北大-华为云 2019 级新工科实验班

个人打卡汇总文档

符豫

2019.10

目 录

Part.1 算法题 - 3 -
1.1 爬楼梯 - 3 -
1.2 逆波兰表达式 - 5 -
1.3 棒球比赛 - 7 -
1.4 检查替换后的词是否有效 - 9 -
1.5 简化路径 - 11 -
1.6 下一个更大元素 1 - 12 -
1.7 下一个更大元素 2 - 16 -
1.8 股票跨度 - 19 -
1.9 盛最多水的容器 - 21 -
1.10 加一 - 22 -
1.11 最大子序和 - 25 -
1.12 按奇偶排序数组 - 27 -
1.13 数组拆分 - 29 -
1.14 递增三元子序列 - 30 -
1.15 移动零 - 33 -

Part.2 安卓学习 - 35 -
2.1 安装 Android Studio - 35 -
下载 - 35 -
SDK - 35 -
gradle - 36 -
SDK Manager - 40 -
彻底解决无法导入库的问题 - 41 -
2.2 约束性布局 ConstraintLayout - 43 -
为什么选择它 - 43 -
依赖和 app 命名空间声明 - 44 -
布局思路 - 45 -
参考线 GuideLine - 45 -
设置约束 - 47 -
链式约束 - 48 -
2.3 安卓代码命名规范 - 48 -
2.4 编写爬虫获得天气图标 - 50 -
2.5 获取天气信息和更新界面 - 51 -
2.6 Service 的学习和使用 - 52 -
2.7 信息在活动间的传递 - 54 -

Part.3 开源讲座报告 - 57 -
3.1 9.20 日讲座 - 57 -

开源的概念和意义 - 57 -
版本控制 Version Control - 59 -
最好的版本管理工具：Git - 59 -
开源社区 - 60 -
开源项目 - 61 -
在讲座中学到的其他内容 - 61 -
3.2 9.27 日讲座 - 62 -
第一部分 - 62 -
第二部分 - 63 -
开始 - 64 -
参考资料 - 65 -

Part.4 开源项目阅读 - 66 -
4.1 第一周学习报告 - 66 -
4.2 第二周学习报告 - 66 -
4.3 第三周学习报告 - 70 -

Part.5 其他 - 71 -
5.1 Git 的使用 - 71 -
Git 指令 - 71 -
Android Studio 与 Github 的关联 - 75 -
5.2 简单爬虫 - 77 -
知识点 - 77 -
程序编写 - 78 -
问题和总结 - 84 -

Part.6 总结 - 85 -

Part.1 算法题

1.1 爬楼梯

题目描述：

题号：LeetCode 70

时间：8.19-8.25

假设你正在爬楼梯。需要 *n* 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢?

注意：给定 *n* 是一个正整数。

示例 1：

```
输入： 2
输出： 2
解释： 有两种方法可以爬到楼顶。
1. 1 阶 + 1 阶
2. 2 阶
```

我的思路：

直接考虑的话，从最底下向上爬楼梯，在每一级都有两种选择，到达下一步后又有两种选择，以此类推。按照这个思路一时摸不着头脑。逆向思考一下，从最高一级往下倒着走，到达第 n 阶台阶有两种方法，从第 n-1 阶台阶向上一步，或者从第 n-2 阶台阶向上两步。如果到达了第 n-1 阶台阶，那么别无选择，下一次只能走一步；如果到达了第 n-2 阶台阶，下一次若走一步，那么和前一种情况相同，所以同样只能走两步。这样，我们可以将问题简化为，对到达第 n 阶台阶的可能次数，就等于到达第 n-1 阶台阶的可能次数，加上到达第 n-2 阶台阶的可能次数。

具体方法来看：

1、首先能想到的就是递归的方法，递归求解 n-1 和 n-2 阶楼梯的步数和，但这样果然“达成了”超时的结果；

2、大一时某次考试，考察了一个求某函数在参数 n 取 1 到 5 的各个结果，那个题目的函数是递归求解的，但当时还不知道递归，所以在解题时候从草稿纸上做了很多冗余的计算，例如 n=3 时求了一遍函数值，n=4 时又要算一遍 n=3 的值，诸如此类，浪费了很多计算的时间。这里思路 1 遇到了超时，同样是因为做了很多重复计算，那么不妨利用一个静态数组，对 n 的每个取值的计算结果，若是初次计算则存入数组中，后面再用到时就直接从数组中取出。前人种树，后人乘凉。

算法代码：

```java
public class Climb_Stairs_70 {

    static int[] array;
    public static int solution(int n) {
        if(n==1) return 1;
        if(n==2) return 2;
        array = new int[n+1];
        array[1]=1;
        array[2]=2;
        return compute(n);
    }

    public static int compute(int n) {
        if(array[n]!=0)
            return array[n];
        else {
            array[n] = compute(n-1)+compute(n-2);
            return array[n];
        }

    }

}
```

其他解法：

1、同样是简单粗暴的利用递归，但是借助 Python 有缓存的思路，相当于利用缓存自动存储了已经计算过的 n 的取值。用作者的话说，“抖个机灵”。

```python
from functools import lru_cache

class Solution:
    @lru_cache(10**8)
    def climbStairs(self, n):
        """
        :type n: int
        :rtype: int
        """

        if n == 1:
            return 1
        elif n == 2:
            return 2
        else:
            return self.climbStairs(n - 1) + self.climbStairs(n - 2)
```

借机了解一下 Python 的 functools 模块。官方介绍为：该模块用于高阶函数，即作用于或返回其他函数的函数。对于这个模块，任何可调用的对象都可以被视为函数，如

同其名字，用于 function 的 tools。

此处用到的 lru_cache 函数，用来申请一块缓存来保存最近调用的函数结果，括号中的参数就是缓存区的大小。当调用了相同参数的函数时，就可以直接从缓存中提取结果。使用该函数的语法如上所示，需要用@来调用。

2、Java 代码的返回值要求为 int，经计算当 n=46 时，结果会发生溢出，所以必有 n<46，可以直接罗列全部结果。

```java
public int climbStairs(int n) {

    int result = 0;

    switch(n){
    case 1: result = 1; break;
    case 2: result = 2; break;
    case 3: result = 3; break;
    case 4: result = 5; break;
    case 5: result = 8; break;
    case 6: result = 13; break;
    case 7: result = 21; break;
    case 8: result = 34; break;
    case 9: result = 55; break;
    case 10: result = 89; break;
    case 11: result = 144; break;
    case 12: result = 233; break;
    case 13: result = 377; break;
    case 14: result = 610; break;
    case 15: result = 987; break;
    case 16: result = 1597; break;
    case 17: result = 2584; break;
    case 18: result = 4181; break;
    case 19: result = 6765; break;
    case 20: result = 10946; break;
    case 21: result = 17711; break;
    case 22: result = 28657; break;
    case 23: result = 46368; break;
    case 24: result = 75025; break;
    case 25: result = 121393; break;
    case 26: result = 196418; break;
    case 27: result = 317811; break;
    case 28: result = 514229; break;
    case 29: result = 832040; break;
    case 30: result = 1346269; break;
    case 31: result = 2178309; break;
    case 32: result = 3524578; break;
    case 33: result = 5702887; break;
    case 34: result = 9227465; break;
    case 35: result = 14930352; break;
    case 36: result = 24157817; break;
    case 37: result = 39088169; break;
    case 38: result = 63245986; break;
    case 39: result = 102334155; break;
    case 40: result = 165580141; break;
    case 41: result = 267914296; break;
    case 42: result = 433494437; break;
    case 43: result = 701408733; break;
    case 44: result = 1134903170; break;
    case 45: result = 1836311903; break;

    }
    return result;
}
```

这个方法虽然看起来很愚笨，但事实上，它提醒我们，面对取值域较小的问题，直接返回值并不是错误的，毕竟我们面对问题的最终目的就是解决这个问题。随时要记住，不要忽略了实际情况来解决问题，要因地制宜，因材施教。

1.2 逆波兰表达式

题目描述：

题号：LeetCode 150

时间：9.2-9.9

根据逆波兰表示法，求表达式的值。

有效的运算符包括 `+`，`-`，`*`，`/` 。每个运算对象可以是整数，也可以是另一个逆波兰表达式。

说明：

- 整数除法只保留整数部分。
- 给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。

示例 1：

```
输入：["2", "1", "+", "3", "*"]
输出：9
解释：((2 + 1) * 3) = 9
```

我的思路：

借助 TQL 公众号的总结和题目推荐，集中做了一部分栈的题目，总结出了一点心得。最开始学习栈的练习题目就是经典的括号匹配问题了，它的思路是，遇到左括号就入栈，遇到右括号就从栈中提取一个与之进行配对。换言之，每一个左括号都是在栈中“苦苦等待”着后面的右括号来将它带走。最终，如果栈空，也就是所有的左括号都“配对成功”，意味着括号匹配成功，反之失败。

事实上，很多关于栈的问题都可以抽象到括号匹配的这个模板上。无论每个问题中是左右括号()匹配，还是 abc 字符串匹配，还是操作数和操作符匹配等等，我们都将它们统一到括号匹配的问题上，用类似的思路进行思考。对问题进行归一化的关键在于，找到每个问题中，扮演左括号的角色（在遍历输入的内容时，遇到之后需要入栈的字符或字符串）和扮演右括号的角色（遇到之后，可以和栈顶元素进行配对，产生出栈操作的字符或字符串）。

当遇到操作符的时候，前面一定是有操作数等待着被这个操作符运算的。需要等待的操作数就是这个问题中的左括号，而可以一次性提取出两个操作数来进行计算的操作符就是这个问题中的右括号。略微区别的是，一个操作符可以和两个操作数进行配对，可以当做一个右括号和两个左括号进行配对。并且，计算的结果还要再压入栈中，可以当做一个右括号和两个左括号配对后，变化成了一个左括号，等待着后续的配对。

算法代码：

```java
public int evalRPN(String[] tokens) {

    int result = 0;
    Stack<Integer> stack = new Stack<Integer>();

    for(String s : tokens) {
        if(s.equals("+")) {
            stack.push(stack.pop()+stack.pop());
        } else if(s.equals("-")) {
            stack.push(-stack.pop()+stack.pop());
        } else if(s.equals("*")) {
            stack.push(stack.pop()*stack.pop());
        } else if(s.equals("/")) {
            int temp = stack.pop();
            stack.push(stack.pop()/temp);
        } else {
            stack.push(Integer.parseInt(s));
        }
    }

    result = stack.pop();
    return result;
}
```

1.3 棒球比赛

题目描述：

题号：LeetCode 682

时间：9.2-9.9

你现在是棒球比赛记录员。
给定一个字符串列表，每个字符串可以是以下四种类型之一：
1. 整数 （一轮的得分）：直接表示您在本轮中获得的积分数。
2. "+" （一轮的得分）：表示本轮获得的得分是前两轮 有效 回合得分的总和。
3. "D" （一轮的得分）：表示本轮获得的得分是前一轮 有效 回合得分的两倍。
4. "C" （一个操作，这不是一个回合的分数）：表示您获得的最后一个 有效 回合的分数是无效的，应该被移除。

每一轮的操作都是永久性的，可能会对前一轮和后一轮产生影响。
你需要返回你在所有回合中得分的总和。

示例 1:

```
输入：["5","2","C","D","+"]
输出：30
解释：
第1轮：你可以得到5分。总和是：5。
第2轮：你可以得到2分。总和是：7。
操作1：第2轮的数据无效。总和是：5。
第3轮：你可以得到10分（第2轮的数据已被删除）。总数是：15。
第4轮：你可以得到5 + 10 = 15分。总数是：30。
```

我的思路：

经过上一道题目的分析，熟悉了将新问题套接到旧问题括号匹配的方法。这个题目拿过来，乍一看有点吓唬人，仔细分析，还是熟悉的味道，都是纸老虎（张宇老师教得好，papertiger）。正常的整数，不需要操作，只需要积分，本可以直接加起来求和，但有些操作还需要利用到前面的分数，所以就存到栈里备用嘛，这就是这道题里的左括号；符号+代表获得前两轮积分之和，这就是需要看一看栈顶的两个元素嘛，求和之后再存入栈中，只不过这里前两轮积分只需要拿出来用一下，取完之后还得放回去；符号 D 同理，比+还简单，只需要看一看栈顶元素，然后 double 它就好了。符号 C 会删掉最近的一个得分，那就很明显是这个题目中的右括号了，会和栈顶的一个数据匹配，然后消除掉。

本题不是匹配问题，最终遍历完输入的内容后栈显然不是空的，我们还需要将栈中所有的元素取出来求个和，得到最后的结果。

在代码编写上，借鉴上一题学到的编程小技巧，将特殊的较少的操作符用 if-else 语句专门处理，剩下的 general 的操作数默认就放进去了。另外，由于题目的特性，不会出现异常的输入，所以说当出现操作符的时候前面一定是有操作数的，默认直接提取就可以了，这样避免了判断的麻烦。

当然，在实际的应用过程中，还是必须要考虑种种异常情形的。因为你永远不知道测试会发生什么样奇怪的 input。

算法代码：

```java
public static int calPoints(String[] ops) {

    int result = 0;

    Stack<Integer> stack = new Stack<Integer>();

    for(String op : ops) {
        if(op.equals("C")) {
            stack.pop();
        } else if(op.equals("D")) {
            int num = stack.peek();
            stack.push(num*2);
        } else if(op.equals("+")) {
            int num1 = stack.pop();
            int num2 = stack.peek();
            stack.push(num1);
            stack.push(num1+num2);
        } else {
            stack.push(Integer.parseInt(op));
        }
    }
    for(int num : stack) {
        result+=num;
    }
    return result;
}
```

1.4 检查替换后的词是否有效

题目描述：

题号：LeetCode 1003

时间：9.2-9.9

给定有效字符串 "abc"。

对于任何有效的字符串 V，我们可以将 V 分成两个部分 X 和 Y，使得 X + Y（X 与 Y 连接）等于 V。（X 或 Y 可以为空。）那么，X + "abc" + Y 也同样是有效的。

例如，如果 S = "abc"，则有效字符串的示例是："abc"，"aabcbc"，"abcabc"，"abcabcababcc"。**无效**字符串的示例是："abccba"，"ab"，"cababc"，"bac"。

如果给定字符串 S 有效，则返回 true；否则，返回 false。

示例 1：

```
输入："aabcbc"
输出：true
解释：
从有效字符串 "abc" 开始。
然后我们可以在 "a" 和 "bc" 之间插入另一个 "abc"，产生 "a" + "abc" + "bc"，即 "aabcbc"。
```

我的思路：

还是一样的思路。

考虑情况，只有遇到 c 的时候，我们才需要进行提取，也就是说 c 相当于括号类型题目中的右括号。遇到 ab 都是直接往里塞就可以。因为 ab 相当于左括号，可以一直一直堆积，类似于左括号在一直一直堆积，只需要后面有足够数量的、顺序正确的 c 来将它们成功配对就 ok 了。当然，遇到 c 的时候，我们必须确保栈顶的元素是 b，栈顶下面紧跟的元素是 a，有任意一步不符合都是配对失败的。

这道题目应该是比前面的又要复杂一些了，但这几道栈的题目思路都很类似，由浅入深，有熟能生巧之感。这也是学习的一种技巧吧。举一反三，善之善者也。

算法代码：

```java
public boolean isValid(String s) {

    Stack<Character> stack = new Stack<Character>();

    char[] sArray = s.toCharArray();
    for(char cur : sArray) {
        if(cur=='c') {
            if(!stack.empty()&&stack.pop()=='b') {
                if(!stack.empty()&&stack.pop()=='a') {
                    continue;
                } else {
                    return false;
                }
            } else {
                return false;
            }
        } else {
            stack.push(cur);
        }

    }
    if(stack.empty()) {
        return true;
    }

    return false;
}
```

1.5 简化路径

题目描述：

题号：LeetCode 71

时间：9.2-9.9

以 Unix 风格给出一个文件的**绝对路径**，你需要简化它。或者换句话说，将其转换为规范路径。

在 Unix 风格的文件系统中，一个点（`.`）表示当前目录本身；此外，两个点 （`..`） 表示将目录切换到上一级（指向父目录）；两者都可以是复杂相对路径的组成部分。更多信息请参阅：Linux / Unix中的绝对路径 vs 相对路径

请注意，返回的规范路径必须始终以斜杠 `/` 开头，并且两个目录名之间必须只有一个斜杠 `/`。最后一个目录名（如果存在）**不能**以 `/` 结尾。此外，规范路径必须是表示绝对路径的**最短**字符串。

示例 1：

```
输入："/home/"
输出："/home"
解释：注意，最后一个目录名后面没有斜杠。
```

我的思路：

这里稍微有点不同的是，输入的是整个字符串，但实际上有效的内容，例如某个目录的名称，都是单词，也就是若干个字符在一组才是有意义的。

这样按照每个字符进行读取就很僵硬，想到利用输入的路径中天生的分隔符斜杠/来对字符串进行 split，这样就将字符串转化成了我们熟悉的往栈中输入的字符串数组。

类似的思想，普通的目录名称，我们就进栈；返回上一级的两个点..相当于右括号，我们就出栈。保持当前目录的一个点.，我们可以直接 continue 掉。

这样问题又转化成了我们熟悉的模式。

此外，再注意一些例如输出的格式，只有根目录时候的特殊性等等问题，就搞定了。

算法代码：

```
public static String simplifyPath(String path) {
    Stack<String> stack = new Stack<String>();
    String[] array = path.split("/");
    StringBuilder result = new StringBuilder("");
    for(String s : array) {
        if(s.equals(".")||s.equals("")) {
            //  表示当前目录，不需要处理
            continue;
        } else if(s.equals("..")) {
            //  表示跳转上一级目录，那么取消上一级的跳转。但如果就在根目录下，则不用跳转
            if(!stack.isEmpty()) {
                stack.pop();
            }
        } else {
            stack.push(s);
        }
    }
    if(stack.empty()) {
        //  如果结果是根目录，那么需要增加一个/来表示之
        result.append("/");
    } else {
        for(String s : stack) {
            result.append("/");
            result.append(s);
        }
    }
    return result.toString();
}
```

1.6 下一个更大元素 1

题目描述：

题号：LeetCode 496

时间：9.11-9.12

给定两个**没有重复元素**的数组 `nums1` 和 `nums2` ，其中 `nums1` 是 `nums2` 的子集。找到 `nums1` 中每个元素在 `nums2` 中的下一个比其大的值。

`nums1` 中数字 **x** 的下一个更大元素是指 **x** 在 `nums2` 中对应位置的右边的第一个比 **x** 大的元素。如果不存在，对应位置输出-1。

示例 1:

```
输入: nums1 = [4,1,2], nums2 = [1,3,4,2].
输出: [-1,3,-1]
解释:
    对于num1中的数字4，你无法在第二个数组中找到下一个更大的数字，因此输出 -1。
    对于num1中的数字1，第二个数组中数字1右边的下一个较大数字是 3。
    对于num1中的数字2，第二个数组中没有下一个更大的数字，因此输出 -1。
```

我的思路：

这一类求“next greater element”的题目基本都可以归结到单调栈类型的问题。可以说，单调栈就是为了处理这一类典型的问题而定义的。

所谓单调栈，是指对数据入栈过程进行控制，确保栈中元素始终保持单调。例如，一个从栈底到栈顶单调递增的单调栈，每当新元素入栈时都要和栈顶元素进行比较。若大于栈顶元素，则正常入栈；若小于栈顶元素，则要对栈中现存元素进行出栈，直到栈顶元素小于当前元素，以确保当前元素入栈后不会破坏单调性。

刚面对这道题目的时候，我还不了解单调栈的知识，于是按照普通的思维开始思考。这道题目分为两步：

1 遍历 nums1 数组，找到当前元素在 nums2 中的位置 i；

2 从索引 i 开始向后遍历 nums2，返回遇到的第一个更大的元素；

为了便于完成第一步，我首先将 nums2 中的元素存到一个哈希表中；然后要进行双层的遍历，时间复杂度较高。做完后想到，事实上如果不利用哈希表，直接双层遍历，复杂度事实上是一样的，只不过是因为习惯了用哈希而总是想刻意的去用哈希，但做题目还是应该具体问题具体分析的。尤其是在有些时候题目规模较小，进行一次遍历去完成哈希表的构建可能还会浪费时间。

算法代码：

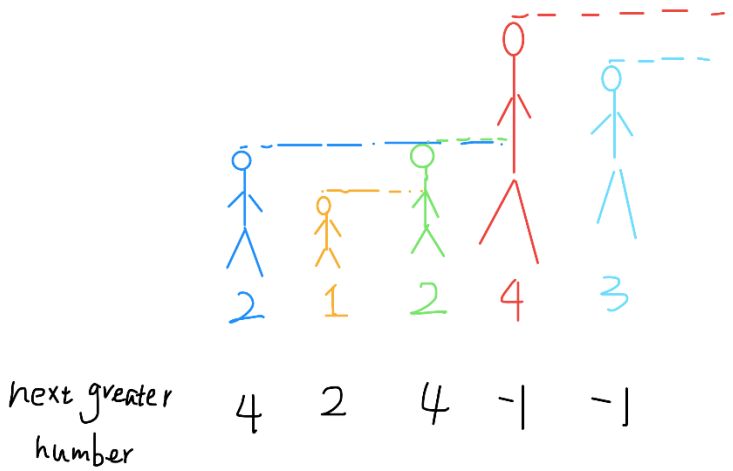
```java
public int[] nextGreaterElement(int[] nums1, int[] nums2) {
    //  利用哈希表记录nums2数组中每个元素所在的位置。双层遍历。
    Map<Integer, Integer> map = new HashMap<Integer, Integer>();
    int[] result = new int[nums1.length];
    for(int i=0;i<nums2.length;++i) {
        map.put(nums2[i], i);
    }

    for(int i=0;i<nums1.length;++i) {
        //  找到该元素在nums2数组中的位置
        int position = map.get(nums1[i]);
        int j = 0;
        for(j=position+1;j<nums2.length;++j) {
            if(nums2[j]>nums1[i]) {
                result[i] = nums2[j];
                break;
            }
        }
        if(j==nums2.length) {
            result[i] = -1;
        }
    }
    return result;
}
```

