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

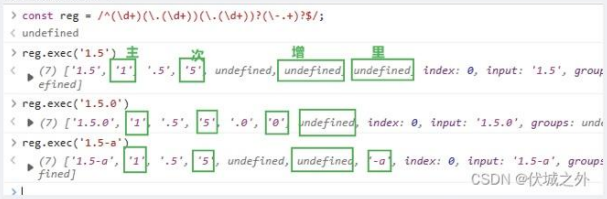
题目解析

简单排序题，考察数组排序

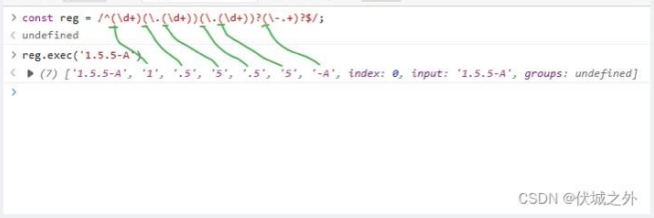
本题和华为机试 - 比较两个版本号的大小_伏城之外的博客-CSDN博客

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：



> 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



JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    console.log(getResult(lines));
    lines.length = 0;
  }
});

function getResult(versions) {
  const reg = /^(\d+)(\.(\d+))(\.(\d+))?(\-.+)?$/;
  return versions.sort((v1, v2) => {
    let [_1, major1, _11, minor1, _111, patch1, mile1] = reg.exec(v1);
    let [_2, major2, _22, minor2, _222, patch2, mile2] = reg.exec(v2);

    major1 = Number(major1);
    major2 = Number(major2);

    if (major1 !== major2) return major2 - major1;

    minor1 = Number(minor1);
    minor2 = Number(minor2);

    if (minor1 !== minor2) return minor2 - minor1;

    if (patch1 !== undefined && patch2 !== undefined) {
      patch1 = Number(patch1);
      patch2 = Number(patch2);
      if (patch1 !== patch2) return patch2 - patch1;
    } else if (patch1 !== undefined && !patch2) {
      return -1;
    } else if (patch2 !== undefined && !patch1) {
      return 1;
    }

    if (mile1 && mile2) {
      return mile1 === mile2 ? 0 : mile2 > mile1 ? 1 : -1;
    } else if (mile1 && !mile2) {
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    console.log(getResult(lines));
    lines.length = 0;
  }
});

function getResult(versions) {
  const reg = /^(\d+)(\.(\d+))(\.(\d+))?(\-.+)?$/;
  return versions.sort((v1, v2) => {
    let [_1, major1, _11, minor1, _111, patch1, mile1] = reg.exec(v1);
    let [_2, major2, _22, minor2, _222, patch2, mile2] = reg.exec(v2);

    major1 = Number(major1);
    major2 = Number(major2);

    if (major1 !== major2) return major2 - major1;

    minor1 = Number(minor1);
    minor2 = Number(minor2);

    if (minor1 !== minor2) return minor2 - minor1;

    if (patch1 !== undefined && patch2 !== undefined) {
      patch1 = Number(patch1);
      patch2 = Number(patch2);
      if (patch1 !== patch2) return patch2 - patch1;
    } else if (patch1 !== undefined && !patch2) {
      return -1;
    } else if (patch2 !== undefined && !patch1) {
      return 1;
    }

    if (mile1 && mile2) {
      return mile1 === mile2 ? 0 : mile2 > mile1 ? 1 : -1;
    } else if (mile1 && !mile2) {
      return -1;
    } else if (!mile1 && mile2) {
      return 1;
    } else {
      return 0;
    }
  })[0];
}
```

Java算法源码

```java
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String str1 = sc.next();
        String str2 = sc.next();

        System.out.println(getResult(new String[]{str1, str2}));
    }

    public static String getResult(String[] versions) {
        String reg = "^(\\d+)(\\.(\\d+))(\\.(\\d+))?(\\-.+)?$";
        Pattern p = Pattern.compile(reg);

