
36       patch1 = Number(patch1);
37       patch2 = Number(patch2);
38       if (patch1 !== patch2) return patch2 - patch1;
39     } else if (patch1 !== undefined && !patch2) {
40       return -1;
41     } else if (patch2 !== undefined && !patch1) {
42       return 1;
43     }
44
45     if (mile1 && mile2) {
46       return mile1 === mile2 ? 0 : mile2 > mile1 ? 1 : -1;
47     } else if (mile1 && !mile2) {
48       return -1;
49     } else if (!mile1 && mile2) {
50       return 1;
51     }
52   });
53 }
```

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     console.log(getResult(lines));
15     lines.length = 0;
16   }
17 });
18
19 function getResult(versions) {
20   const reg = /^(\d+)(\.(\\d+))(\.(\\d+))?(\\-\\.+)?$/;
21   return versions.sort((v1, v2) => {
22     let [_1, major1, _11, minor1, _111, patch1, mile1] = reg.exec(v1);
23     let [_2, major2, _22, minor2, _222, patch2, mile2] = reg.exec(v2);
24
25     major1 = Number(major1);
26     major2 = Number(major2);
27
28     if (major1 !== major2) return major2 - major1;
29
30     minor1 = Number(minor1);
31     minor2 = Number(minor2);
32
33     if (minor1 !== minor2) return minor2 - minor1;
34
35     if (patch1 !== undefined && patch2 !== undefined) {
36       patch1 = Number(patch1);
37       patch2 = Number(patch2);
38       if (patch1 !== patch2) return patch2 - patch1;
39     } else if (patch1 !== undefined && !patch2) {
40       return -1;
41     } else if (patch2 !== undefined && !patch1) {
42       return 1;
43     }
44
45     if (mile1 && mile2) {
46       return mile1 === mile2 ? 0 : mile2 > mile1 ? 1 : -1;
47     } else if (mile1 && !mile2) {
48       return -1;
49     } else if (!mile1 && mile2) {
50       return 1;
51     } else {
52       return 0;
53     }
54   })[0];
55 }
```

Java算法源码

```
1  import java.util.*;
2  import java.util.regex.Matcher;
3  import java.util.regex.Pattern;
4
5  public class Main {
6    public static void main(String[] args) {
7      Scanner sc = new Scanner(System.in);
8
9      String str1 = sc.next();
10     String str2 = sc.next();
11
12     System.out.println(getResult(new String[]{str1, str2}));
13   }
14
15   public static String getResult(String[] versions) {
16     String reg = "^((\\d+)((\\.(\\d+))((\\.(\\d+))?(\\-\\.+)?))?)$";
17     Pattern p = Pattern.compile(reg);
18
19     Arrays.sort(versions, (v1, v2) -> {
20       Matcher m1 = p.matcher(v1);
21       Matcher m2 = p.matcher(v2);
22
23       if (m1.find() && m2.find()) {
24         Integer major1 = Integer.parseInt(m1.group(1));
25         Integer major2 = Integer.parseInt(m2.group(1));
26
27         if (major1 != major2) return major2 - major1;
28
29         Integer minor1 = Integer.parseInt(m1.group(3));
30         Integer minor2 = Integer.parseInt(m2.group(3));
31
32         if (minor1 != minor2) return minor2 - minor1;
```

Java算法源码

```
1 import java.util.*;
2 import java.util.regex.Matcher;
3 import java.util.regex.Pattern;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         String str1 = sc.next();
10        String str2 = sc.next();
11
12        System.out.println(getResult(new String[]{str1, str2}));
13    }
14
15    public static String getResult(String[] versions) {
16        String reg = "^((\\d+)(\\.(\\d+))?(\\.(\\d+))?(\\.-.+)?$)";
17        Pattern p = Pattern.compile(reg);
18
19        Arrays.sort(versions, (v1, v2) -> {
20            Matcher m1 = p.matcher(v1);
21            Matcher m2 = p.matcher(v2);
22
23            if (m1.find() && m2.find()) {
24                Integer major1 = Integer.parseInt(m1.group(1));
25                Integer major2 = Integer.parseInt(m2.group(1));
26
27                if (major1 != major2) return major2 - major1;
28
29                Integer minor1 = Integer.parseInt(m1.group(3));
30                Integer minor2 = Integer.parseInt(m2.group(3));
31
32                if (minor1 != minor2) return minor2 - minor1;
33
34                String patch1 = m1.group(5);
35                String patch2 = m2.group(5);
36
37                if (patch1 != null && patch2 != null) {
38                    int patch1_intVal = Integer.parseInt(patch1);
39                    int patch2_intVal = Integer.parseInt(patch2);
40                    if (patch1_intVal != patch2_intVal) {
41                        return Integer.parseInt(patch2) - Integer.parseInt(patch1);
42                    }
43                } else if (patch1 != null) {
44                    return -1;
45                } else if (patch2 != null) {
46                    return 1;
47                }
48
49                String mile1 = m1.group(6);
50                String mile2 = m2.group(6);
51
52                if (mile1 != null && mile2 != null) {
53                    return mile2.compareTo(mile1);
54                } else if (mile1 != null) {
55                    return -1;
56                } else if (mile2 != null) {
57                    return 1;
58                } else {
59                    return 0;
60                }
61            }
62
63            return 0;
64        });
65
66        return versions[0];
67    }
68 }
69
70 }
```

Python算法源码

```
1 import re
2
3 # 输入获取
4 v1 = input()
5 v2 = input()
6
7
8 # 算法入口
9 def getResult(v1, v2):
10     pattern = r"^((\d+)(?:(\.(\\d+))?(?:(\.(\\d+))?(?:(-|.+)?)?$)"
11
12     major1, minor1, patch1, mile1 = re.findall(pattern, v1)[0]
13     major2, minor2, patch2, mile2 = re.findall(pattern, v2)[0]
14
15     if major1 != major2:
16         if int(major1) > int(major2):
17             return v1
```

Python算法源码

```
1 import re
2
3 # 输入获取
4 v1 = input()
5 v2 = input()
6
7
8 # 算法入口
9 def getResult(v1, v2):
10     pattern = r"^(\d+)(?:\.\d+)?(?:\.\d+)?(?:\-(.+))?$"
11
12     major1, minor1, patch1, mile1 = re.findall(pattern, v1)[0]
13     major2, minor2, patch2, mile2 = re.findall(pattern, v2)[0]
14
15     if major1 != major2:
16         if int(major1) > int(major2):
17             return v1
18         else:
19             return v2
20
21     if int(minor1) != int(minor2):
22         if int(minor1) > int(minor2):
23             return v1
24         else:
25             return v2
26
27     if patch1 != patch2:
28         if patch1 != "" and patch2 != "":
29             if int(patch1) > int(patch2):
30                 return v1
31             elif int(patch1) < int(patch2):
32                 return v2
33         elif patch1 != "" and patch2 == "":
34             return v1
35         elif patch1 == "" and patch2 != "":
36             return v2
37
38     if mile1 != mile2:
39         if mile1 != "" and mile2 != "":
40             if mile1 > mile2:
41                 return v1
42             elif mile1 < mile2:
43                 return v2
44         elif mile1 != "" and mile2 == "":
45             return v1
46         elif mile1 == "" and mile2 != "":
47             return v2
48
49     return v1
50
51
52 # 算法调用
53 print(getResult(v1, v2))
```

[复制](#)