标准解法：

题解中对单调栈思路的讲解非常形象和易懂，我来试着描述一下。如果把数组中的数字按照大小想象成个头高矮的人，前后排列站好，那么显然，较矮的人会被前面的高个子挡住。换言之，如果一个矮个子前面有高个子，那么从后往前看，这个矮个子就相当于不存在。

借一个图：



转换成数字，前面这段话的意思就是，如果一个较小的数前面有更大的数（例如 3 前面有 4），那么再往前面的所有数字（4 前面的 2，1，2）在向后找 next greater element 的时候，一定不会选中 3（因为 3 被 4 挡住了，换言之，4 的存在让 3 失去了意义）。这不代表前面的数字一定会选中 4（例如 1 后面的 2 就比 1 大），但一定

不会选中 3。

但是我们可以直接删掉 3 吗(既然它已经这么没有意义了)？显然是不可以的。注意我前面标红的字，3 只是对限定在 4 前面的数字是没有意义的，但是 4 以后的数字仍然是需要 3 的。例如在 4 和 3 中间再插入一个 1（数列变成 2 1 2 4 1 3)，显然对于 1 它的 next greater element 就是 3 了。

为了利用这样的特点，我们逆向思考，考虑将 nums2 中的元素从后往前入栈。每次当前元素入栈的时候，先看看它和栈顶元素的大小关系（注意，当前元素在数列中位置靠前，栈顶元素在数列中位置靠后）即当前元素是“站在更靠前的人”，如果他个子较高，那么他会挡住栈顶元素；如果他个子较矮，那么栈顶元素对于他就是 next greater element。即，如果当前元素大于栈顶元素，那么栈顶元素就可以出栈了，因为当前元素会“挡住它”。当然，如果出栈后下一个栈顶元素依然比当前元素小，则仍需要出栈。所以此处要用到一个 while 循环。

（例子 1，栈中自顶向下是 1 2 3 4 5，当前元素是 6，那么 6 会一口气挡住栈中全部元素，1 2 3 4 5 会依次出栈)。

（例子 2，栈中自顶向下是 1 2 3 4 5，当前元素是 4，那么 4 会一口气挡住 1 2 3 4——虽然栈顶元素 4 和当前元素 4 相同，但仍然会让其中一个没有意义。而下面碰到 5，4 比 5 小，5 就不用出栈了，4 挡不住它，而且 5 会作为 4 的 next greater element)。

但这只是对栈的操作。前面的 while 循环结束后，我们看看栈中是否还有剩下的元素。如果没有（例如上一段的举例 1，1 2 3 4 5 全部出栈，栈空)，说明后面没有比它更大的元素，那么应该用-1 来记录；如果有，那么栈顶的这个元素就是它的 next greater element（如上一段的举例 2)。这时候，哈希表就派上用场了。我们可以用（元素，对应的 next greater element）来作为键值对存储到哈希表中，来便于查找。

```java
public int[] nextGreaterElement2(int[] nums1, int[] nums2) {
    int[] result = new int[nums1.length];
    Stack<Integer> stack = new Stack<Integer>();
    Map<Integer, Integer> map = new HashMap<Integer, Integer>();

    for(int i=nums2.length-1;i>=0;--i) {
        //  倒序入栈
        while(!stack.empty()&&stack.peek()<=nums2[i]) {
            //  如果后面的数字（先入栈的数字）比现在的数字小，那么可以抛弃了
            //  因为再往前的数字向后找大数的时候，会被这个大的拦住，不可能看得到后面的小数了
            stack.pop();
        }
        if(stack.empty()) {
            map.put(nums2[i], -1);
        } else {
            map.put(nums2[i], stack.peek());
        }
        stack.push(nums2[i]);
    }

    int i=0;
    for(int num : nums1) {
        result[i++] = map.get(num);
    }
    return result;
}
```

小结：

因为这一段单调栈的知识是第一次遇到，而这个题目是这一知识点最基本的题型，可以作为模板题来学习和记忆，所以啰嗦的多了一些，也是为了让自己之后看到的时候能容易想起来题目的思路吧。

经过一段时间的题目练习，我发现很多解题思路都可以在多道同型题目中举一反三的，区别仅在于对结果的处理方式或者输出方式等，做题的时候要透过现象看本质，多做同类型的题目来熟能生巧（再次感谢 tql 提供的题型分类和同类型题目总结）。

1.7 下一个更大元素 2

题目描述：

题号：LeetCode 503

时间：9.15-9.16

给定一个循环数组（最后一个元素的下一个元素是数组的第一个元素），输出每个元素的下一个更大元素。数字 x 的下一个更大的元素是按数组遍历顺序，这个数字之后的第一个比它更大的数，这意味着你应该循环地搜索它的下一个更大的数。如果不存在，则输出 -1。

示例 1:

```
输入: [1,2,1]
输出: [2,-1,2]
解释: 第一个 1 的下一个更大的数是 2;
数字 2 找不到下一个更大的数;
第二个 1 的下一个最大的数需要循环搜索，结果也是 2。
```

我的思路：

利用类似于解决递归问题复杂性太高的思想，将已经计算出的结果存储在 result 数组中，从后面往前比较，每次先和后面的元素比较，如果后面的小于当前元素，那么可以和后面元素在 result 数组中的结果比较。

因为是从后往前比较，所以前面的基本都能找到 result 的结果。当然，数组是循环的，也可能碰到 result 中还没存到结果的情况。

Java 给整型数组默认初始化是 0。但是由于输入的 nums 数组中可能有大量负数，所以这个 0 值对于很多数来说已经可以作为 next greater element 了（事实上没有到达 0 那么大），所以我首先初始化它们为 integer 类的最小值。

算法代码：

```java
public int[] nextGreaterElements(int[] nums) {

    int[] result = new int[nums.length];
    for(int i=0;i<result.length;++i) {
        result[i] = Integer.MIN_VALUE;
    }

    for(int i=nums.length-1;i>=0;--i) {
        boolean flag = false;
        int j=i+1;
        while(j<nums.length) {
            if(nums[i]<nums[j]) {
                result[i] = nums[j];
                flag = true;
                break;
            } else if(result[j]>nums[i]) {
                result[i] = result[j];
                flag = true;
                break;
            } else {
                j++;
            }
        }
```

```
        if(!flag) {
            for(j=0;j<i;j++) {
                if(nums[i]<nums[j]) {
                    result[i] = nums[j];
                    flag = true;
                    break;
                } else if(result[j]>nums[i]) {
                    result[i] = result[j];
                    flag = true;
                    break;
                }
            }
        }
        if(!flag) {
            result[i] = -1;
        }
    }
    return result;
```

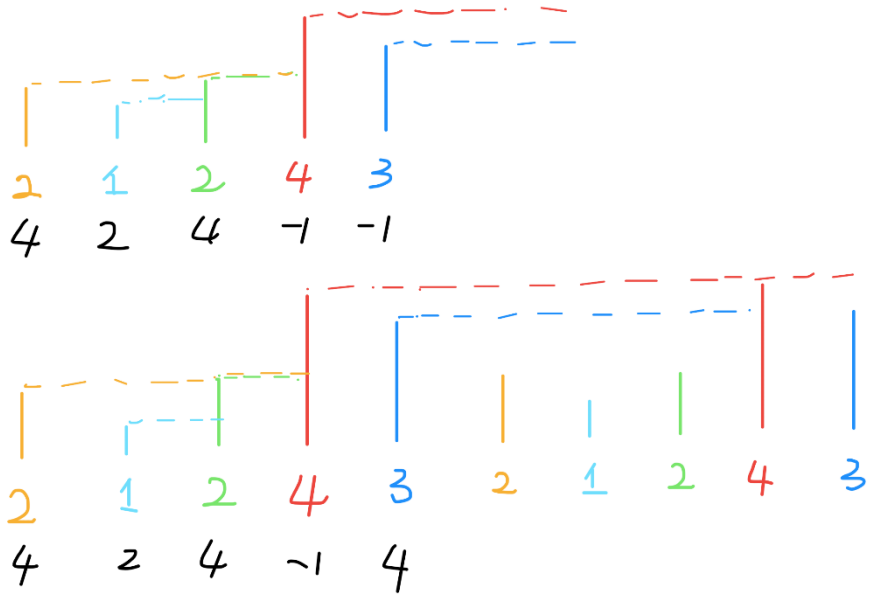
标准解法：

前面说过，next greater element 类型的问题是单调栈的典型应用问题，可以套用其模板。但前面学过的只是向后找（找数组中位于当前元素右边的元素）更大的元素，本题将数组扩展到循环数组，这意味着当前元素左边的元素也可以被选中。

换言之，这一次，每个元素都有两次机会找到它的 next greater element。为了模拟循环的效果，可以将数组模拟成原来的两倍大小。这个模拟可以通过直接扩展数组的方式，也可以利用取余运算来简化操作。

其他的部分依然可以借用单调栈的算法模板。

借一个图：



```java
public int[] nextGreaterElements2(int[] nums) {
    int n = nums.length;
    int[] result = new int[n];
    Stack<Integer> stack = new Stack<Integer>();

    for(int i=2*n-1;i>=0;--i){
        while(!stack.empty()&&nums[i%n]>=stack.peek()) {
            stack.pop();
        }
        if(stack.empty()) {
            result[i%n]=-1;
        } else {
            result[i%n] = stack.peek();
        }
        stack.push(nums[i%n]);
    }

    return result;
}
```

1.8 股票跨度

题目描述：

题号：LeetCode 901

时间：9.15-9.16

编写一个 `StockSpanner` 类，它收集某些股票的每日报价，并返回该股票当日价格的跨度。

今天股票价格的跨度被定义为股票价格小于或等于今天价格的最大连续日数（从今天开始往回数，包括今天）。

例如，如果未来7天股票的价格是 `[100, 80, 60, 70, 60, 75, 85]`，那么股票跨度将是 `[1, 1, 1, 2, 1, 4, 6]`。

示例：

```
输入：["StockSpanner","next","next","next","next","next","next","next"], [[],[100],[80],[60],[70],[60],[75],[85]]
输出：[null,1,1,1,2,1,4,6]
解释：
首先，初始化 S = StockSpanner()，然后：
S.next(100) 被调用并返回 1，
S.next(80) 被调用并返回 1，
S.next(60) 被调用并返回 1，
S.next(70) 被调用并返回 2，
S.next(60) 被调用并返回 1，
S.next(75) 被调用并返回 4，
S.next(85) 被调用并返回 6。

注意 (例如) S.next(75) 返回 4，因为截至今天的最后 4 个价格
(包括今天的价格 75) 小于或等于今天的价格。
```

我的思路：

题目比较复杂，首先分析各种特殊的情况。

首先，如果当前元素前面没有数，或者前面的数比它大（破坏了要求前面连续小于等于当前元素的条件），则不需要进行更复杂的推算，就只有它本身可以被计算

（注意这里要返回的是 1,而非 0，因为题目条件是包括今天的，这一点后面还会遇到坑）。

其次，如果当前元素前面有数，我们就要进行一些观察了。这里首先要考虑，因为本题输入的特殊形式，是通过不断的函数调用的形式来给出元素序列，而非直接告诉我们一个元素的数组，所以需要在这个 StockSpanner 类中构建静态的栈，来进行元素记录。

然后我们考虑如何构造栈。前面说过，如果比前面小，我们可以直接返回 1 作为结果，利用之前学习单调栈的思路，这个元素不会“挡住”前面更大的元素，我们把它放到栈中；反之，如果当前元素大于前面的元素（栈顶元素），我们就要一直往前找，一直找到某个元素大于它，则前面这些都是连续的、比当前元素小的部分。

但是，我们举一个例子，譬如栈中已有的序列为【1（栈底） 3 5（栈顶）】，此时要判断的元素是 7，它开始从栈顶往下找，5，3，1，都比它小，它应该返回 4，并且把自己加入栈顶，序列变成 1 3 5 7；下一个要判断的元素是 9，它又开始往下找，7,5,3,1，都比它小，它应该返回 5，并且把自己加入栈顶。这时候问题来了，5 3 1 这个序列被数了两次，这种冗余是不能接受的。

但这种冗余也提示了我们规律的所在。9 和 7 都数了 5 3 1 序列，9 还把 7 数了进去，这说明当当前元素大于栈顶元素时，它会包含栈顶元素对应的值；然后，栈顶元素出栈，栈顶元素下面的元素仍有可能比当前元素小，则继续要把下面元素对应的值加上。这又回到了我们熟悉的 while 循环模式，同时栈也会被构造成一个从栈底到栈顶单调递减的单调栈。

算法代码：

```
public static Stack<Integer> stack ;
public static Stack<Integer> result;     //  另一个栈存储对应位置的个数
public StockSpanner901() {
    stack = new Stack<Integer>();
    result = new Stack<Integer>();
}

public int next(int price) {
    //  先把当前元素入栈
    if(stack.empty()||price<stack.peek()) {
        //  前面没有元素，则只能是1
        stack.push(price);
        result.push(1);
        return 1;
    }

    int ans = 1;
    while(!stack.empty()&&price>=stack.peek()) {
        stack.pop();
        ans = ans+result.pop();
    }
    stack.push(price);
    result.push(ans);

    return ans;
}
```

优化解法：

根据评论中提示的思路，最后一条 case 容易超时，这条数据中存在大量的连续重复输入，所以可以考虑 cache 的思想，记录下上一次的输入和结果。

```
public static int lastCase;
```

```
if(price==lastCase) {
    int temp = result.peek();
    stack.push(price);
    result.push(temp);
    return temp;
}
```

1.9 盛最多水的容器

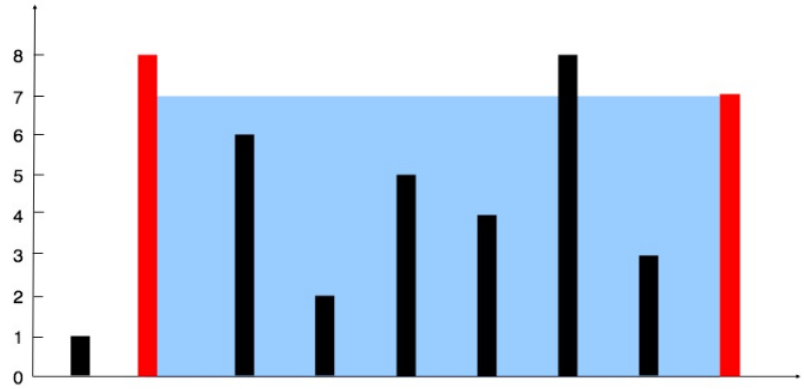
题目描述：

题号：LeetCode 11

时间：9.17-9.18

给定 n 个非负整数 a_1，a_2，…，a_n，每个数代表坐标中的一个点 (i, a_i) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, a_i) 和 $(i, 0)$。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器，且 n 的值至少为 2。



图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例:

```
输入：[1,8,6,2,5,4,8,3,7]
输出：49
```

我的思路：

双指针分别指向两头。每次双指针容纳的面积，应该和两头较短的那一条有关，然后乘上它们的距离。

如果指针向内移动，那么横向距离在缩短。

如果我们移动较大的那一头，那移动后，横向距离缩短，纵向距离不会超过较短的那一头，所以这样面积一定会减少的。

因此，我们只能移动较小的那一头，将较小的向内移动一步，这样，如果移动后变长，它才有可能得到一个更大的面积。

如此迭代，直到双指针相遇。

时间复杂度为 O（n），因为只需要由外向内进行一次扫描。

算法代码：

```
public static int maxArea(int[] height) {
    int left = 0;
    int right = height.length-1;
    int result = -1;
    int lower = -1;
    while(left<right) {
        if(height[left]<height[right]) {
            lower=height[left];
            left++;
        } else {
            lower = height[right];
            right--;
        }
        int temp = lower*(right-left+1);
        if(temp>result) {
            result = temp;
        }
    }
    return result;
}
```

1.10 加一

题目描述：

题号：LeetCode 66

时间：9.17-9.18

给定一个由**整数**组成的**非空**数组所表示的非负整数，在该数的基础上加一。

最高位数字存放在数组的首位， 数组中每个元素只存储**单个**数字。

你可以假设除了整数 0 之外，这个整数不会以零开头。

示例 1:

```
输入: [1,2,3]
输出: [1,2,4]
解释: 输入数组表示数字 123。
```

我的思路：

这不是一道难题，我的思路就很直接，从末位开始往前看（从低位往高看），每一次将当前位加一。如果当前位不存在进位的情况（即当前位为 0-8），那么再往前也同样不会产生进位，可以直接跳出循环，输出结果；

如果在某一位遇到了 9，那么这一位加一后会变成 0，同时产生一个进位。如何表示进位呢？只需要继续往前循环就可以了。循环未停止意味着产生了进位（否则就 break 掉了）。

最终，如果判断循环进行到了最前面一位，说明整体又产生了一个进位（即给出的数据是 99,999 之类），并且产生的进位一定是 1，那么就需要新建一个更大的数组，在数组第 0 位读入 1，后面的位将原数组中对应位置放进去。

算法代码：

```java
public static int[] solution(int[] digits) {
    int n = digits.length;
    int i;
    for(i=n-1;i>=0;--i) {
        if(digits[i]<9) {
            digits[i]++;
            break;
        }
        digits[i]=0;
    }

    if(i<0) {
        int[] array = new int[n+1];
        array[0]=1;
        for(int j=0;j<n;j++) {
            array[j+1]=digits[j];
        }
        return array;

    } else {
        return digits;
    }

}
```

其他解法：

评论中的一个思路太巧妙了。

首先把最后的一位加一，如果没有产生进位，那么再往前也不会产生进位，直接返回数组；

如果当前位产生了进位，那么再往前看一位，前面这一位也要+1（相当于刚才这部分内容重复一遍，又像是问题规模缩小了 1，有点递归的感觉），这时候如果产生了进位，还是相同的操作；如果没有进位，那么同样可以直接返回。

如果一直看到了第一位，还是发生了+1，那么一定说明后面的所有位（包括第一位）都变成了 0，也就是 99 999 9999 等数字+1 的情况。那么我们直接 new 一个长度增加 1 的数组，将 array[0]位置设置为 1 即可。后面的部分都默认被初始化为了 0.

```java
public int[] plusOne(int[] digits) {
    for(int i=digits.length-1;i>=0;--i) {
        digits[i] = (digits[i]+1)%10;
        if(digits[i]!=0) {
            return digits;
        }
    }
    digits = new int[digits.length+1];
    digits[0] = 1;
    return digits;
}
```

1.11 最大子序和

题目描述：

题号：LeetCode 53

时间：9.21-9.22

给定一个整数数组 `nums` ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

示例:

```
输入：[-2,1,-3,4,-1,2,1,-5,4],
输出：6
解释：连续子数组 [4,-1,2,1] 的和最大，为 6。
```

我的思路：

当面对一个思路不确定的问题时候，我想起之前做类似出栈入栈序列的题目给出的思路，就是模拟一下操作的过程。譬如这道题，最朴素的思路就是双层循环，找到最大的某一段的和。

但是这个过程中肯定会存在一些可以优化的部分。类似于所有找最大值的题目，我们用一个额外的变量 value 来存储当前找到的最大值。

首先，我们在外层循环中，轮到某一个 index 位置的元素开始一段子序列时，如果开头的这个数字是非正数，那么从下一个开始一定是更好的结果，那么直接跳过这一轮就可以了；如果当前值已经大于前面找到的最大值 value 了，那么可以直接用当前值更新 value。

在每一轮外层循环中，还要用一个额外的变量 sum 来存储这一段子序列的最大值。从外层 index 的下一个开始内层循环。当加了某一个元素导致这一轮的 sum 变

成负数，说明前面这一段都没有意义了，这一轮就可以跳过了，下一轮从内层循环找到的位置的下一个开始（不需要完全双层的循环）。但要注意，在跳过之前，还要比较一下加上最后这个元素之前，已经得到的 sum 是否能够大于 value。

如果内层循环找到的当前元素是非负数，那么直接增加到 sum 上是没有问题的；但如果当前元素是负数，不代表它不可以被加上来，因为后面的内容可能会把它给抵消掉，所以可以在增加这个元素之前记录下之前的 sum，如果之前的 sum 大于 value，就先更新一下 value。

最终，如果某一轮内层循环找到了最后，那么说明这一轮循环都是“顺利”的，可以确定当前的 sum 就是最大值。

算法代码：

```
public static int solution(int[] nums) {
    int n = nums.length;
    int i, j;
    int value=Integer.MIN_VALUE;
    for(i=0;i<n;) {
            //  i是一段子序列的开头。如果开头是负数，直接不用看了。
        if(nums[i]>value)
            value = nums[i];
        if(nums[i]<=0) {    //
            ++i;
            continue;
        }
        int sum = nums[i];  //  从当前开始求和

        for(j=i+1;j<n;j++) {    //  从i的下一个位置开始找
            if(nums[j]+sum<=0) {    //  加这个还不如不加
                if(value<sum) {
                    value = sum;    //  当前最大值。虽然这一波要跳过了，但是它前面的可能确实是最大的了。
                }
                i=j+1;
                break;  //  放弃这一波，从后面开始
            }
```

```
            }
            if(nums[j]<0) {
                if(value<sum) {
                    value = sum;    //  这个值是负数，那就先存一下，看看后面还会不会有更大的
                }
                sum = sum+nums[j];
            } else {
                sum = sum+nums[j];
                if(sum>value) {
                    value = sum;
                }
            }
        }
        if(j==n) return value;
    }
    return value;
}
```

其他解法：

由于我在思考这个问题的时候完全是按照思路的逻辑直接写下来的，所以总觉得其中会有冗余的部分。但在我整理自己的代码之前，就看到一个别人的思路特别好理解，代码特别整洁的方法。

只进行一层循环，每一轮都将当前元素也加到当前和 sum 上。当前面的若干个数的和大于 0 时，这一段数的存在就是有必要的，可以一直增加下去（sum>0）.