        Arrays.sort(versions, (v1, v2) -> {
            Matcher m1 = p.matcher(v1);
            Matcher m2 = p.matcher(v2);

            if (m1.find() && m2.find()) {
                Integer major1 = Integer.parseInt(m1.group(1));
                Integer major2 = Integer.parseInt(m2.group(1));

                if(major1 != major2) return major2 - major1;

                Integer minor1 = Integer.parseInt(m1.group(3));
                Integer minor2 = Integer.parseInt(m2.group(3));

                if(minor1 != minor2) return minor2 - minor1;
```

Java算法源码

```java
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String str1 = sc.next();
        String str2 = sc.next();

        System.out.println(getResult(new String[]{str1, str2}));
    }

    public static String getResult(String[] versions) {
        String reg = "^(\\d+)(\\.(\\d+))(\\.(\\d+))?(\\-.+)?$";
        Pattern p = Pattern.compile(reg);

        Arrays.sort(versions, (v1, v2) -> {
            Matcher m1 = p.matcher(v1);
            Matcher m2 = p.matcher(v2);

            if (m1.find() && m2.find()) {
                Integer major1 = Integer.parseInt(m1.group(1));
                Integer major2 = Integer.parseInt(m2.group(1));

                if(major1 != major2) return major2 - major1;

                Integer minor1 = Integer.parseInt(m1.group(3));
                Integer minor2 = Integer.parseInt(m2.group(3));

                if(minor1 != minor2) return minor2 - minor1;

                String patch1 = m1.group(5);
                String patch2 = m2.group(5);

                if(patch1 != null && patch2 != null) {
                    int patch1_intVal = Integer.parseInt(patch1);
                    int patch2_intVal = Integer.parseInt(patch2);
                    if(patch1_intVal != patch2_intVal) {
                        return Integer.parseInt(patch2) - Integer.parseInt(patch1);
                    }
                } else if(patch1 != null) {
                    return -1;
                } else if(patch2 != null) {
                    return 1;
                }


                String mile1 = m1.group(6);
                String mile2 = m2.group(6);

                if(mile1 != null && mile2 != null) {
                    return mile2.compareTo(mile1);
                } else if(mile1 != null) {
                    return -1;
                } else if(mile2 != null) {
                    return 1;
                } else {
                    return 0;
                }
            }

            return 0;
        });

        return versions[0];
    }

}
```

Python算法源码

```python
import re

# 输入获取
v1 = input()
v2 = input()


# 算法入口
def getResult(v1, v2):
    pattern = r"^(\d+)(?:\.(\d+))(?:\.(\d+))?(?:\-(.+))?$"

    major1, minor1, patch1, mile1 = re.findall(pattern, v1)[0]
    major2, minor2, patch2, mile2 = re.findall(pattern, v2)[0]

    if major1 != major2:
        if int(major1) > int(major2):
            return v1
```

Python算法源码

```python
import re

# 输入获取
v1 = input()
v2 = input()


# 算法入口
def getResult(v1, v2):
    pattern = r"^(\d+)(?:\.(\d+))(?:\.(\d+))?(?:\-(.+))?$"

    major1, minor1, patch1, mile1 = re.findall(pattern, v1)[0]
    major2, minor2, patch2, mile2 = re.findall(pattern, v2)[0]

    if major1 != major2:
        if int(major1) > int(major2):
            return v1
        else:
            return v2

    if int(minor1) != int(minor2):
        if int(minor1) > int(minor2):
            return v1
        else:
            return v2

    if patch1 != patch2:
        if patch1 != "" and patch2 != "":
            if int(patch1) > int(patch2):
                return v1
            elif int(patch1) < int(patch2):
                return v2
        elif patch1 != "" and patch2 == "":
            return v1
        elif patch1 == "" and patch2 != "":
            return v2

    if mile1 != mile2:
        if mile1 != "" and mile2 != "":
            if mile1 > mile2:
                return v1
            elif mile1 < mile2:
                return v2
        elif mile1 != "" and mile2 == "":
            return v1
        elif mile1 == "" and mile2 != "":
            return v2

    return v1


# 算法调用
print(getResult(v1, v2))
```