一旦在某个时刻，sum 的值≤0 了，则前面这些记录都没有意义了，要清空，sum 等于当前的值，重新开始一轮寻找。

利用一个额外的 result 来记录当前已经找到的最大值。

我的思路相比这个思路，的缺点就在于没有使用 result，来以此找到各个部分的最大子数组值，然后将他们分别进行比较；我是试图一次性找到全局的最大值。这样是多此一举的。

```
//  这种思路更简洁严谨，但是复杂度相似
public int maxSubArray(int[] nums) {
    int result = nums[0];
    int sum = 0;
    for(int num: nums) {
        if(sum>0) {
            sum = sum+num;
        } else {
            sum = num;
        }
        result = Math.max(sum, result);
    }
    return result;
}
```

1.12 按奇偶排序数组

题目描述：

题号：LeetCode 905

时间：9.23-9.24

给定一个非负整数数组 `A`，返回一个数组，在该数组中， `A` 的所有偶数元素之后跟着所有奇数元素。

你可以返回满足此条件的任何数组作为答案。

示例：

```
输入：[3,1,2,4]
输出：[2,4,3,1]
输出 [4,2,3,1]，[2,4,1,3] 和 [4,2,1,3] 也会被接受。
```

我的思路：

整体来说，题目的要求是将偶数往前方，奇数往后放，它们的具体次序没有要求。设想一下场景，最前面的奇数可能会被放到最后面，最后面的偶数亦然。这种

“大开大合”的数字大挪移让我想到了快速排序中对大于或小于基准值的数字进行的前后移动，思路就通畅了。

设定两个指针（也是常用的双指针法），一个从前往后找奇数，另一个从后往前找偶数。当两个指针都遇到期望的数字时，进行一下操作。首先看看这两个指针是否找到了数组的端点处，或者是否存在两个指针错位了（后面的指针跑到了前面的指针的前面），这些情况代表数组顺序已经符合要求了，不需要再进行交换操作；如果没有这些特殊情况，就说明前面的指针找到了一个奇数，后面的指针找到了一个偶数，奇数相对于偶数在前面，这时就把两个位置对应的数字交换一下即可。

复杂度分析：时间复杂度为O(N)，因为只进行了一遍扫描；空间复杂度为O(1)，没有利用额外的空间。

算法代码：

```
    int l = a.length;
    int left = 0;
    int right = l-1;     //   双指针从两头开始遍历

    while(left<right) {
        while(left<l-1&&a[left]%2==0) {
            //   找到左边的奇数
            left++;
        }
        while(right>0&&a[right]%2!=0) {
            right--;
        }

        if(left==l-1||right==0||right<=left) {
            return a;
        }

        // swap
        int temp =a[left];
        a[left] = a[right];
        a[right] = temp;
        left++;
        right--;
    }
    return a;
```

其他解法：

利用一个额外的数组，遍历两边待排序数组，第一遍读取偶数存入新数组中，第二遍读取奇数。

时间复杂度同样为 O(N)，但由于要生成一个同样大小的新数组，所以空间复杂度也为O(N)。

稍微优化，也可以只遍历一遍，遇到偶数就从前往后放，遇到奇数就从后往前放。

很少发现自己的思路居然是最优解，还是很开心的。能够想到利用快排的思想和双指针的思路，感觉也是自己的一点点小进步吧。

1.13 数组拆分

题目描述：

题号：LeetCode 561

时间：9.23-9.24

给定长度为 **2n** 的数组, 你的任务是将这些数分成 **n** 对, 例如 $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ ，使得从1 到 n 的 $\min(a_i, b_i)$ 总和最大。

示例 1:

```
输入: [1,4,3,2]

输出: 4
解释: n 等于 2, 最大总和为 4 = min(1, 2) + min(3, 4).
```

我的思路：

看到题目和示例输入输出之后我的第一反应就是将数组排序后，每相邻的两个结成对。尝试着写了一下提交，发现居然正确了。然后我开始思考它的正确性如何判定。

每一个数对中取最小的，意味着每一个数对中只有较小的那个数对于最终的结果有意义，另一个数无论大小都会被它抵消掉，所以说，无论 2 和 1 组对还是 10 和 1 组对，都只有一个 1 会加到结果上，那么我们当然希望能让 2 和 1 组对，而想办法保留下来 10。换个通用的情况来说明，最大的数和最小的数组对，第二大的数和第二小的数组对，最大的数和第二大的数就都没有了。而最大的数和第二大的数组对，就可以保留下来第二大的数，而最小的数和第二小的数组对结果还是一个最小的数。这其中暗含的感觉就是要尽可能的让成对的两个数相差较小。而最能得到这种情况的方案就是将数组排序，然后将相邻的两个数组对。

算法代码：

```java
public int arrayPairSum(int[] nums) {
    Arrays.sort(nums);
    int result = 0;
    int l = nums.length;
    for(int i=0;i<l;i=i+2) {
        result+=nums[i];
    }
    return result;
}
```

1.14 递增三元子序列

题目描述：

题号：课堂练习题

时间：9.25-9.26

编写程序，读取一个整数n，代表数组arr[ ]的长度，之后读取n个整数作为数组arr[n]中的元素。判断这个数组中是否存在长度为 3 的递增子序列并输出。

示例1：

样例输入：
5
1 2 3 4 5
样例输出：
true

示例2：

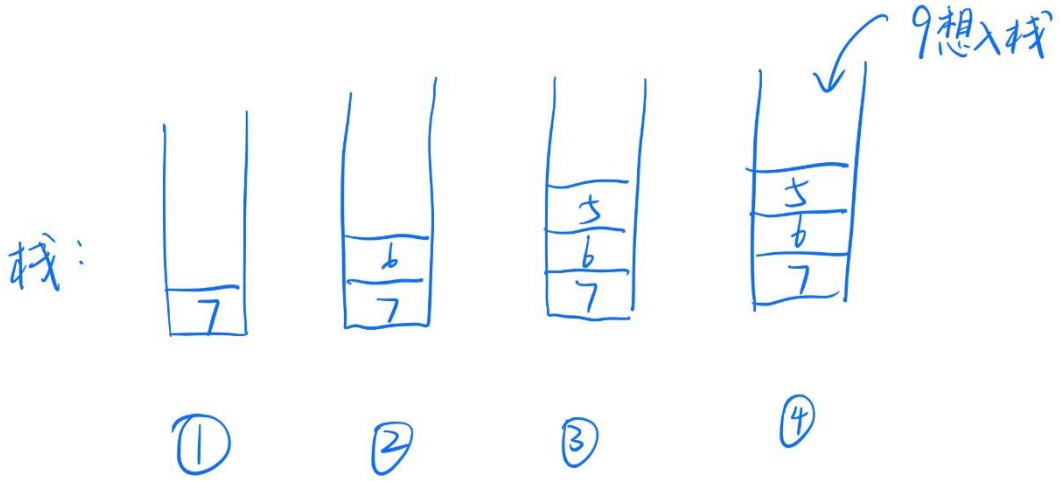
样例输入：
4
5 4 3 2
样例输出：
false

我的思路：

这是一个类似于之前做过的单调栈类型的题目。

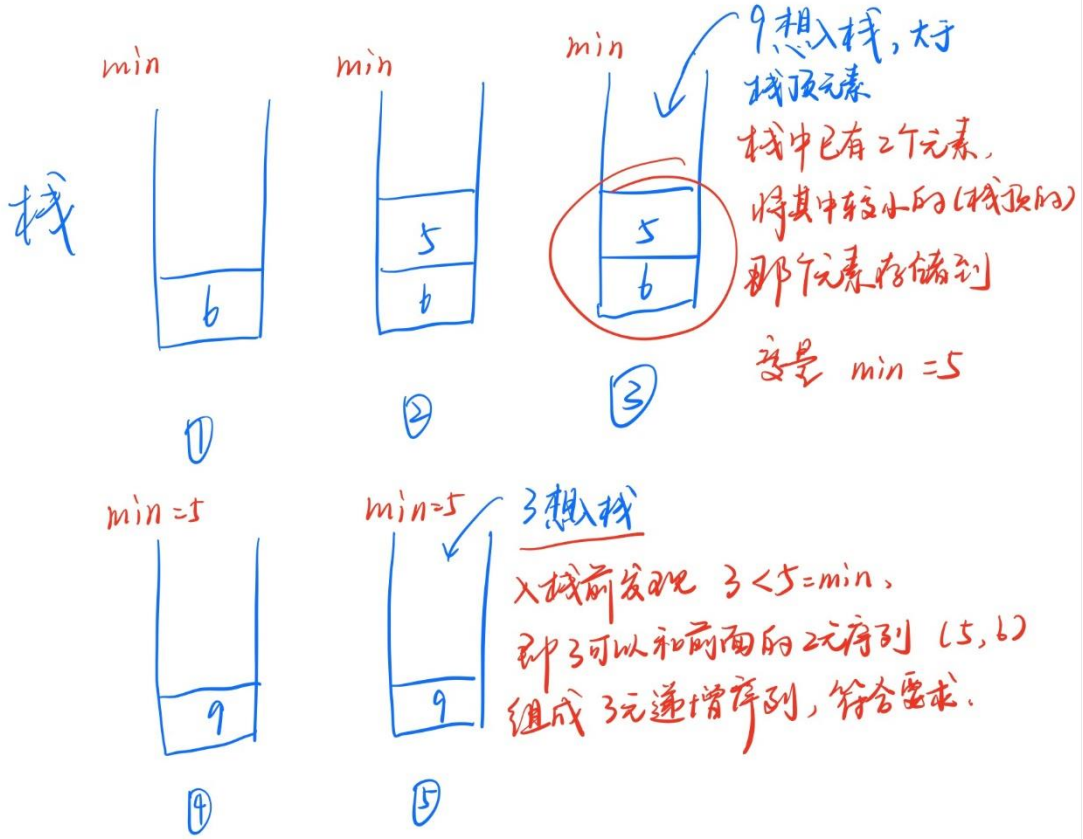
子序列在数组中不需要连续，只需要从前往后保持递增，元素个数达到 3 个就可以。我的思路是构建一个栈，从后往前遍历数组，将数组中的元素按照递减的方式入栈。也就是说，要保持栈中元素自顶向下单调递增，当入栈时的元素比栈顶元素大时，要进行一定次数的出栈操作来保持单调性。

以数组 8 3 9 5 6 7 为例。从后往前遍历，入栈。首先把最后一个元素直接入栈（这时候栈空，直接入栈减少一次循环次数）。接下来的过程如图：



此时，9 想入栈，如果直接按照我们前面的思路，直接将 5 6 7 依次出栈，把 9 放在栈底，那么我们能看到剩下的 8 3 9 已经不能组成三元递增子序列了，但 5 6 7 的确是一个满足题意的要求，所以在每次需要进行出栈操作之前，我们先检查一下栈中元素的个数。如果栈中有不少于 3 个元素，那么由于栈中元素时单调有序的，所以可以确保这些元素满足了三元递增子序列的要求，可以直接返回 true。

那么其他情况要不要单独处理呢？也需要的。换一个例子 8 3 9 5 6 来说明，流程类似，如图所示。



算法代码：

```java
public class Problem5_2 {
    public static void main(String[] args) {

        //  读取数据
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();
        int[] nums = new int[n];
        for(int i=0;i<n;++i) {
            nums[i] = s.nextInt();
        }

        //  初始化一个栈，用于倒序存储数组元素
        Stack<Integer> stack = new Stack<Integer>();
        //  先把数组末位元素压入栈中，减少一次循环
        stack.push(nums[n-1]);
        //  用min变量来保存构成2元序列的出栈元素中的最小值。
        int min = 0;
        //  用一个布尔变量来记录是否有符合条件的2元序列
        boolean flag = false;

        //  倒序遍历数组
        for(int i=n-2;i>=0;--i) {
            //  类似单调栈的思想，如果栈顶元素存在并且小于当前元素，就要将它出栈
            //  利用while，一直找到栈顶元素比当前元素大为止
            while(!stack.empty()&&nums[i]>=stack.peek()) {
```

```java
        //  为了防止已经构成有效的3元序列的元素被弹出无法检测，所以出栈前先进行判断。
        //  因为入栈过程中已经确保了栈中元素自底向上是递减的，所以如果栈中已经有了不少于3个元素，
        //  那么这3个元素在数组中就构成了递增3元子序列，可以直接判定为true
        if(stack.size()>=3) {
            System.out.println("true");
            return;
        }
        //  如果栈中有2个元素构成有序序列，那么也要将它们保存下来，因为前面可能存在一个更小的元素
        //  可以和这2个元素构成3元序列。如果直接删除不记录一下，就会忽略掉这种情况。
        //  例如8 3 9 6 7这样的样例
        if(stack.size()==2) {
            //  只有2个元素的情况需要特殊判断，所以只保存较小的这一个(栈顶的)元素就可以。最小的存在min变量中
            int temp = stack.peek();
            if(!flag) {
                flag = true;
                min = temp;
            } else {
                if(temp<min) {
                min = temp;
                }
            }

        }
        //  栈顶元素出栈
        stack.pop();
    }
```

```java
            //  判断一下当前元素能否和前面存在的2元序列组成3元序列
            if(nums[i]<min) {
                System.out.println("true");
                return;
            }
            //  当前元素入栈
            stack.push(nums[i]);
        }

        System.out.println("false");
    }
}
```

1.15 移动零

题目描述：

题号：LeetCode 283

时间：9.25-9.26

给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

示例:

```
输入: [0,1,0,3,12]
输出: [1,3,12,0,0]
```

我的思路：

题目要求必须在原数组上操作，不能依赖额外的数组，也就是要求空间复杂度时 O(1)。这种在原数组上进行操作和判断的题目前面遇到过，思路大概为在遍历数组的过程中找到当前正在检测的元素“应该”放在什么位置，然后直接把那个位置的值置为当前元素。也就是说，我们利用赋值的方法将原数组重新写了一遍，从而避免了复杂的一次次移动。

这道题目要求将所有的 0 移动到末尾，其他的元素保持相对顺序不变放在数组靠前的位置。那么只需要在遍历的过程中把非零元素从数组 0 号位置开始往后放。这样遍历一遍以后，所有的非零元素都被放到前面，记录一下放到了第几号位置，剩下的部分全都用 0 填充就可以了（因为放置的过程中并没有改变 0 元素的数量）。

算法代码：

```java
public void moveZeroes(int[] nums) {
    int pos = 0;
    int l = nums.length;
    for(int i=0;i<l;++i) {
        if(nums[i]!=0) {
            nums[pos] = nums[i];
            pos++;
        }
    }
    for(int i=pos;i<l;++i) {
        nums[i] = 0;
    }

}
```

Part.2 安卓学习

2.1 安装 Android Studio

时间：9.13-9.14

记录了安装 Android Studio 过程中遇到的问题和学到的小知识小技巧。AS 我本科期间使用过，但是这次安装 3.0+版本的 AS 过程中遇到了很多超出预料的问题，困扰了我一大段时间，在崩溃的边缘游走，最终做出了一定的“妥协”终于装好了，将遇到的坑和问题保留一下。

下载

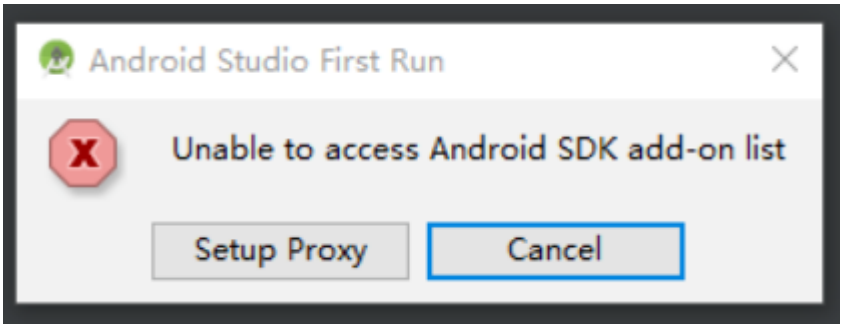
（1） 官网，目前最新版本是 3.5.0。
https://developer.android.google.cn/studio

（2） 安卓中文社区，目前提供下载的版本是 3.4.1，我选择的是这个版本。
http://www.android-studio.org/

SDK

（1） SDK 的作用：Android SDK（Software Development Kit，软件开发工具包）被软件开发工程师用于为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件的开发工具的集合。它提供了 Android API 库和开发工具构建，测试和调试应用程序。简单来讲，Android SDK 可以看做用于开发和运行 Android 应用的一个软件。每一个安卓版本会对应一个 API，需要分别下载。

（2） 安装完成后初次启动 AS，因为电脑中没有安装 sdk 的原因，会弹出这个对话框（因为安装时候一直在焦头烂额处理各种奇奇怪怪的问题，所以没有截图，下面的截图基本都来自网络）：



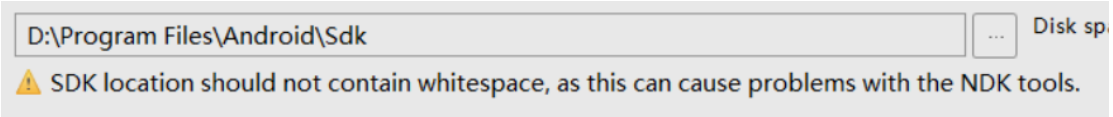
选择 Setup Proxy 可以设置代理服务器，以在国内的镜像网站上进行下载；选择 Cancel 会暂时跳过这一步，在后面进行下载。

（3） 在 AS 的安装目录下的\bin\idea.properties 文件末尾增加语句：

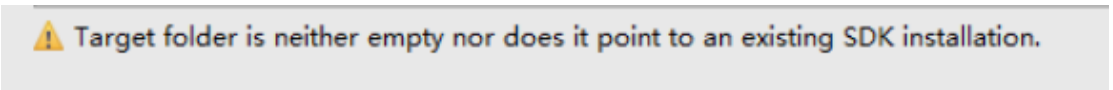
“disable.android.first.run=true”可以让 AS 在初次启动过程中不检测 SDK，但这种方法不推荐，因为进行开发是必定要安装 SDK 的。

（4） 跳过这一步后进入的界面，需要选择本地 SDK 所在的目录或者将下载的 SDK 进行保存的目录。这里存在两种小问题：

a) 路径中不能存在空格。比较容易忽视的是 Problem Files 文件夹自带空格，以及自己的用户名也可能中间有空格。这两点容易被下意识的忽略。



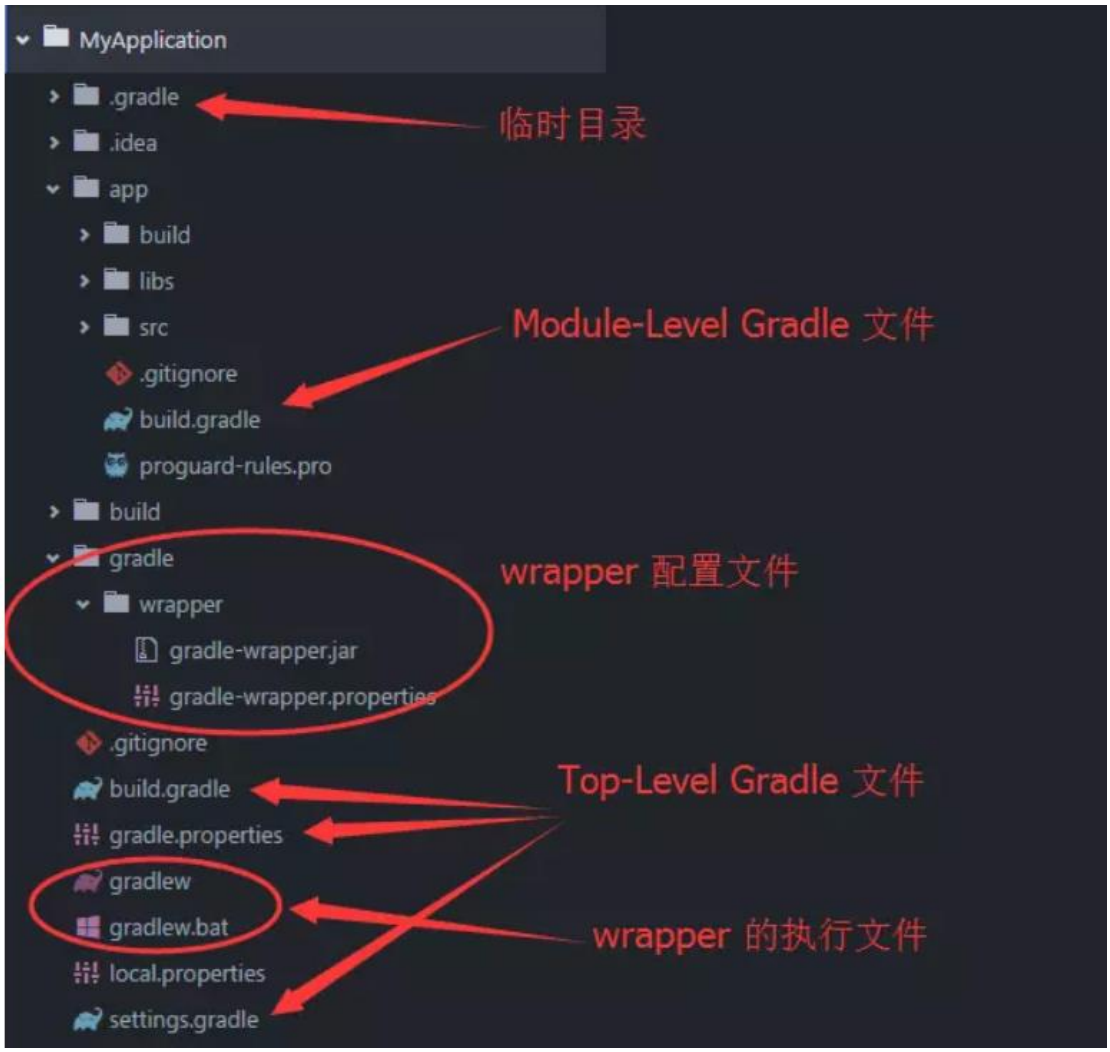
b) 目录指向的文件夹若非存放 SDK 的目录，则应该是空的。若里面存有其他的文件，也会出现提示。



（5） 这个界面上勾选下载 SDK，这里只会下载一个最新的 Android Q 版本的，其他的在启动后可以在 SDK Manager 中继续补充。我感觉校园网在下载 SDK 上好像没有遇到问题，能够顺利的下载下来。

gradle

（1） 安卓的项目是基于 gradle 构建的，它类似于 Maven，是用来给我们构建项目提供一些默认的配置和维护依赖等。它主要包括一下几个部分：



（图片来自掘金）

临时目录是这个项目（Module）需要用到的 gradle 版本，下载后存储在.gradle 文件夹中。这个文件夹存放在 C:\Users\Username\.gradle。每个项目根据它对应的安卓版本不同，需要对应下载不同的 gradle。

Top-level 的 build.gradle 用来配置所有项目的相关属性，它存在于单个项目的文件目录之外；这里面存放了 gradle 的插件版本（plugin version）。注意，gradle 插件版本和 gradle 是完全不同的两个东西，插件版本取决于 AS 这个软件的版本，在文件中记录如下：

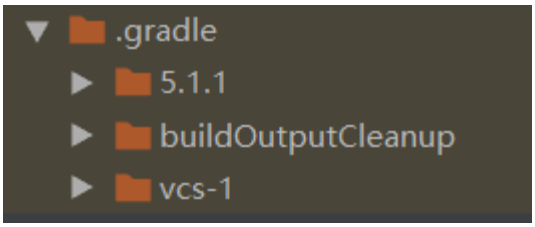
```
dependencies {
    classpath 'com.android.tools.build:gradle:3.4.1'
```

Gradle 版本指的是后面写到的 gradle-wrapper.properties 文件下记录的版本。但是插件版本和 gradle 版本有一定的对应关系，记录在官网中：https://developer.android.com/studio/releases/gradle-plugin.html#updating-gradle

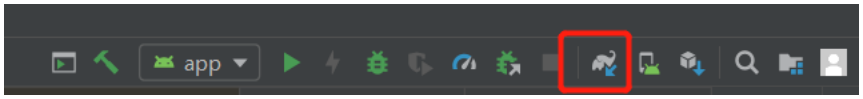
Module-level 的 build.gradle 用来配置所在项目的属性，其中包含了当前项目需要用到的一些 dependencies；

Wrapper 目录下的 gradle-wrapper.properties 文件下存放了当前项目需要用到的 gradle 的版本的下载地址，临时目录中存放的 gradle 就是符合这里规定的版本（5.1.1，因对应的安卓版本而不同）。
如果因为网络原因无法自动下载 gradle，我们可以在这个文件中找到 gradle 版本号，然后在网址：
https://services.gradle.org/distributions/
中找到对应的文件，下载后手动存放到目录：
C:\Users\Username\.gradle\wrapper\dists\gradle-5.1.1-all\97z1ksx6lirer3kbvdnh7jtjg\

```
distributionUrl=https\://services.gradle.org/distributions/gradle-5.1.1-all.zip
```



（2） 新建项目的过程中会进行 Sync Project with Gradle Files 的过程，可以点击工具栏的小图标。



这个过程困扰了我非常非常久，遇到了一系列玄学问题，我把我能记住的记在下面（虽然有些我也不知道是不是因为我的某些操作而解决的）。

（3） Could not GET ‘https://dl.google.com/dl……’ 等内容。看起来似乎是因为网络问题无法访问 google 网站等。尝试过如下方案，网上似乎都有人使用它们成功过，可以都试一试：

a) 配置代理：在 file-settings-appearance&behavior-system settings-http proxy 目录下进行配置，输入镜像的 url 和端口。这样会在 gradle.properties(top-level，非单个项目所有)文件中写入这个信息，如：

```
systemProp.https.proxyPort=80
systemProp.http.proxyHost=mirrors.neusoft.edu.cn
```

如果后面希望关闭代理，不但需要在 http proxy 中选中“no proxy”，还要将该文件中的这部分代理信息注释掉。
可用的代理：
https://my.oschina.net/u/920274/blog/3000565

b) 修改 top-level 的 build.gradle 文件。Repositories 中的内容表示需

要在什么地方寻找需要的依赖，例如 google()，jcenter()。

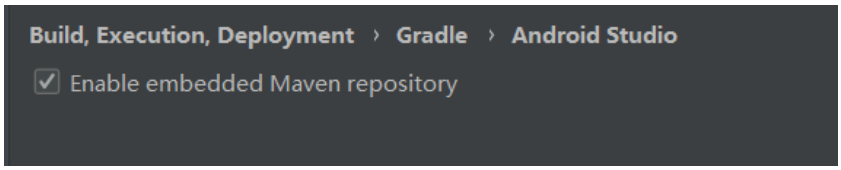
```
buildscript {
    repositories {
        google()
        jcenter()
        maven { url 'http://maven.aliyun.com/nexus/content/groups/public/' }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
        maven { url 'http://maven.aliyun.com/nexus/content/groups/public/' }
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

这些地址可能无法访问，我最终成功的方案是增加了 maven 这一行，它后面标记的 url 是利用了阿里云提供的目录来下载相关的依赖。如果要增加 maven 项，需要在 File | Settings | Build, Execution, Deployment | Gradle | Android Studio 目录下勾选如下选项，表示允许项目中嵌入 maven 库。



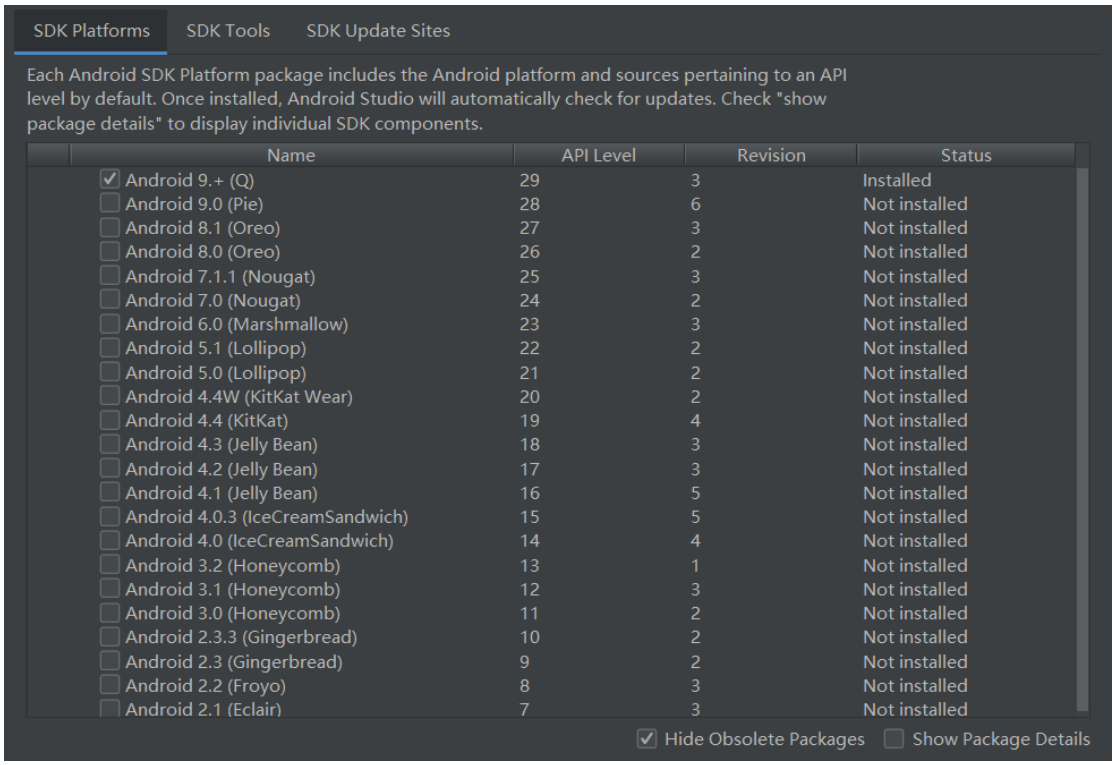
（4） Could not download junit.jar (junit:junit:4.12)。这个问题我一直没找到很好的办法彻底解决。经查资料，junit.jar 是一个用于 test 相关的库，所以我暂时先关闭了这个功能，方法为，找到 module-level 的 build.gradle 文件，将要求的关于 test 和 junit 的依赖部分注释掉（后面三行）：

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
//  testImplementation 'junit:junit:4.12'
//  androidTestImplementation 'androidx.test:runner:1.2.0'
//  androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

（5） Cause: unable to find valid certification path to requested target。这个问题原本没出现，但是因为我之前总是出现超时、同步失败、无法访问某些 url 来下载文件等问题，我认为是网络问题，于是设置了代理，后来在不断的调试中把代理改来改去，删掉又增加等等，结果突然某一刻出现了这个报错。我猜测是因为删掉了什么内容导致无法访问 https 网站等，缺少有效证书。网上给出的方法包括导入需要的证书等，我最后也没有解决这个问题。最终，通过卸载软件重装恢复到初始状态来绕开了这个问题。注意，重装的时候如果重装相同版本的 AS，会保留之前的设置，不会恢复初始状态；只有更换一个版本的 AS，才能在安装中选择删掉另一个版本遗留的配置信息。

（6） 所有提示 Gradle project sync failed 的问题，在关闭 AS 重启后不会出现，但这时候事实上还是没有同步成功，只不过需要你手动点一下 sync 按钮再次开始这个任务才会出现，而刚创建新项目的时候是会自动 sync 的。但显然不可能瞒天过海，在没有 sync 成功的情况下，所有的控件（如 textview 等）都会标红报错，图形化界面也不会显示。我之前一度以为 sync 经过我的重启成功了，而这些文本框按钮等出现了奇异的问题，我非常苦恼为什么系统自动提供的代码会报错（……）。只有 gradle 构建成功，才可以开始开发。

SDK Manager



在 SDK Manager 中存在非常多令人眼花缭乱的下载列表。SDK Plagforms 中为各个不同的安卓版本对应的 API，如果你需要开发某个安卓版本适配的 app 就需要下载

对应的 SDK；SDK Tools 是一些通用的开发和调试工具。具体需要下载的内容在这篇博客中讲解的非常详细：
https://www.cnblogs.com/tsingke/p/9054044.html

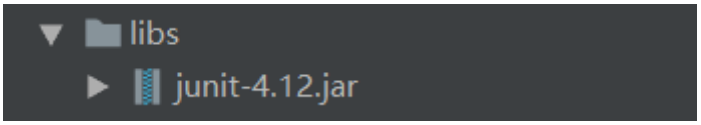
彻底解决无法导入库的问题

时间：9.21

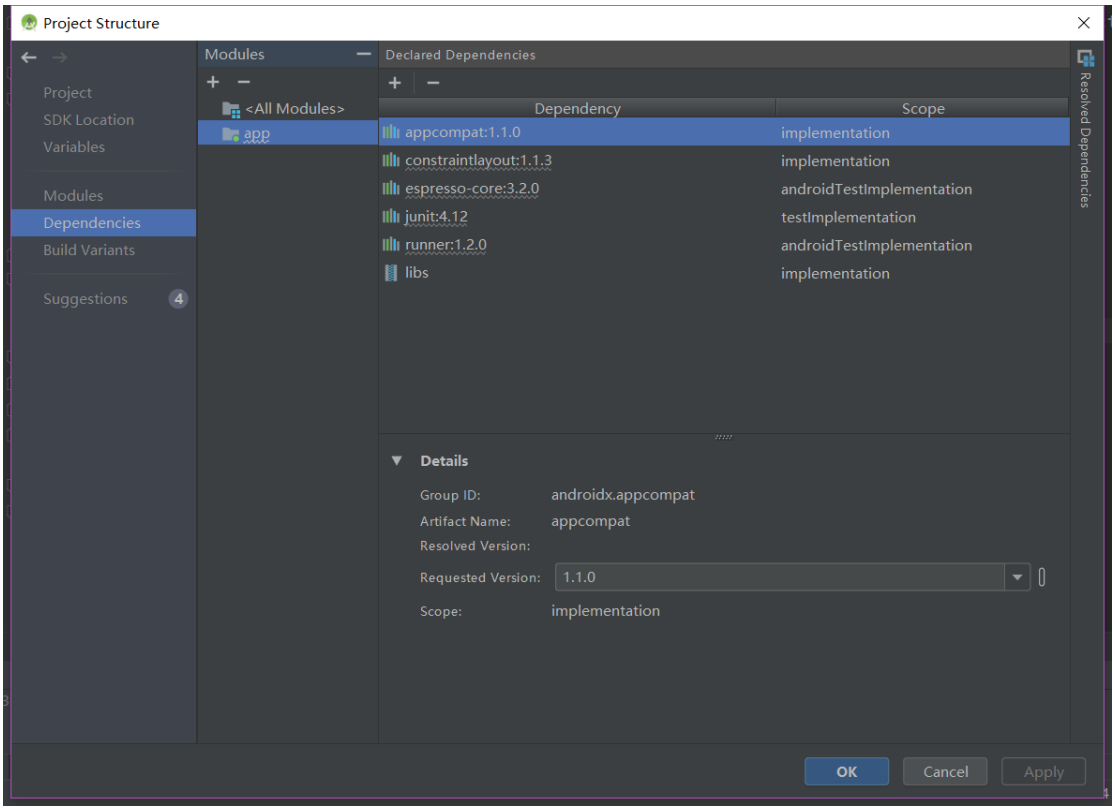
发现在注释掉请求 test 相关的库后，虽然项目的 build 不会出现问题了，但是后来开始项目 run 起来调试的时候，还是会因为项目中存在初始化时就自动生成的 test 和 Androidtest 相关的类，它们会导入 junit，而注释会导致它们报错。

事实上，junit 4-12 这个 jar 包确实存在在了相关 gradle 文件夹下，但不知道为什么无法检索到它们，总是请求 download。

我也试着将它手动 copy 到了 libs 文件夹下。依然没有效果。



点击 project structure 可以查看项目依赖库的情况：



App 栏写的是 app 级 build.gradle 文件中添加的那些依赖。这里和文件内容是一一对应的。在下面 details 那一栏可以修改要求的版本号。

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation "junit:junit:${var}"
    androidTestImplementation 'androidx.test:runner:1.2.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

}
```

（这里不知道啥时候直接写的 4.12 版本号成了变量形式表示，这个变量存在 top level 的 build.gradle 文件中。

仍然没有什么发现。尝试用中文搜了一下问题!!!
之前都是照搬报错信息原文 could not download junit 4-12.jar 云云
我修改成了 AS 无法下载 junit

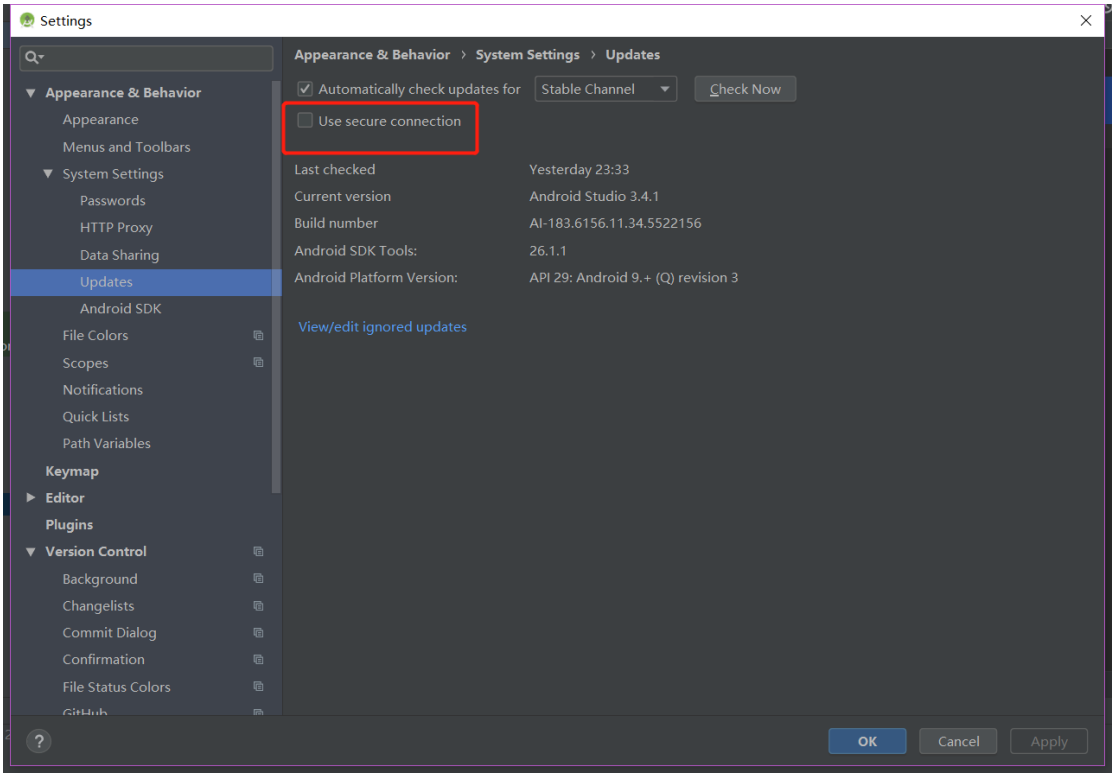
然后找到一个答案。
先是说 jcenter 这个库可能无法访问，建议在它的前面增加 maven 库：

```
buildscript {
    repositories {
        maven { url 'http://maven.aliyun.com/nexus/content/groups/public/' }
        google()
        mavenCentral()
        jcenter()
```

```
allprojects {
    repositories {
        google()
        mavenCentral()
        jcenter()
        maven { url 'http://maven.aliyun.com/nexus/content/groups/public/' }
    }
}
```

这里指这个 mavenCentral，下面那个阿里云的 maven 是我之前就添加过的。我也不知道这两个有没有冲突或冗余；
但其实我觉得这一步不是最重要的。重要的是!!!

File | Settings | Appearance & Behavior | System Settings | Updates
这个路径下，有一个选项！



将它取消勾选。

这个类似的功能我已经需求了很久了。因为之前我就一直在怀疑是不是 https 类型地址的问题。总之进行完这两步，终于可以了!!!!

2.2 约束性布局 ConstraintLayout

时间：

最近的移动软件开发课程开始学习开发安卓 APP 项目。大学期间也学习过相关的内容，但是那时候初学编程，很多知识和方法都不求甚解，完全是面向“成果”在编程，也就是只要能做出想要的效果、实现想要的功能（甚至可以退而求其次，或者省略一些做不出来的功能）就好，对整体的理解设计、代码的整洁质量、方法的原理内涵等都忽略了。现在想来，这些才是更重要的更值得学习的东西。这一次希望借着这门课程的机会，将相关的知识好好梳理学习一遍。

从布局开始。选择了学习 ConstraintLayout 约束性布局。

为什么选择它

如果使用线性布局，不可避免要用到若干线性布局的嵌套。如果使用相对布局，它在生成的时候需要测算两次。这些特点会影响到性能。少的或者没有嵌套称为布局的扁平化，而约束性布局能实现布局最大程度的扁平化，性能更好。

安卓开发无法绕开的一个问题是屏幕的适配性问题。这是由于安卓系统应用范围广

泛且丰富，不同手机厂商的各种机型规格变化太多。如果不考虑适配性问题，设计出来的界面可能只是在调试时和你的模拟器相匹配。

实现屏幕适配方法也很多，例如用可以根据屏幕的像素密度在代码中等比例转换的 dp 作为尺寸的单位，可以编写多套 values 文件夹、类似于方法重载效果的 dimens 适配，百分比布局适配，针对不同的分辨率分别编写多套布局适配等。

使用约束性布局也是一种适配方式,而且是 Google 官方提倡的布局方式。

私心：看了一下网上的介绍和教程，我发现这种布局方式是一种很严谨很规则的方式，它可以严格对每个控件做到控制其四个方向分别所延申到的位置。对我来说这种布局方式非常优雅。虽然有些繁琐，但强迫症的我喜欢这种规则性，正如我喜欢 Java。

依赖和 app 命名空间声明

因为这是 AndroidStudio 当前版本（以及自该布局推出后的所有版本）默认的布局方式，所以在 app/build.gradle 文件内已经添加了对它的依赖：

```
implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
```

在应用该布局的布局文件中要添加这样一条语句。可以看到它的结构类似于创建该文件时自动生成的“xmls:android= ……”那一句，其实这个句子是在声明命名空间，我理解的就是类似于 c++程序中通过声明 using namespace std 之后可以用 std 命名空间内的一些方法一样。默认声明了 android 这个命名空间，所以我们平时经常用到的那些诸如“android:id=……”、“android:text=……”这些规定控件的相关属性的句子，都是因为声明了 android 命名空间之后，才可以写这些以“android:”开头的语句。

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/bg_demo"
    android:theme="@android:style/Theme.NoTitleBar"
    >
```

这个布局中有些属性的规定语句是类似于“app:layout_constraintGuide_percent = ……”这样规定的，所以举一反三，它们要用“app:”开头，也就是用到了 app 这个命名空间。

例如下图，这个 TextView 空间中就用到了很多 app 语句来对它进行约束。

```
<TextView
    android:id="@+id/tv_demo_weekday"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:text="FRI"
    android:textSize="20dp"
    android:gravity="center"
    app:layout_constraintTop_toBottomOf="@+id/hgl_0.05"
    app:layout_constraintBottom_toTopOf="@+id/hgl_0.15"
    app:layout_constraintLeft_toRightOf="@id/tv_demo_locaiton"
    app:layout_constraintRight_toLeftOf="@id/vgl_0.95"
    app:layout_constraintHorizontal_weight="1"
    />
```

布局思路

我对这个布局也并没有非常系统和完整的了解。就我目前用到的部分，我理解的这个布局如同它的名字描述的那样，通过对布局内的控件加以严格的“约束”来实现定位。严格到什么程度呢，以一个文本框为例，我们可以规定它的四个方向分别相对于边框或其他控件的距离，来控制这个控件的位置。

“就像是四条橡皮筋将里面的控件紧紧的勒住了”。我们把四条橡皮筋写好，里面的控件自然就定位了。

参考线 GuideLine

约束性布局设计了一种新控件来充当橡皮筋的角色，就是参考线。参考线的宽高不用设置（可以写出 0dp，无实际意义），因为它不会在布局中显示出来。它的作用相当于在布局中绘制了一些无形的线，就像相对布局中可以用四个边框作为参照物那样，为了在布局内部提供更多更精细更严谨的参照物，我们在布局里根据需要绘制若干条横向（horizontal）或纵向（vertical）的线，来辅助定位，帮助控件找到更多可以作为“相对于某某在左边”的辅助工具。

下图中，两个红框中的内容表示了线的方向；蓝框中的内容用百分比的方式表示这条线的位置，例如这两个参考线分别是距离上边框 70%位置的横线，和距离左边框 5%位置的竖线。

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/hgl_0.70"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:orientation="horizontal"
    app:layout_constraintGuide_percent="0.70" />

<androidx.constraintlayout.widget.Guideline
    android:id="@+id/vgl_0.05"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:orientation="vertical"
    app:layout_constraintGuide_percent="0.05"
    />
```

如果我们用两横两纵四条参考线将控件“圈起来”，那么里面的控件的宽高也不需要设置了，它会默认匹配这四条参考线约束起来的位置。

由于不同手机的宽高比可能不同，例如 720*1280 和 1080*1920 的宽高比是相同的，但是和 320*480 的相比略有不同。这样，四条橡皮筋可能会产生一定的变形（例如在前者的比例中计算出的基准线位置，应用到后者中，会有略微的不适配）。

解决方法为：只设置三条橡皮筋（例如去掉下面的），然后设置控件的宽高比：

```
<ImageView
    android:id="@+id/iv_demo_weather"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:src="@drawable/ic_weather_sunny"
    app:layout_constraintTop_toBottomOf="@id/hgl_0.20"
    app:layout_constraintBottom_toTopOf="@id/hgl_0.40"
    app:layout_constraintDimensionRatio="1:1"
    app:layout_constraintLeft_toLeftOf="@id/vgl_0.05" />

<TextView
    android:id="@+id/tv_demo_temperature_realtime"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:text="27°"
    android:textSize="50dp"
    android:gravity="center"
    app:layout_constraintTop_toBottomOf="@id/hgl_0.20"
    app:layout_constraintBottom_toTopOf="@id/hgl_0.30"
    app:layout_constraintRight_toLeftOf="@id/vgl_0.95"
    app:layout_constraintDimensionRatio="h,1:2"
    />
```

由于宽度是按照屏幕百分比计算出来的，而高度又是根据宽高比计算出来的，所以可以说高度也是按百分比计算的，这样就用约束者布局实现了屏幕适配了。

设置约束

借用一张网图。它的约束语句非常多，但是四个方向的思路都是类似的可以举一反三，所以也很好理解，不需要刻意记忆：

下面来看看相对定位的常用属性：

layout_constraintLeft_toLeftOf

layout_constraintLeft_toRightOf

layout_constraintRight_toLeftOf

layout_constraintRight_toRightOf

layout_constraintTop_toTopOf

layout_constraintTop_toBottomOf

layout_constraintBottom_toTopOf

layout_constraintBottom_toBottomOf

layout_constraintBaseline_toBaselineOf

layout_constraintStart_toEndOf

layout_constraintStart_toStartOf

layout_constraintEnd_toStartOf

layout_constraintEnd_toEndOf

好的方法可以通过名字看出来它的作用。这些属性就是规定了控件的左边是相对于某某的左边/左边是相对于某某的右边/右边是相对于某某的左边……以此类推，顾名思义。这里的参照物可以是边框（parent，父布局），也可以我们设置好的参考线。用下图来举例：

这个记录日期的文本框，它的顶部 top 被一条横向的基准线约束（距离上边框 5%），下部 bottom 被一条横向的基准线约束（距离上边框 15%），左边 left 被一条纵向的基准线约束（距离左边框 5%），右边被另一个控件 tv_demo_location 约束。

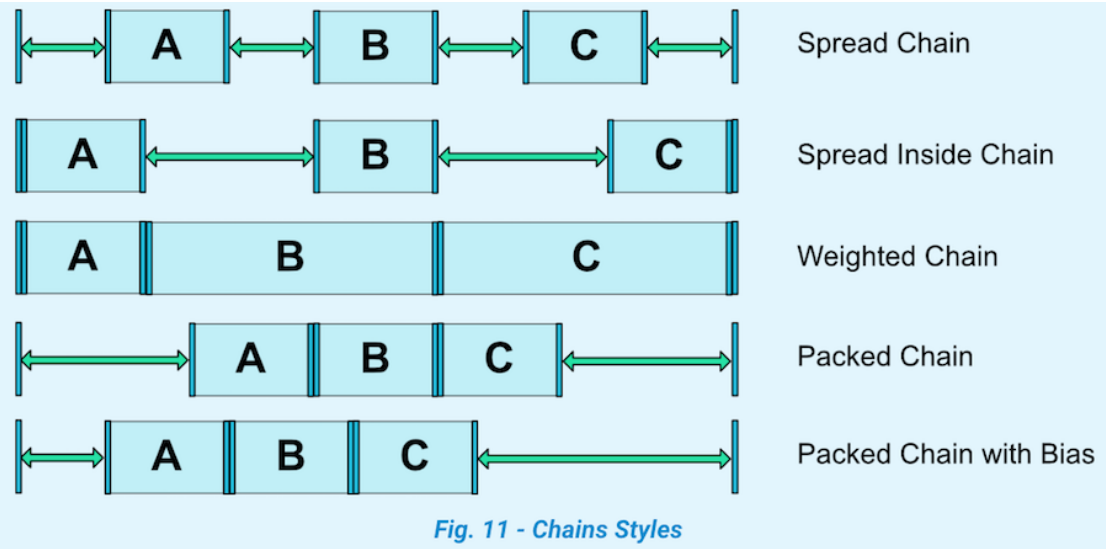
```
<TextView
    android:id="@+id/tv_demo_date"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:text="09/27\n2019"
    android:textSize="20dp"
    android:gravity="center"
    app:layout_constraintTop_toBottomOf="@+id/hgl_0.05"
    app:layout_constraintBottom_toTopOf="@+id/hgl_0.15"
    app:layout_constraintLeft_toLeftOf="@id/vgl_0.05"
    app:layout_constraintRight_toLeftOf="@+id/tv_demo_location"
    app:layout_constraintHorizontal_weight="1"
    />
```

链式约束

多个控件在同一条水平线或竖直线上是常见的现象。在横轴或竖轴上的控件相互约束的时候，可以组成一个链式约束。

layout_constraintVertical_chainStyle

利用这个语句可以对一整条链设置布局，有不同的形式，如下。链条的特性在第一个 view 上设置。借用一个网图来说明不同的链式约束。只需要声明响应的参数即可：



参考资料:

https://blog.csdn.net/qdyzhenshitaishuaile/article/details/76696193

https://www.jianshu.com/p/17ec9bd6ca8a

https://www.cnblogs.com/angrycode/p/9739513.html

https://www.jianshu.com/p/f387687b1826

2.3 安卓代码命名规范

时间：9.29-10.8

安卓项目的代码中存在很多需要命名的地方，并且很多命名都需要一定的对应性，例如布局文件中各种控件的命名，对应的 java 类中的 UI 控件的命名，java 类中本身包含的各种变量的命名等。我觉得首先了解基本的命名规范是必要的，因为这可以避免写代码时候可能产生的混淆和杂乱。不过命名规范应该并非完全统一，我这里只写出我参考的这份资料所推荐的命名方法。

布局文件 layout：

R.layout.类型名_功能名

其中，R.layout 是自带的前缀（例如在 java 文件中查找 layout 时需要加上这个前缀），在 layout 命名时只需要后面的部分。

类型名就是应用这个布局的模块的类型，例如常用的整个界面是 activity，或其中的部分 fragment 碎片，用于对话框中的布局 dialog，还有 adapter、widget、item 等。也就是说我们可以为那些部分都单独写一个布局文件，只需要在命名的时候体现出来它们的类型。

功能名是这个布局中的内容，例如登录界面命名成 login，注册界面命名成 register 等，易于理解就好。那么，一个用来登录的活动界面就应该命名成 activity_login。

与安卓组件相对应的 Java 类的命名：

模块/功能名_类型名

上面的这些布局文件都需要有一个对应的 Java 类对它们进行操作，这个类名和它所对应的布局文件名一一对应就好，模块和功能名就对应上面的功能名（login 等），类型名就是各种组件的类型。例如 activity_login 对应的 Java 类要命名成 LoginActivity。

还有一些组件不属于布局文件，但是也遵循相似的命名规范，例如广播接收器 BroadcastReceiver，服务 Service，也根据功能命名成 LoginReceiver，LoginService 等。

布局中的控件名：

控件缩写_所在模块_功能名

因为控件的类型都较长，所以有一些约定的缩写，例如 tv（TextView），et（EditText），btn（Button）等；所在模块给的例子是 user 用户模块等，我理解的是用于区分不同模块或界面中可能具有的相同内容，但是不同的界面中的内容一般不会同时拿过来用，所以可以根据需要来填写或省略；功能名就是这个控件的作用，例如显示温度 temperature，显示风力 wind 等等。一个显示天气的文本框应该命名为 tv_weather。我在用的时候加上了所在的界面名称，写成了 tv_demo_weather。暂时还不确定中间的这部分是否有存在必要，还要在后续的实践过程中去积累经验。此外要注意，界面中的布局类型（线性布局，相对布局等）也是控件的一种，缩写成 ll，rl 等。

Java 类中对应的 UI 控件变量：

作用域_功能_控件缩写

作用域指变量的公有/非公有以及静态/非静态的属性，分别用 g,p,s,m 来开头，因为目前还没有涉及到考虑这些作用域类型的细致地步，所以暂时先没有考虑这个部分；功能名和控件缩写就和它对应的布局中的控件的功能命名相同就好，例如前面距离的那个 tv_weather 这里就命名成 weatherTv（如果考虑作用域，例如非公有非静态以 m 开头，就应该命名成 mWeatherTv）。在不考虑作用域的情况下，我还看到一种命名规范，就是用控件来开头，写成 tvWeather，这样让控件的缩写都是小写，更容易辨别一些。

图片资源 drawable 命名：

我之前尝试在 drawable 文件夹下新建不同的文件夹来对图片资源进行分类，但这样貌似是不被许可的，可能是因为在引用 drawable 中资源的时候只有一个@drawable 来

进行标识。那么就只能在图片命名的最前面对类型进行划分，来方便区分。

目前只用到了普通图标图片，命名为：

ic_具体模块_功能名

以我代码中的 68 种天气的图标为例，它们被命名为 ic_weather_100，后面的编号代表着天气的类型。

其他的图片类型还有工具栏图标，样式选择器，样式形状等，这些等到用到的时候再去学习。

包命名：

com_个人/公司_应用名_功能模块名

com 是固定的，中间两个因人因应用而异，主要是最后一个功能模块名需要注意。例如我们常用的活动、服务、广播等安卓组件，它们对应的 Java 类都要被分门别类的放到 activity、service、receiver 包中，其他的模块就是和 Java 共通的，如数据库相关操作的 db 包，公共工具方法的 util 包，JavaBean 数据类的 bean 包等。

2.4 编写爬虫获得天气图标

时间：9.29-10.8

开始在网上随机找了一些天气图标，发现它们一般都不够全，每一套一般都只有十个左右的内容，这样不但要把复杂的各种天气类型多对一的进行映射，还会导致图标显示的信息不准确不完整。之后发现提供天气信息的和风天气接口中本身就带有各种天气类型所对应的图标，但是较多（共 68 个），所以考虑写一个简单的爬虫将它们下载下来。

目标网址：https://dev.heweather.com/docs/refer/condition

使用之前学习过的 selenium 库来连接 url。事实上这个库单纯的用来连接 url 有些大材小用了。分析源代码发现所有的图标都存放在<tr>标签中：

```
▼<tr>
    <td>100</td>
    <td>晴</td>
    <td>Sunny/Clear</td>
  ▼<td>
      <a href="https://cdn.heweather.com/cond_icon/100.png" title="晴天图标">100.png</a>
    </td>
  </tr>
```

所以我们利用 find_elements_by_tag_name 的方法来找到所有的 tr 标签。这里用的是 find_elements 而非 find_element，是因为我们这里要查找的不是某个精确的 tr，而是所有的 tr。用 find_elements 能给我们返回一个列表 data。

还是看上面的截图，我们得到列表 data 之后，遍历其中的每一个内容，就是在依次访问每一个 tr 标签了。

分析每一个 tr 标签对中的内容来获取我们想要的部分。可以看到 tr 标签内有唯一的 a 标签，其中的 href 属性是我们所需要的下载图片的地址，a 标签的文本内容 text 是我们可以用来对图片进行命名的信息（因为这个编号和天气 API 返回的天气代码一一对应，便于根据返回信息去查找对应的图片资源文件）。那么再用一次 find_element_by_tag_name（注意

这里是 element 而非 elements），分别用 get_attribute 方法找到 href 中的内容，和 text 属性找到文本内容。

文本内容用来构成图片的名称。根据前面写到的命名规范，我们直接在下载和存储图片的时候就按照规范命名好 ic_weather_{text}。

打开 href 中的 url，读取其中的内容，用 python 的打开文件方法新建一个如上命名的文件，然后将 url 中读取的内容写入。

```
for eachdata in data:
    # 读取图片的下载地址
    imgurl = eachdata.find_element_by_tag_name("a").get_attribute("href")
    imgname = eachdata.find_element_by_tag_name("a").text
    name = "ic_weather_{}".format(imgname)
    img = url_open(imgurl)
    with open(name, 'wb') as f:
        f.write(img)
```

补充一个遇到的小问题。除了包含所有图标的这些 tr 标签，在最前面还有一个包含其他信息的 tr 标签，这个在列表 data 中会干扰到其他我们需要的部分。利用 del 方法来将这个元素删除即可。

2.5 获取天气信息和更新界面

时间：9.29-10.8

开始在网上随机找了一些天气图标，发现它们一般都不够全，每一套一般都只有十个左右的内容，这样不但要把复杂的各种天气类型多对一的进行映射，还会导致图标显示的信息不准确不完整。之后发现提供天气信息的和风天气接口中本身就带有各种天气类型所对应的图标，但是较多（共 68 个），所以考虑写一个简单的爬虫将它们下载下来。

目标网址：https://dev.heweather.com/docs/refer/condition

使用之前学习过的 selenium 库来连接 url。事实上这个库单纯的用来连接 url 有些大材小用了。分析源代码发现所有的图标都存放在<tr>标签中：

```
▼<tr>
    <td>100</td>
    <td>晴</td>
    <td>Sunny/Clear</td>
  ▼<td>
      <a href="https://cdn.heweather.com/cond_icon/100.png" title="晴天图标">100.png</a>
    </td>
  </tr>
```

所以我们利用 find_elements_by_tag_name 的方法来找到所有的 tr 标签。这里用的是 find_elements 而非 find_element，是因为我们这里要查找的不是某个精确的 tr，而是所有的 tr。用 find_elements 能给我们返回一个列表 data。

还是看上面的截图，我们得到列表 data 之后，遍历其中的每一个内容，就是在依次访问每一个 tr 标签了。

分析每一个 tr 标签对中的内容来获取我们想要的部分。可以看到 tr 标签内有唯一的 a 标签，其中的 href 属性是我们所需要的下载图片的地址，a 标签的文本内容 text 是我们可以用来对图片进行命名的信息（因为这个编号和天气 API 返回的天气代码一一对应，便于根据返回信息去查找对应的图片资源文件）。那么再用一次 find_element_by_tag_name（注意这里是 element 而非 elements），分别用 get_attribute 方法找到 href 中的内容，和 text 属性找到文本内容。

文本内容用来构成图片的名称。根据前面写到的命名规范，我们直接在下载和存储图片的时候就按照规范命名好 ic_weather_{text}。

打开 href 中的 url，读取其中的内容，用 python 的打开文件方法新建一个如上命名的文件，然后将 url 中读取的内容写入。

```
for eachdata in data:
    # 读取图片的下载地址
    imgurl = eachdata.find_element_by_tag_name("a").get_attribute("href")
    imgname = eachdata.find_element_by_tag_name("a").text
    name = "ic_weather_{}".format(imgname)
    img = url_open(imgurl)
    with open(name, 'wb') as f:
        f.write(img)
```

补充一个遇到的小问题。除了包含所有图标的这些 tr 标签，在最前面还有一个包含其他信息的 tr 标签，这个在列表 data 中会干扰到其他我们需要的部分。利用 del 方法来将这个元素删除即可。

2.6 Service 的学习和使用

时间：9.29-10.8

安卓应用在后台运行时执行的内容就是服务 Service，它不需要和用户交互，不依赖于用户界面，并且确保能在后台长期运行。服务依赖于创建服务时所在的进程。只有这个创建它的进程终止时，依赖于这个进程的服务才停止。

为了防止服务中运行的程序阻塞主线程，最好在服务的内部创建子线程，把需要服务完成的工作在子线程中运行，这样就不会影响主线程的正常工作。

创建线程的方法：

和 Java 中相同，可以创建一个继承 Thread 的自己的线程类，并重写 run 方法；也可以让自己的线程类实现 Runnable 接口，并重写 run 方法。最简单和常用的方法是使用匿名类：

```
new Thread(new Runnable() {
    @Override
    public void run() {

    }
}).start();
```

应用 service 来实现后台更新数据：

通过网上查阅资料，我了解到一种思路，利用服务和广播的机制，设置一个后台服务，每隔一段时间就发送一个广播，然后在天气显示界面的活动中接收这个广播，每当接收到广播后，就执行一次更新天气的方法。

首先创建一个服务，在服务中新建一个定时器，按时发送广播。这里设置的时间是 30 分钟。发送服务还是要使用 Intent 对象，可以说是神通广大了。由于发送的是全局广播，所以要在发送的时候设置一个 action 作为标志，以让希望接收到这个广播的活动能够通过 action 来找到它，类似于通过隐式 Intent 启动活动的方式。

```
//  创建一个定时器
new Timer().scheduleAtFixedRate(new TimerTask() {
    @Override
    public void run() {
        //  发送广播仍然使用Intent对象
        Intent timeIntent = new Intent();
        //  设置动作名称，用于接收
        timeIntent.setAction("TIME_CHANGED_ACTION");
        sendBroadcast(timeIntent);
    }
    //  设置时间间隔
}, delay: 0, period: 30*60*1000);
```

在主活动中创建一个广播接收器子类，继承自系统的广播接收器。这里重写的 onReceive 方法就是表示我们这个接收器在接收到广播后执行的动作。把更新实况天气的方法放在这里面，以确保每一次接收到广播的时候都调用它。广播是每隔 30 分钟发送一次的，那么我们这里每隔 30 分钟接收到一次并执行理论上就能够实现要求。

```
public class MyTimeBroadCast extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {
        requestWeatherNowBySDK(DAXING_ID);
    }
}
```

接下来还需要创建一个 MyTimeBroadCast 类的实例用来接收广播。同时，回想在发送广播的时候附带了 action 信息，我们还没有写在哪里读取这个信息，也就是找到这个特定的广播。方法是创建一个 IntentFilter 类的实例，顾名思义这是一个起到过滤作用的类，同样通过给它添加 action 的内容，将参数设置成和发送广播的 action 参数

相同，就能达到我们的目的了。最后，将这个过滤器和广播接收器对象通过注册接收器方法绑定在一起。

```
myTimeBroadCast = new MyTimeBroadCast();
IntentFilter filter1 = new IntentFilter();
filter1.addAction("TIME_CHANGED_ACTION");
registerReceiver(myTimeBroadCast, filter1);
```

2.7 信息在活动间的传递

时间：9.29-10.8

直接在包的三级目录下创建新的 activity（和 MainActivity 相同的目录），注意每个活动和服务都要在配置文件 AndroidManifest.xml 下注册。这里 IDE 会自动给我们配置好。

传递信息的组件：Intent。可以用于启动活动，启动服务，发送广播。
暂时只关注它用来启动服务的功能。它有显式和隐式两种方式。

显式：
通过 startActivity 来启动服务，利用 Intent 的对象来注册启动活动的上下文（当前活动）和被启动的活动名称。这样就建立了一个对应的关系。相当于 Intent 对象的起点和终点都被确定了。注意这个对应不是一一对应的，因为在当前活动中显然可以通过定义不同的 Intent 对象来启动不同的服务。

```
Intent intent = new Intent( packageContext: MainActivity.this, TestActivity.class);
intent.putExtra( name: "count", data);
startActivity(intent);
```

隐式：
通过在配置文件中给不同的活动添加它们可以响应的不同的 action 和 category 属性，来赋给这个活动一种“特征”。之前有一个问题就是为什么一开始创建的 MainActivity 就是 app 启动时候的默认活动，这时候这个问题得到了解答。查看 AndroidManifest.xml 文件中这个活动的相关信息：

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

这里，系统在创建这个活动的时候就添加了它的 action 和 category 参数。这两个参数的内容决定了要在 app 启动的时候启动这个活动，就相当于有一个我们看不到的、app 自动创建的 Intent 对象根据 action 和 category 的内容来自动启动了这个活动，如同我们在活动中启动其他的活动一样。

隐式的 Intent 方法就是这样。在创建 Intent 对象的时候，构造方法传递的参数不再是“起点和终点”，而是特定的 action 和 category 的值。这时候不通过起点和终点来找到目标，而是系统根据传递的 action 和 category 来找到符合要求的活动。如果找不到符合要求的活动，app 就可能会崩溃。

隐式Intent能够启动其他程序的活动，让多个应用程序之间的功能共享成为可能。例如添加 action 为 Intent.ACITON_VIEW 可以用来启动浏览器。这是一个系统内置的属性内容，相当于启动了系统给我们写好的一个活动（也就是浏览器）：

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.baidu.com"));
startActivity(intent);
```

可以看到，之前填写目标启动活动的参数位置改成了这个 action 的内容，然后传递一个 url 地址作为 data 值，就能够在启动的浏览器中打开百度的网址了。

传递数据：

我们创建 Intent 对象的时候可以随着这个对象捎带一些信息进行传递，方法就是利用 putExtra 方法，传递的信息通过键值对的方式指定，一个 key 用来标志这个信息，一个 value 表示这个键所存储的信息的内容，类似于 SharedPreferences 持久化存储信息的方式。在被启动的活动中，通过 getStringExtra（以及其他）等方法来获得相应类型的数据。

```
Intent intent = getIntent();
String we = intent.getStringExtra( name: "count");
```

其他一些有用的内容：

StartActivityForResult

在启动活动的时候就指定了希望被启动的活动在销毁的时候能够返回数据，因为我们前面创建 Intent 对象的时候传入了“起点和终点”，所以我们可以顺着这条“路径”找到要启动的活动，也能在我们需要从被启动的活动返回的时候准确找到之前的活动。所以说想要在返回活动的时候“捎带”一些信息回来也是很自然的事情。只需要把之前启动活动的方法 startActivity 改成 startActivityForResult 就可以了，同时要多传入一个参数作为请求码，也就是一个标志。

```
Intent intent = new Intent( packageContext: MainActivity.this, TestActivity.class);
intent.putExtra( name: "count", data);
  startActivity(intent);

startActivityForResult(intent,  requestCode: 1);
```

在被启动的活动中，也要设置一下与 result 对应的内容。由于要返回信息，所以同样是通过创建 Intent 对象，借助这个对象来传递信息，只不过这一次我们不需要用到它的启动活动的功能了。然后在为它设置一个响应码，也是帮助接收方确认和识别这条信息：

```
Intent intent2 = new Intent( packageContext: TestActivity.this, MainActivity.class);
intent2.putExtra( name: "data_return",  value: "from btn for main");
setResult(RESULT_OK, intent2);
finish();
```

根据前面的经验，我们有了用来标识的请求码和响应码，一定还有一个地方要用来识别这些内容，这部分代码要写在返回的活动中。通过 switch 来选择需要响应的请求码，在每一个请求码下通过 if 语句再来找到期望的响应码：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode){
        case 1:
            if(resultCode==RESULT_OK){
                String returnedData = data.getStringExtra( name: "data_return");
                Log.i(TAG,  msg: "onActivityResult: "+returnedData);
                testTv.setText(returnedData);
            }
            break;
        default:
    }
}
```


参考资料:

Android：你不能忽略的代码命名规范 https://www.jianshu.com/p/b7a644ea0d25

Android 代码规范-命名规范 https://www.jianshu.com/p/97187278a5fb

和风天气 API 文档 https://dev.heweather.com/docs/getting-started/

郭霖《第一行代码》第二版

Part.3 开源讲座报告

3.1 9.20 日讲座

目录

- 开源的概念和意义
  - 什么是开源
  - 开源的十条准则
  - 开源对学习的帮助
- 版本控制
  - 必要性
  - 单人更迭版本的控制思路
  - 多人合作时的版本控制的发展
  - 多人合作中需要考虑的问题
- 最好的版本管理工具：Git
  - 特点
  - 基本操作
- 开源社区
  - 开源社区在哪里
  - 代码规范
  - 如何加入开源社区
- 开源项目
  - 如何找到开源项目
  - 适合初学者的开源项目
- 在讲座中学到的其他内容

开源的概念和意义

- 什么是开源

  从名字本身来看，开源首先是开放源代码，让整个社区可以互相学习、开放协作。从它最明显的作用上看，开源让我们可以借用他人造好的轮子，让自己的代码和项目有更好的起点和更多可利用的工具。对自己代码的满足感和渴望与他人分享并得到认可的成就感是开源背后的心理学效应，严密有效的法律条款保障了开源社区的权益。开源源自社区，源自普通的独立的编程爱好者，但星

星之火可以燎原。最终，开源社区让企业的态度从负面逐步转变到正面，最终认可和接受了开源，并成为开源社区的贡献者。

- 开源的十条准则

  1.Free Redistribution license 不能限制任何一方将软件作为一个另一个集成了多来源程序的软件的一部分进行出售或赠予，亦不能对使用该开源软件要求付费。

  2.Source Code 开源项目必须包含源代码，并允许代码形式和编译形式的开放。其他和该项目有关的产品也应该随源代码一起公开，或以免费或低成本方式让用户获得。不得故意混淆源代码，不得给出代码的中间形式（如预处理器或比翻译器的输出），源代码应该是提供给大家的首选格式。

  3.Derived Works license 必须允许修改和派生项目，并允许它们开源在相同的 license 条款下。

  4.Integrity of The Author's Source Code 一般要限制修改形式的源代码，正常要通过补丁文件的方式来开放修改后的源代码。衍生的项目可能会需要使用与原生版本不同的版本号和名称。这是为了保持原作者代码的完整性。

  5. No Discrimination Against Persons or Groups 拒绝歧视。

  6. No Discrimination Against Fields of Endeavor license 不得限制任何人在某些特定领域使用开源项目。

  7.Distribution of License 开源项目已经获得的 license 对项目重新分配给的对象也应自动生效。

  8.License Must Not Be Specific to a Product license 不能只针对程序的某个特定版本或特定产品有效。

  9.License Must Not Restrict Other Software license 不 得对随着软件一起开放的其他软件加以限制。

  10.License Must Be Technology-Neutral license 的规定不得以任何独立的技术活接口形式为前提。它应该是和技术无关的。

- 开源对学习的帮助
  - 遇到问题首先可以直接查看源代码。开源社区中会有很多相关内容的讨论，可以方便的帮助你解决遇到的很多问题。
  - 各种开源的工具、软件和框架是我们提高生产力的好帮手。站在巨人的肩膀上，我们的视野更开阔。
  - 开源能提高我们的合作能力、沟通能力，它已经成为一种文化形式。在开源社区可以帮你找到志同道合的伙伴。

    - 开源社区很多都是英文社区，还能锻炼英文阅读和交流能力。

版本控制 Version Control

- 必要性
  个人对文件的频繁修改，导致一系列冗余和不必要的麻烦。多人对同一文件进行操作，更会产生同步和更新方面的问题。
- 单人更迭版本的控制思路
    - 将 ctrl+s 自动触发一个版本，会导致过多的无意义的版本，冗余，不必要。
    - 手动提交，当同一版本修改了若干文件时，这些文件需要理应同一批提交。
    - 文件所在的目录结构是层次复杂的，又应该怎样解决？SVN 在这方面的优点是每次只 checkout 一个目录，可以分目录管理权限。与之对应，git 只在根目录下有一个.git 文件夹来保存控制信息。
- 多人合作时的版本控制的发展
    - Visual Source Safe 要求同时只有一个人可以申请修改文件。在这个人提交之前，其他人没有权限对文件进行修改。这样显然效率过于底下。
    - SVN 改进的部分：利用全局自增长的版本号来确保不同人对版本进行的更新有序；原子提交的机制以防止部分提交失败；分支机制来确保可以同时进行多个工作而互相独立。
- 多人合作中需要考虑的问题
    - **版本号**　用来为一个阶段性的工作成果命名，版本号的推进意味着版本和功能的更迭。可以通过规定一些关于版本号命名的规范来方便用户从版本号中读取有用信息，例如奇偶数的含义，第几位小数点的含义，alpha、beta、final 隐含的阶段信息等。
    - **围绕版本的工作**　代码提交的方式不同。git 在提交中可以创建 bug 分支和 feature 分支。前者用于临时修复 bug，修复之后合并分支；后者用于提交和测试新特性，为了防止对主分支的干扰（因为还处于测试阶段），也可以临时创建它，之后合并。工作阶段也不同。包括特性开发期的版本，发布前准备期的版本，以及正式上线后针对稳定版的长期维护版本，在稳定版本上不断打补丁直至宣布 EOS（End of Support）。
    - **分支的含义**　分支功能提供了一个暂时的专属于你的仓库；当你在一段时间内需要独立工作，不想影响整体的代码，但又需要版本管理来维护你自己的提交，就可以自己创建一个分支，你的提交和推送都在自己的分支上进行，直到开发完毕后，再合并到主分支上。Git 对分支的创建、合并和删除都非常高效，所以最安全和合理的方法是你始终使用分支来完成某个任务，合并分支后再删除它。

最好的版本管理工具：Git

- 特点

    - 分布式版本控制，在每个人的本机上都有完整的版本系统；可以与远程代码仓库进行交互。分布式意味着可以离线工作，有了冗余机制。并且，分布式系统也有充当中央服务器的主机，来便于不同用户之间交换他们的推送。
- 基本操作
  之前的学习报告中已经总结和学习了 git 的常用操作，这里再简短的整理和复习一下。
    - git init: 初始化本地仓库
    - git add: 上传文件到暂存库
    - git add .: 上传所有有改动的文件
    - git commit -m "commit message"： 提交暂存区的文件到本地分支上并附加提交信息
    - git push origin master: 将提交到本地分支的内容推送到远程库。其中 origin 是远程库的名称，master 是本地分支的名称
    - git push -u origin master: 在初次推送的时候需要加上-u 的参数，它将本地库 master 和远程库 origin 关联起来
    - git status: 查看当前 add 和 commit 的状态
    - git log: 查看版本日志
    - git pull --rebase origin master: 获取远程库的更新内容

开源社区

- 开源社区在哪里
    - Github 就是很好的开源社区，每个项目下的 issues 模块和 pull requests 模块都是交流讨论的地方。Github 强大的 clone 和 branch 功能让用户能够自由且简单的参与到开源项目之中。首先 Fork 别人的开源项目，这个操作会让你在你的 GitHub 账号下克隆一个相同的代码库，你可以并且只可以操作这个你复制过来的代码库。在你的本地利用 git 指令从你的账号下 clone 仓库，然后你就可以在本地对它进行修改。将修改后的内容首先推送到你账号下面的代码库，然后对官方库发起 pull request，这表明你申请修改官方库。如果被接受，就意味着你为这个项目贡献了代码啦。
    - 社区的官网，邮件系统，各种论坛，StackOverflow，知乎等等都可以看作开源社区。
- 代码规范
  为开源社区贡献的代码必须是优雅的、工整的、符合规范的。怎么样算好的代码？
    - 代码中各种包、类、方法、变量的命名符合规范；
    - 注释的格式符合规范
    - 异常和错误处理符合规范
    - 有完备的测试用例
    - 好的代码要简洁并让人容易读懂

好的开源社区对提交信息和 pull request 也有规范性要求，要注意查看和学习特定项目的特定要求：

- 标题的规范和命名格式
- 内容，贡献的代码完成了什么工作
- 参考链接和关联的 issues
- 空行、缩进、每行字数等格式规范

测试用例有什么作用？

- 自动化测试套件是开源软件质量的保证
- 提交的代码需要通过测试用例
- 新添加的功能要一并给出配套的测试用例
- 主动贡献测试用例也是对开源社区的贡献

- 如何加入开源社区
  - 找到自己感兴趣的项目开始阅读和学习
  - 阅读项目的 issues， pull requests 列表来了解别人的讨论内容
  - 将项目下载到本地，安装、编译和使用
  - 使用项目的过程中遇到 bug，尝试修复以及上报给社区
  - 好的 bug 报告应该是：声明自己已经尝试了 xx 方法进行测试和解决，但仍存在某某问题
  - 在社区里回答别人的问题

开源项目

- 如何找到开源项目
  - 代码托管网站：GitHub， Google Code， Sourceforge， launchpad 等
  - 技术社区：infoQ， OSChina， CSDN 等
  - 其他：各大科技公司的研究院，Google Summer of Code，搜索引擎，微博，微信公众号，blog，BBS，StackOverflow， 知乎等
- 适合初学者的开源项目
  - 找自己感兴趣的项目，找自己常用的项目，带着目的和兴趣去选择项目
  - 找依赖少的项目。即尽可能不要依赖其他的项目、插件、库等内容，这样能方便上手，尽快的开始代码的阅读和学习
  - 找社区活跃度高的项目。这说明它的社区关注度高，开发者重视，并且遇到问题更容易在社区中找到答案
  - 同一个项目会有多年更迭的版本，可以首先选择较早的版本阅读，一方面早期版本功能更少，更容易上手学习；另一方面可以在阅读版本更替信息中来学习开发者的设计和更新思路

在讲座中学到的其他内容

- 老师讲到的他的面试问题：讲述你工作之前解决过的最困难的问题；如何解决的这个问题；向面试官清楚简洁的介绍问题所在的领域和背景。
- Talk is cheap, show me your code.

3.2 9.27 日讲座

姜宁老师的技术讲座主要分为两部分，第一部分介绍了开源基金会 Apache 以及它的社区理念，第二部分介绍了参与开源社区和开源项目的一些经历和建议。

第一部分

- **Apache Software Foundation** 是 Apache 的软件基金会，基金会维护和管理着社区以及开源在社区的各种项目。我们比较熟悉的有 Apache HTTP Server、Hadoop、Lucene、Maven、Struts、Tomcat 等。Apache 所支持和维护的项目都遵循 Apache License。
- **License**[1]： 开源许可证是一种用于开源项目的法律许可标志。虽然项目开源到社区和网络，但如果希望开放版权、供用户使用，仍需要开源许可来声明不保留版权，让用户免费使用、修改。

  许可证分为两类，宽松式的许可（Permissive License）对用户无限制，甚至可以修改代码后闭源，只要求注明原始作者，MIT、Apache 许可都属于这一类；Copyleft 许可对应于版权的 Copyright，含义是不经许可，允许用户复制，GPL、Mozilla 许可属于这一类。

  之前听到一些关于开源的名词，有些陌生，列举如下：

  Distribution：分发指将版权作品从一方转移到另一方。一般只有在分发时，才需要遵守许可证。

  Notice：所有的开源许可证都要求软件分发时要向用户披露软件中包含源代码以及代码的原作者。

- **项目领域**：AI，物联网，大数据，金融管理，气候变化，区块链
- **办事原则：社区胜于代码（Community over code）**[2]。首先维护社区质量。
  - 达成共识，让代码运行起来。Apache 社区不依赖权威，只依赖共识来做出决定。所以，快速达成共识有助于工作的展开，即使只是粗略的共识。先让代码运行起来，然后再去纠结后面的细节。

    - 小的可逆步骤。想到之前在《腾讯传》中看到关于腾讯的开发模式，他们提倡小步快跑，快速迭代。也即，每一次版本迭代只更新较小的内容，这样既不会让用户在使用体验上感受到较大的差异（可能导致不习惯），又能够加快迭代速度。每次少量更新也更有利于代码的维护和问题的发现。后面姜宁老师在讲到向开源项目进行 PR 的时候也最好每次只提交较小部分的修改，这样更有利于项目管理者对你提交的内容进行检查和审核。
    - 任人唯贤。通过为社区做出贡献来积攒声望和功绩。这种声望并不会给你带来权威和权力，但是能够提高你在社区中的公信力和知名度，让你更容易得到大家的认可。
- **实践创造完美：**
    - 开始未必是完美的。但要尽快开始，尽快达成共识，尽快投入工作。提早发布是更好的选择，因为互联网的世界更新换代速度太快了。小步快跑，频繁发布是 Apache 倡导的模式。初入社区的开发者和用户可能对复杂的协议和政策不熟悉，当然这些协议和规则更多都是指导性、建议性。社区新人还是要尽可能尽快融入社区之中，在开发中学习，保证每一个发布都比上一个更好。通过频繁的迭代更新，让项目逐步变得完美。

第二部分

- **社区**：A community usually refers to a group of people who interact and share certain things as a group.
- **开源的本质是开放和协作**。开放，降低了大家获取信息的成本；协作，让大家跨越各种界限进行合作。后面看到姜宁老师所列的他的一天，其中在中午和晚上两个特定的时间写着"欧洲/北美同事上线，邮件 IRC 交流。"这让我感慨这种工作模式的魅力。人在家中，就可以与世界各地志同道合的工作伙伴协作办公。这也正是开源社区和开源项目的魅力。
- **优势：** 企业的开发模式项目需求复杂、开发周期长（无法快速上线，可能被同类产品抢占先机），人员流动大、跨部门团队协作和沟通的流程复杂（沟通成本高）。相较之下，开源项目的开发能够轻易的实现跨域协作沟通、分布式开发（开源本身就是在这一基础上发展而来的），它依托于社区的管理和组织不断发展壮大，依托于开源爱好者的热情来进行开发，迭代快，效率高全方位开放。核心在于分布式开发。
- **对参与者的吸引力：** 参与者选择自己喜欢的项目投身其中，源于兴趣，充满热情；参与开源开发是一个挑战智力的过程，通过这个过程获得成就感和认可，就像上一周庄表伟老师讲的，背后有一种心理学效应在；工作模式灵活，如姜宁老师形容的，是一种"邮件驱动"的开发方式，工作的成果通过社区中的提交记录以及获得的积分声望等记录下来，你的工作成果都被记录下来，并通过某种 rank 的方式衡量、记录着，能给人一种打游戏过程中过关升级的感觉，这是充满成就感的；你的工作成果转化成了影响力：对社区贡献过好点子的人，对工作认真负责要求严格的人拥有更高的影响力，这提高了你的知名度和公信力，在开源社区就能收到更多的关注和尊重，你的能力也更能得到认可，之后可能会在社区的人员组织中更上一层；在开源社区中没有企业里那样严格刻意的上下级管理制度，只有技术更牛的人作为技术领导，带着你更进一步。

- **加入开源社区：** 为开源社区做贡献的方法很多，结合上一次讲座的内容，总结如下：
    - 首先是使用开源社区的软件。当你希望为一个项目做贡献——包括解决问题、提交 bug 以至贡献代码，你首先要了解它，这也是我们现在阅读开源项目的课程任务的目的之一。使用它意味着首先要将这个项目部署在本地环境中，那么如何安装，如何配置，如何使用等又是需要我们去尝试和试错的。这个过程本身就很有学习的价值。
    - 使用的过程中会遇到各种各样的 bug，首先通过自己的努力去解决这些问题，如果发现这的确是一个在社区中还找不到较好方法或者没有人提出过的问题，那么可以向社区提交自己的 bug 报告。无论提出问题还是解决问题，都是对开源项目做出的贡献。但要注意，好的 bug 报告需要对问题和问题发生的环境、状态有详细的描述，要给出发生 bug 的测试用例。只有让其他用户能够重现你的 bug，他们才更可能给出解决方案。
    - 文档。阅读项目文档是学习一个项目最好的方法之一，读文档的过程也帮助你熟悉和了解这个项目，这可以和前面说的使用项目软件一同进行。当遇到不懂的操作和功能时，翻阅文档像查找字典一样值得信赖。这里又有两个进一步学习的方向。一方面，当开源社区只提供英文文档时（当然一般英文文档的可靠性更高更准确），你完全可以通过自己的努力为社区贡献中文文档，这也是技术门槛比较低的为社区做贡献的方式（但它意义和责任重大，而且翻译的准确与否也涉及到很多未来的阅读者）。另一方面，当发现文档不能够满足需求的时候，可以自己动手来对文档进行完善，参考资料就是源代码。当你能够对项目文档做出完善，也就能够证明你对这个项目的理解程度了。
    - 更进一步。开源社区是一个互动的地方，Github 被称作全球最大的同性交友平台也不是完全没有道理。参与互动的方式多种多样，首先可以阅读项目的 issues 列表和 pull requests 列表，了解项目进展过程中曾经经历过的更迭和遇到的问题以及大家的讨论，从发展的眼光、改变的角度来由浅入深的了解一个项目是一种好的学习方法。进一步的可以参与邮件列表的讨论，帮助软件用户解决问题。但要注意，社区是一个平等、友善、尊重、信任、谦逊的平台，沟通要以技术为基础，以互相尊重为前提，不要提问简单重复的问题，要让自己的问题有价值、值得社区讨论和回答。对社区的接触从提问开始，逐步丰富完善强化自己，最终变得能够解决别人的问题。

开始

从示例代码开始是一个好的选择，从单元测试模块开始也容易入手；读文档和翻译文档都可以学习，直接上手源代码借助注释和实践的帮助也可以学习；回答问题很棒，提出好的问题也丝毫不逊色——无论怎样，先开始吧，先迈出第一步，从零开始，从任何可能的起点开始。正如 Apache 所倡导的那样，先让代码工作起来。一无所知的时候就追求完美是最愚蠢的，那也可能只是你逃避开始的一个借口。喜欢讲座中的一句话，用以自勉：

在开源社区唯一阻挡你的是自己的惰性！

参考资料

[1] 阮一峰：开源许可证基础知识扫盲 https://www.oschina.net/news/90054/opensource-license-introduction

[2] 程序员 Apache 开源社区进击之路 https://www.oschina.net/news/80239/programar-develop-in-apache-software-foundation

Part.4 开源项目阅读

4.1 第一周学习报告

时间：9.11-9.17

1、项目名称：bottle
Github 地址：https://github.com/bottlepy/bottle

2、项目介绍：
Bottle 是一个快速、简洁、轻量级的基于 WSIG 的微型 Web 框架，此框架只由一个 .py 文件组成，除了 Python 的标准库外，其不依赖任何其他模块。不足 5000 行代码，包含了如下 web 框架常用的功能：
路由系统：将不同请求交由指定函数处理。
模板系统：快速的、python 化的内联模板引擎，支持 mako、jinja2、cheetah 等模板。
公共组件：用于提供处理请求相关的信息，如：表单数据、上传文件、cookies、请求头和其他 HTTP 相关的数据等。
服务：内置 http 开发服务器，支持 paste、fapws3、bjoern、google app engine、cherrypy 或任何其他支持 wsgi 的 http 服务器。

3、选择原因
（1）bottle 的代码量较小，只有四千多行，并且结构简单，都包含在同一个 python 文件中，对于初学者来说阅读门槛相对较低；
（2）bottle 由 python 语言编写，这门语言大家都有所了解，不需要在阅读前专门再去学习语法。并且该项目只依赖 python 标准库，不依赖其他的 python 框架和库，这样就不需要在阅读源码之前再去安装其他的框架，也不需要在阅读过程中遇到其他框架中的函数再去专门阅读其他的框架代码；
（3）bottle 是一个 web 开发的框架，在后续的项目实践开发中可以用到，经过深入学习和源码了解后也有较好的实用性；通过以 bottle 作为例子进行学习也能帮助我们了解到 web 框架的作用和工作原理。

4.2 第二周学习报告

时间：9.18-9.24

我们组阅读的 bottle 是一个由 python 写成的轻量级 web 框架。在具体去读它的代码之前，首先要了解一下 web 框架究竟用来做了什么。之后，简单了解了在 bottle 这个框架中，是如何和来自浏览器的 HTTP 请求进行对接和匹配的。

1、HTTP 请求和响应

浏览器发送 HTTP 请求（request）给服务器，服务器解析请求后，给出返回结果，发送 HTTP 响应（response）给浏览器。

请求和响应的格式都是固定的，包含头信息 header 和可能有的数据信息 body。

http request：

http 方法(GET,POST 等)，路径，域名。POST 类型的请求还会包含一个 body 部分，里面是用户数据。

http response：

响应代码（200 成功，3xx 重定向，4xx 客户端发送的请求有误，5xx 服务器端处理出现问题）；响应类型（content-type 项，例如有 html 文本，各种格式的图片等响应）；响应会携带内容在 body 中，例如服务器端返回的 html 代码。

一个 http 请求只处理一个资源。我们要编写的是服务端的响应，那么我们在 response 相关的函数中把 html 信息发送回去就好了。

Web 应用分为静态和动态两种。

静态的 web 应用：先把 html 文件存储好，用一个 http 服务器软件（apache，nginx 等）接收用户请求-读取存好的 html-返回给用户。

动态生成 html 的 web 应用：要在服务器端动态创建出 html，然后在浏览器向我们的服务器发送 http 请求时，我们可以针对不同的请求将不同的 html 结果通过 http 响应的方式（作为响应的 body 部分）返回给它们。浏览器收到响应后，就取出 html 显示在浏览器中。由于内容需要动态生成，这些动态生成的部分需要我们用 python、java 等后台语言编写。但服务器发过来的请求不能直接和这些后他语言相关联，这时候需要一个中间人:WSGI。

2、WSGI

WSGI 要求服务器端实现一个响应函数（网上的示例中一般写作 application），这个函数用来针对特定的 url 和 http 方法（method）响应 http 请求。这个函数要接受两个参数，environ 是一个字典，包含了环境信息、各种配置等；start_response 是一个可调用的函数,它接受 status 和响应头两个参数。这两个都是由 http 协议传送过来的。

利用 WSGI，我们可以直接读取这两个参数；否则我们就要自己捕获、解析 http 请求等。这时候，我们只需要关心如何从 environ 这个字典中拿到 http 请求信息，然后构造 html，通过 start_response 函数发送 header，在 return 语句中返回 body。

为了得到这两个参数，我们必须用一个 WSGI 服务器来调用这个响应函数。只有服务器来调用它，我们才能获悉传递给响应函数的两个参数。

这个响应函数就是一个 WSGI 应用，但 WSGI 应用可以是任何可以被调用的对象。如果是函数，就直接调用它；如果是类，那么这个类要实现__call__方法以供调用。

利用 python 自带的 wsgiref 模块可以实现一个最简单的服务器。通过 make_server 创建服务器，指定 ip，端口，可调用的对象。这里只是用来启动服务器，打开某个 url。实际的响应内容都在“可调用的对象”里面。

这个流程大致为：服务器收到请求--解析请求报文并将解析结果存放在字典

environ 中--字典和另一个函数 start_response(用来处理头信息)作为参数,调用 WSGI 应用--应用的返回结果成为新的 response，返回给客户端。

总结一下：

Web 应用程序的入口都是一个 WSGI 应用。HTTP 请求的所有输入信息都可以通过 environ 获得,HTTP 响应的输出都可以通过 start_response()加上函数返回值作为 Body。Web 应用通过写不同的 WSGI 处理函数，针对不同的 HTTP 请求进行响应。我们需要 web 框架来帮助我们处理不同的 HTTP 请求。

在 WSGI 接口的层次上进一步抽象，专注于用一个函数处理一个 url。url 到函数的映射交给 web 框架。

将 url 和函数关联起来的机制：python 装饰器 @app.route（url，method)。

接着后面 def 一个函数就好，用来专门处理这个 url 和对应方法的 http 请求。

3、Bottle

Bottle 框架中起主要作用的有这么几个类，Bottle，Route，Router。

每个 Bottle 类的实例都是一个 WSGI 的应用，它可以作为可响应的对象。但既然它是一个类，就需要首先在__call__方法中响应。这个方法接收 environ 和 start_response 参数，接着调用 wsgi 方法，将两个参数再传过去。

wsgi 方法通过调用 Bottle 类的_cast 方法来给出返回内容。_cast 方法将参数转换成 wsgi 可接受的内容，并设置正确的 http 响应头。这个方法需要一个参数，这个参数由_handle 方法提供。

_handle:用服务器传过来的 environ 作为参数，首先解析出路径 path；然后将这个 env 字典和全局变量 request 绑定。之后，两个参数 route 和 args 通过路由器中的匹配方法 match 来匹配相关的响应函数。match 接收的参数就是这个 env 字典。解析出 route 和 args 的结果之后，将它们再存到 env 字典中。得到的参数 args，调用 route 类的 call 方法，call 返回的结果就是 out 的内容。

_cast 方法给出的结果就是 wsgi 方法正常情况下应该返回的内容，也就是我们前面将 wsgi 应用的作用时提到的返回给浏览器的响应 response 中的 body 部分数据。http 状态码和 response 的 header 部分通过 wsgi 方法接收的 start_response 函数参数进行传递。这样，我们前面将的 wsgi 的相关知识就在 bottle 这个 web 框架中找到了对应。

上面这一段提到了 match 方法，这个方法就是用来解决我们在原理部分提到的，需要将不同的 WSGI 处理函数和它们能够处理的 HTTP 请求相关联的问题。利用 Bottle 写 web 应用时，通过 route 装饰器将响应函数绑定到一个（或一类，利用通配符）url 上，将 url、响应函数、http 方法和其他参数等组合到一起，成为一个 Route 类的实例 route。这是 Bottle 的核心。

为了不写太多的静态路由匹配一个又一个的 url，用动态路由的方法来使得映射函数可以通配某类 url。一个通配符包含在一个<>中，通配符之间用/分隔开。并且，动态路由中 URL 中的通配符都可以作为参数传递到映射函数中。

Router 提供的用来筛选通配符的 filter 有四类，int，float，path 和 re，默认的 filter 为 re，也就是正则表达式类型，也提供了一个默认的正则匹配模式。也可以用 add_filter 方法来添加新的 filter。

每一个 route 都是一个 WSGI 应用，可以根据需要创建多个 route，用于适配不同的 HTTP 请求。为了整体管理这些 route 对象，Router 类的 add 方法会将它们都添加到 Router 类的一个实例中。router 中包含两个分别存储静态 url 和动态 url 的集合 static 和 dyna_routes，添加进来的 route 对象存储在这里面。

由于 router 对所有的 route 进行统一管理，所以在这个类中的方法 match 用于匹配 HTTP 请求，根据其中的路由信息返回第一个满足请求的 target。target 可以是 id，字符串或响应函数。

这一周大概了解了一个 web 框架的设计思路和作用，并带着这种整体的思路在 bottle 的源代码中找到了对应的部分。目前对 bottle 的结构和处理请求的流程有了基本的了解，接下来要深入到方法中学习一下 bottle 具体如何实现它的功能。

参考资料：

https://www.cnblogs.com/rainfd/p/bottle2.html

https://www.liaoxuefeng.com/wiki/1016959663602400/1017805733037760
http://www.zhangdongshengtech.com/article-detials/251

4.3 第三周学习报告

时间：9.25-10.11

WSGIHandler 函数

这个函数要接受两个参数，environ 是一个字典，包含了环境信息、各种配置等；start_response 是一个可调用的函数，它接受 status 和响应头两个参数。

这个方法首先将接受的 environ 参数绑定到 request 对象上，bind 方法主要是做一些初始化的工作，包括从字典变量 environ 中提取出有用的基本信息，包括 url 地址，http 方法，cookie 信息等。

这些内容被转存到 request 对象中，并作为参数来调用 match_url 方法。这个方法实现了将不同的 WSGI 处理函数（也就是此处得到的 handler 返回值）和它们各自能够处理的来自不同 url 和 http 方法的请求关联起来的功能。args 是一并找到的和这个处理函数有关的参数。

如果没有找到能够处理当前 url 和 http 方法对应的请求的函数，那么就是我们常见的 404 not found 错误。这意味着客户端发送的请求有误，这个请求不是服务器能够处理的请求，4xx 的错误码意味着客户端请求出错；

如果是服务器端出现错误，也就是发生了异常 Exception，那么会报 500 错误，5xx 的错误码意味着服务器端报错，这时候调用 bottle 的相关异常处理机制来解决问题，此处不再深究。

```
def WSGIHandler(environ, start_response):
    # 全局 request、response，每个线程独立
    global request
    global response

    # bind 当前 environ 数据
    request.bind(environ)
    response.bind()
    try:
        # 根据 path 和 method 找到处理方法和参数
        handler, args = match_url(request.path, request.method)
        if not handler:
            raise HTTPError(404, "Not found")
        # 执行返回 output 数据
        output = handler(**args)
    except BreakTheBottle, shard:
        # Bottle 错误产生的输出
        output = shard.output
    except Exception, exception:
        # 处理内部错误，500 错误
        response.status = getattr(exception, 'http_status', 500)
        errorhandler = ERROR_HANDLER.get(response.status, error_default)
        try:
            output = errorhandler(exception)
        except:
            output = "Exception within error handler! Application stopped."

        if response.status == 500:
            request._environ['wsgi.errors'].write("Error (500) on '%s': %s\n" % (request.path, exception))

    db.close() # DB cleanup
```

如果一切正常，这个 handler 和相关的 args 参数就是我们要返回的内容。这意味

着我们成功找到了处理对应请求的函数。但在返回它之前，还有一些其他工作要做。

首先是查看这个处理函数是否包含 read 属性。如果有，则这是一个类似文件的对象（file-like objects），那么用字典文件 environ 中的处理文件的修饰器 wsgi.file_wrapper 来对它进行操作；

若不是文件对象，再判断它是否是字符串类型的对象。如果是，将它作为列表的元素进行输出；

最后，要将响应对象 response 中的 cookies 内容提取出来，按字段分别添加到响应头 response.header 中。再将 response 中的状态码 status 提取出来，并根据状态码得到对应的状态描述信息，组合成需要返回的响应信息中的状态信息。

将状态信息和响应头作为参数提交给 start_response 函数，这是 http 响应请求的 header 部分；将 output 直接返回，这是 http 响应请求的主体 body 部分。

```
if hasattr(output, 'read'):
    fileoutput = output
    if 'wsgi.file_wrapper' in environ:
        output = environ['wsgi.file_wrapper'](fileoutput)
    else:
        output = iter(lambda: fileoutput.read(8192), '')
elif isinstance(output, str):
    output = [output]

for c in response.COOKIES.values():
    response.header.add('Set-Cookie', c.OutputString())

# finish
status = '%d %s' % (response.status, HTTP_CODES[response.status])
start_response(status, list(response.header.items()))
return output
```

Part.5 其他

5.1 Git 的使用

时间：9.19-9.20

Git 指令

我的本机上有一些笔记文件。首先把这些文件上传到 github，借助这个过程来熟悉一下 git 操作。

• 初始化

选择一个文件夹作为本地库的存放位置，在文件夹下右键选择 git bash here，打开 git 的命令行。执行如下命令来初始化一个本地仓库。

Git init

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note
$ git init
Initialized empty Git repository in E:/note/.git/
```

• 添加文件

将之前的笔记文件复制到这个目录下，执行如下命令来将文件添加到暂存区。

Git add <filename>

如果需要将所有内容都添加：

Git add －A（需要大写）

或

Git add .

.git	2019/9/20 18:31	文件夹	
编译原理.docx	2017/12/29 13:09	Microsoft Word ...	5,373 KB
信息检索.docx	2018/6/22 19:35	Microsoft Word ...	1,364 KB

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git add -A
```

• 查看状态

可以不断向暂存区中 add 文件，当不知道上传的内容后，可以执行如下指令来查看当前暂存区的状态：

Git status

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   "\344\277\241\346\201\257\346\243\200\347\264\242.docx"
        new file:   "\347\274\226\350\257\221\345\216\237\347\220\206.docx"
```

• 提交文件

多次将文件 add 到暂存区后，可以一次性提交到分支：

Git commit －m “commit message”

后面附加的是当前提交的说明信息。这个信息很重要，可以帮助识别和区分不同的版本，以及不同提交实现的不同功能等等。最好是能够有一套信息规范。

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git commit -m "commit demo"
[master (root-commit) c360b05] commit demo
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 "\344\277\241\346\201\257\346\243\200\347\264\242.docx"
 create mode 100644 "\347\274\226\350\257\221\345\216\237\347\220\206.docx"
```

这时候，再用 status 命令查看仓库的状态，之前暂存区的那两个 to be committed 的文件已经没有了，取而代之的是 working tree clean。

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git status
On branch master
nothing to commit, working tree clean
```

• 查看日志

日志文件中保存了各个版本提交的信息，使用如下指令来查看日志：

Git log

可以看到上一次提交的信息“commit demo”。

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git log
commit c360b0592820864edd099136197edd8c8b8e2e46 (HEAD -> master)
Author: Young <fuyuxyfy@163.com>
Date:   Fri Sep 20 18:47:38 2019 +0800

    commit demo
```

• SSH 加密设置

本地仓库和 GitHub 上的远程仓库之间信息传输需要通过 SSH 加密。下面的配置教程主要来自廖雪峰的 git 教程。

创建 SSH Key。

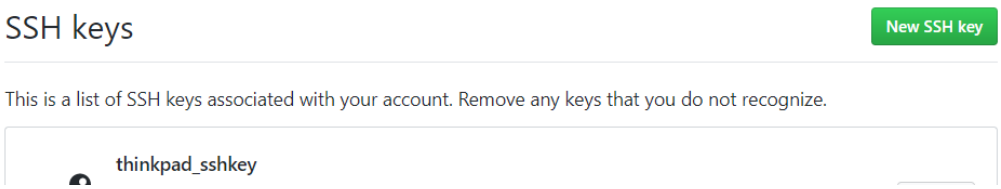
检查用户主目录下是否有.ssh 目录以及目录下是否有 id_rsa 和 id_rsa.pub 两个文件。前者是私钥，后者是公钥。如果没有，在 git bash 中执行如下命令：

Ssh-keygen －t rsa －C “emailaddress”

后续的存储位置和输入密码只需要用 enter 跳过，即使用默认位置和不设置密码。

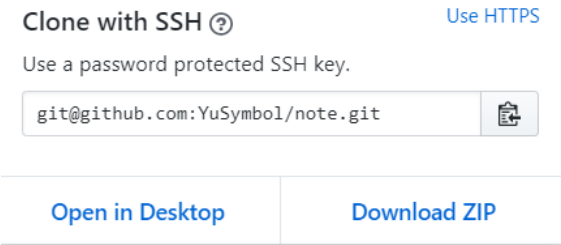
```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ ssh-keygen -t rsa -C "fuyuxyfy@163.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Young Shaw/.ssh/id_rsa):
Created directory '/c/Users/Young Shaw/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Young Shaw/.ssh/id_rsa.
Your public key has been saved in /c/Users/Young Shaw/.ssh/id_rsa.pub.
```

打开 GitHub，找到 settings-personal settings-SSH and GPG keys 目录，新建一个 SSH key 项，将 id_rsa.pub 中的内容复制到里面就可以了。



• 创建和绑定远程仓库

首先在 github 上创建仓库，复制它的 SSH 地址：



此时，我们的本地仓库中已经存放了一些文件，但新建的这个位于 github 上的与之对应的远程仓库还是空的。那么为了实现它们 之间的传输，首先要将两个仓库绑定：

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git remote add origin git@github.com:YuSymbol/note.git
```

• 推送更新

之前经过 add 将更新上传到暂存区，经过 commit 将更新提交到分支。但这些操作产生的影响都是对本地的，还需要将这些都推送到 github 上保存下来，需要执行如下指令：

Git push －u origin master

其中，-u 这个参数是只有初次推送的时候才需要加上的；master 是本地分支的名字，origin 是远程库的默认名字，并且一般都被用于指代远程库。

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git push -u origin master
The authenticity of host 'github.com (13.229.188.59)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,13.229.188.59' (RSA) to the list of known
 hosts.
To github.com:YuSymbol/note.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:YuSymbol/note.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

但是输入指令后出现了错误。这是因为我们在 github 上创建 repository 时一般都会随着库的创建自动生成一个 readme 文件。这个文件在本地库中是没有的，所以在将本地信息向远程推送时，必须确保要先把这个文件传送到本地。这个指令也是我们从远程获取更新信息的指令：

Git pull －rebase origin master

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git pull --rebase origin master
warning: no common commits
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:YuSymbol/note
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master
First, rewinding head to replay your work on top of it...
Applying: commit demo
```

之后，我们能发现 readme.md 文件已经在本地文件夹中了：

名称	修改日期	类型	大小
.git	2019/9/20 23:24	文件夹	
README.md	2019/9/20 23:23	MD 文件	1 KB
编译原理.docx	2019/9/20 23:23	Microsoft Word ...	5,373 KB
信息检索.docx	2019/9/20 23:23	Microsoft Word ...	1,364 KB

然后重新执行一遍 push 的指令(不需要再重新 add 和 commit 了)：

```
Young Shaw@DESKTOP-E5S8E20 MINGW64 /e/note (master)
$ git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 6.50 MiB | 869.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To github.com:YuSymbol/note.git
   364558f..59259ec  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

打开 github，可以看到已经上传到了远程仓库中：

README.md	Initial commit	5 hours ago
信息检索.docx	commit demo	8 minutes ago
编译原理.docx	commit demo	8 minutes ago

README.md

note

daily learning note

这些就是创建本地仓库、远程仓库、将二者进行关联、本地上传、从远程读取这些基本流程需要用到的指令了，可以说是最常用最基本的指令。其他的指令在之后用到之后，我会对这个文档保持更新。

Android Studio 与 Github 的关联

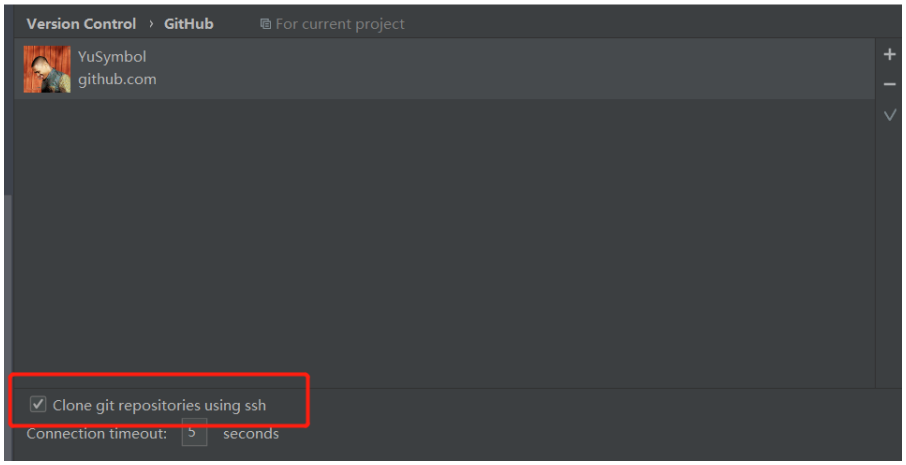
时间：9.21-9.22

• 关联 github

路径：File | Settings | Version Control | GitHub

在当前界面上输入自己的 Github 账号的密码。

前面我们已经配置了 SSH 连接，此处选中使用 SSH 以避免每次在版本更新时输入密码：

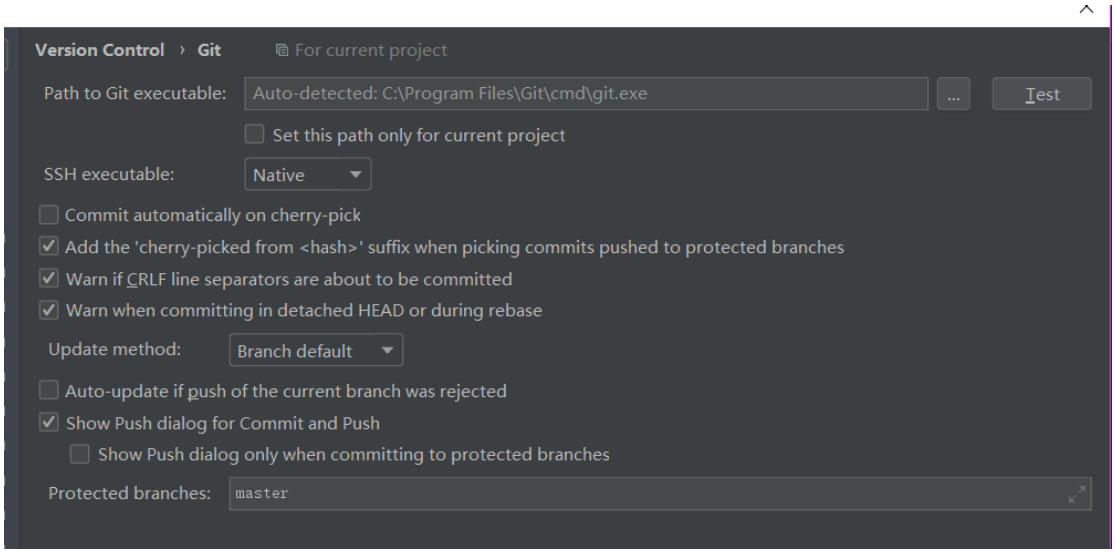


• 关联 git

路径：File | Settings | Version Control | Git

这里会自动检测到本地的 git 所在位置。其他的选项我都保持默认设置。

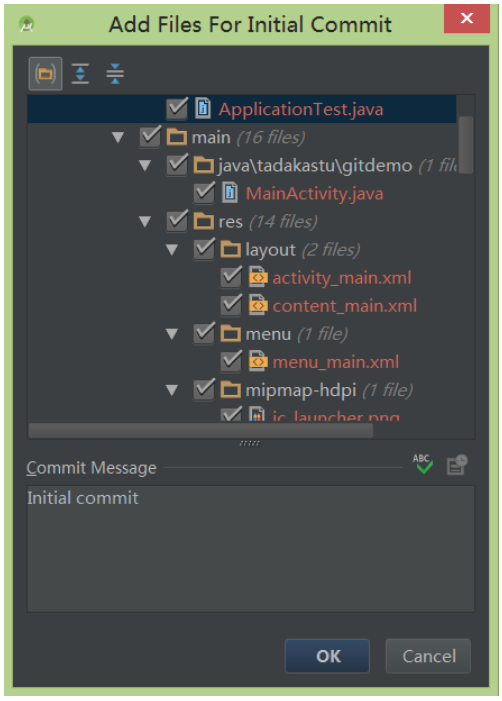
可以点击 test 来测试 git 路径是否正确。



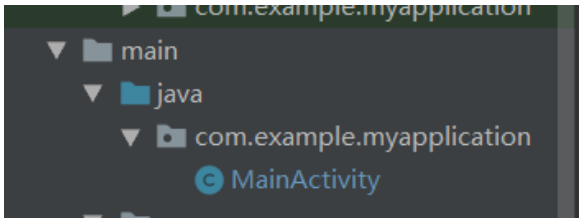
• 当前 project 上传到 github

路径：VCS|Import into Version Control|Share Project on Github

第一次提交时，会弹出对话框，让你输入远程库的名称和描述（之后就会在 Github 上自动创建这个库，而不像直接用 Git 命令行那样需要提前在本地和远程都先创建好库）。创建完成后，会发现项目中所有的文件都变成了红色。不要紧张，这只是说明这些内容都没有和远程同步，也就是还没有更新的内容。将它们全部选中，填写熟悉的 commit 信息，然后选中 commit and push，就可以在 github 上看到这些内容了。



每当有文件发生了更新，文件名称会变成蓝色，提醒你可以进行更新。



5.2 简单爬虫

时间：8.12-8.18

知识点

之前简单了解过 Python 编写爬虫，但没有具体实践过。这次通过网络资料自学编写了一个爬取音乐的 Python 程序，借助这个过程了解了一些相关的 Python 库的作用和用法，以及编写爬虫的基本思路，还接触到了借助浏览器的调试功能进行抓包分析以及网页源代码分析的一些知识。

Python 库和模块

selenium：web 自动化测试工具

tkinter：绘制图形界面

requests：web 相关的模块，用于发送请求，获取响应结果等

re：正则表达式模块

urlretrieve：下载二进制文件

浏览器调试功能

抓包分析

提取元素

……

正则表达式

程序编写

问题分析

以获取 QQ 音乐网站中的内容为例。在 QQ 音乐的首页输入关键字“理想”，会得到下图所示的返回列表，和当前搜索结果页对应的 url。

https://y.qq.com/portal/search.html#page=1&searchid=1&remoteplace=txt.yqq.top&t=song&w=理想

借助调试功能我们找到每个曲目对应的歌曲页 url，那么首要的任务是获取其中的链接。

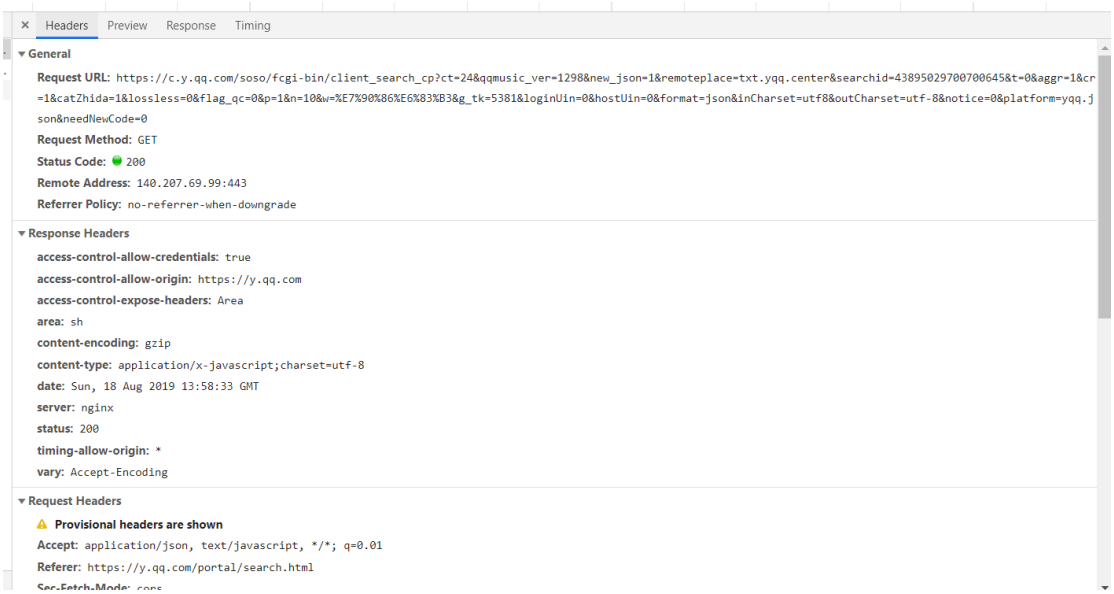
```
▼<span class="songlist__songname_txt">
  ▼<a href="https://y.qq.com/n/yqq/song/002YiXmX3PKZHE.html"
  class="js_song" title="理想 《陈二狗的妖孽人生》电视剧主题
  曲">
      <span class="c_tx_highlight">理想</span> == $0
      <span class="songlist__song_txt">《陈二狗的妖孽人生》电
      视剧主题曲</span>
    </a>
  </span>
```

但查看网页源代码后发现，这些 html 元素是触发 js 动作后才出现的，并不能通过直接爬取网页源码获得。利用调试功能中的 network 项，对发起搜索后的网页抓包结果进行分析：

（1） clear 清屏

（2） 输入关键词进行搜索，在 network 界面得到结果。

（3） Headers 为头部信息，能看到发出请求的 url 和请求类型（GET），以及一些参数信息。Preview 中是返回资源的预览信息，虽然大多数看不懂，但其中能发现一些返回的搜索结果列表的蛛丝马迹，验证了之前的猜测，这些曲目列表的 html 代码的确是通过这样一次 request 之后才返回的。



抓包结果中并没有找到明显的我们所需求的 url 地址，猜测应该是被加密过了。这里没有深入了解对 js 内容进行解密，而是借助一个 Python 库 selenium 来解决这个问题。

获取曲目 url

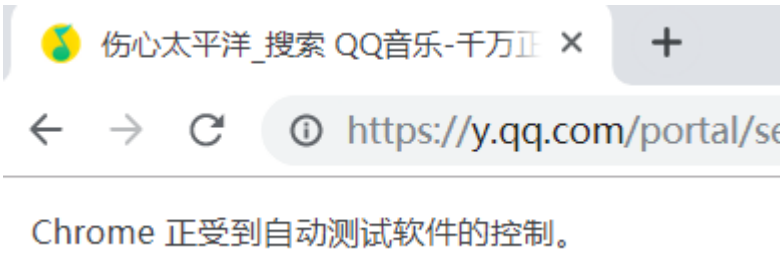
根据网上资料所说，selenium 库“可以模拟真实浏览器，自动化测试工具，支持多种浏览器，爬虫中主要用来解决 JavaScript 渲染问题”。我们这里就用它来直接获得触

发 js 事件之后返回的网页源代码。

导入：
```
from selenium import webdriver
```

使用：
```
driver = webdriver.Chrome()
driver.get(url)
```

根据使用的浏览器不同，可以选择 Chrome，Firefox 等不同方法。通过调用函数自动打开的浏览器中，会显示它“正受到自动测试软件的控制”。



这里需要的 url 参数，就是我们刚才在抓包中看到的 request url 的内容，也就是最开始提到的搜索结果页的 url。这样 driver 变量中就存入了我们所需要的源代码，接下来我们需要在这些代码中定位到链接所在的位置：

```
Elements  Console  Sources  Network  Performance  Memory  Application  Security  Audits
<!--结果列表 S-->
<!-- 单曲 -->
▼<div class="js_search_tab_cont search_tab_cont" id="song_box" style="display: block;">
  ▶<div class="mod_songlist_toolbar">…</div>
  ▼<div class="mod_songlist">
    ▶<ul class="songlist__header">…</ul>
    ▼<ul class="songlist__list">
      ▼<li class mid="100641069" ix="10">
        <!--
                hover : songlist__item--hover
                偶数行: songlist__item--even
                -->
        ▼<div class="songlist__item" onmouseover="this.className=(this.className+' songlist__item--
        hover')" onmouseout="this.className=this.className.replace(/ songlist__item--hover/, '')">
          ▶<div class="songlist__edit songlist__edit--check sprite">…</div>
          ▼<div class="songlist__songname">
            ▶<a href="https://y.qq.com/n/yqq/mv/v/u0015fv5wtk.html" class="songlist__icon
            songlist__icon_mv sprite" rel="noopener" target="_blank" title="MV">…</a>
            ▼<span class="songlist__songname_txt">
              ▼<a href="https://y.qq.com/n/yqq/song/002YiXmX3PKZHE.html" class="js_song" title="理想 《陈
              二狗的妖孽人生》电视剧主题曲" > == $0
                <span class="c_tx_highlight">理想</span>
                <span class="songlist__song_txt">《陈二狗的妖孽人生》电视剧主题曲</span>
              </a>
            </span>
            ▶<div class="mod_list_menu">…</div>
          </div>
          ▶<div class="songlist__artist" title="赵雷">…</div>
          ▶<div class="songlist__album">…</div>
          <div class="songlist__time">05:13</div>
          <div class="songlist__other">


                    </div>
        </div>
      </li>
      ▶<li class="js_songlist__child" mid="200669704" ix="11">…</li>
      ▶<li class="js_songlist__child" mid="102696799" ix="12">…</li>
      ▶<li class mid="951391" ix="13">…</li>
      ▶<li class mid="203642550" ix="14">…</li>
      ▶<li class mid="102297656" ix="15">…</li>
      ▶<li class mid="4950025" ix="16">…</li>
      ▶<li class="js_songlist__child" mid="794845" ix="17">…</li>
      ▶<li class mid="235474092" ix="18">…</li>
      ▶<li class="js_songlist__child" mid="105856112" ix="19">…</li>
    </ul>
```

这里又学到了 Chrome 的一个小技巧。在调试界面中，对这个 html 标签<a></a>右键-Copy-Copy XPath，可以定位这个标签在整个页面的 html 元素中的位置。复制的内容如下：

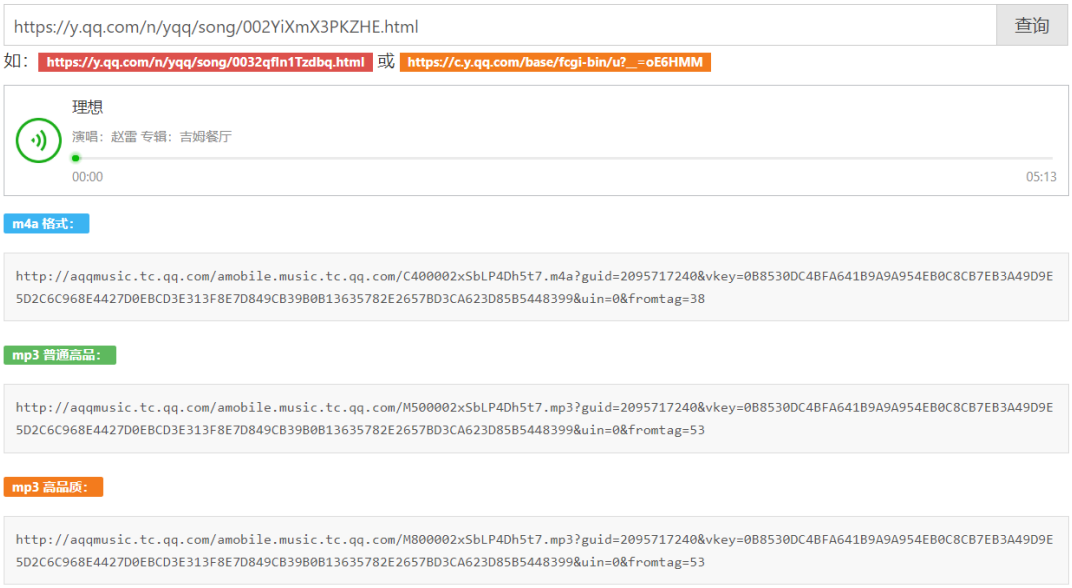
//*[@id="song_box"]/div[2]/ul[2]/li[1]/div/div[2]/span/a

这一段的意思是，首先以一个唯一的 id 为“song_box”的标签为起点，找到这个标签下的第二个 div 标签，找到这个 div 标签下的第二个 ul 标签，以此类推，最终定位到 a 标签。然后利用 selenium 提供的利用 XPath 查找元素的方法，来定位标签。我们需要的是 a 标签中的 href 属性，所以在附加一个 get_attribute 方法。

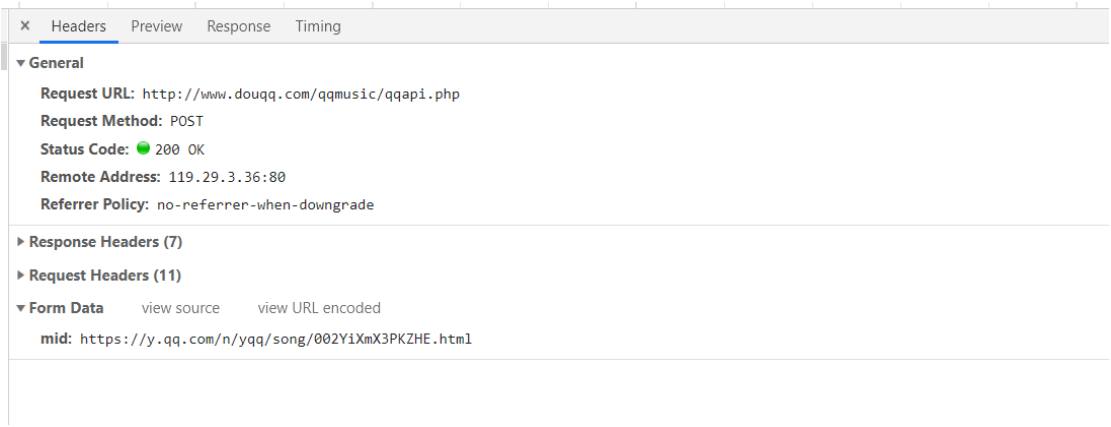
```
data = driver.find_element_by_xpath("//*[@id='song_box']/div[2]/ul[2]/li[1]/div/div[2]/span[1]/a").get_attribute('href')
```

获取下载地址

这里借助了一个第三方网站 http://www.douqq.com/qqmusic/，它提供了一个接口，我们给出歌曲页 url，就能得到它解析好的音乐下载 url。这里一个是简化操作，另一个借助它的帮助可以下载到一些需要付费的音乐。输入我们前面提取到的 href 属性中的 url，在该网站上的检索效果如下：



有了前面的经验，我们再对它进行抓包分析：



发现，这里是提供了一个 POST 类型的请求，请求的参数不包含在 url 中（所以 request url 总是不变的），它通过附加的 data 信息来传递参数。那么我们只需要把我们进行检索的 url 封装成该网站所需求的格式（mid: url）就可以了。这里又用到了 Python 的 requests 模块来发送 POST 请求。该方法中，url 参数就是 request url 对应的内容，data 参数就是 POST 方法传递的信息。这里我们传入的 data 要符合网站需要的格式，也就是一个键为 mid，值为 url 的字典变量。

通过 POST 请求后返回的结果，我们直接存取它的 text 内容。这里又有一个小技巧，利用 json 模块的 loads 函数对返回的 text 内容进行处理，可以使结果得到一定的格式化，以便于后续的处理。

```
url = "http://www.douqq.com/qqmusic/qqapi.php"
req = requests.post(url, data=data).text
# 用json模块进行处理，但返回结果还是字符串类型
req = json.loads(req)
```

处理的结果中存在\/这样的错误，这可以通过 replace 函数来替换成\或/。无论是正斜杠还是反斜杠，在 url 地址中都是可以被正确识别的。

{"mid":"003u2qmP0Mp2pW","m4a":"http:\/\/aqqmusic.tc.qq.com\/amobile.music.tc.qq.com\/C400

```
# url不区分正反斜杠，//或\\都可以
req = req.replace('\/','/')
```

网站能够给出多种音质的下载链接，我们只选择其中一种就可以。以“mp3 普通高品”音质为例，我们找到它在返回结果中对应的部分，是以“mp3_1”开头后接 url 的格式，那么用正则表达式来找到它后面的地址是很好的选择。

"mp3_1":"http://aqqmusic.tc.qq.com/amol

```
pattern = re.compile('"mp3_1":"(.*?)",')
# findall返回得到的是列表类型
result = re.findall(pattern, data)[0]
```

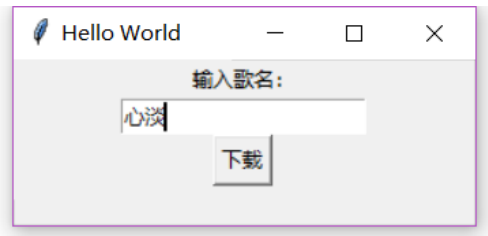
下载二进制文件

这里利用了 Python 的 urllib.request 库中的 urlretrieve 模块来直接进行下载，可以作为知识储备直接记下来。

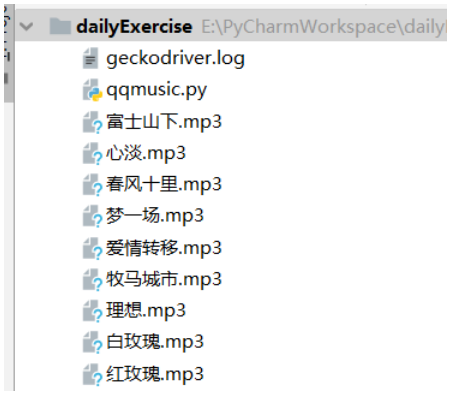
```
from urllib.request import urlretrieve
urlretrieve(url, name+'.mp3')
print("{} 下载成功！".format(name))
```

其他工作

将相关功能封装到函数中，利用 tkinter 库绘制窗口，通过在文本框中输入歌曲名来进行下载。目前只学习了这一功能并简单实现，还没有进行窗口的美化以及 exe 程序的封装等。




```
E:/PyCharmWorkspace/dailyExercise/qqmusic.py:88: 
  driver = webdriver.Chrome(chrome_options=option
https://y.qq.com/n/yqq/song/000ZdefE3ZQLlu.html
{"mid":"000ZdefE3ZQLlu","m4a":"http:\/\/aqqmusic.
{"mid":"000ZdefE3ZQLlu","m4a":"http://aqqmusic.tc
http://aqqmusic.tc.qq.com/amobile.music.tc.qq.com
心淡 下载成功！
```



问题和总结

问题

利用 selenium 库中的方法自动控制浏览器时报错，经查，发现我所安装的 3.x 版本的 selenium 库需要 Chrome 和 Firefox 等浏览器的相应组件支持，并且组件版本和浏览器版本也要对应，不然仍会报错。Chrome 的组件叫做 chromedriver，Firefox 的组件叫做 geckodriver，下载后需要安装在配置过环境变量的目录中。

总结

• 对爬虫的各种神奇之处有所耳闻，但没有实践过。很好奇，这个程序我知道只是爬虫最最基础的应用，但是已经足以让我激动以及激发起我的兴趣。我相信兴趣是最好的老师，也希望从这些容易上手的简单程序开始逐步熟悉 Python 和爬虫。

• 学习的过程中会伴随着意想不到的收获。例如写这个程序的过程中我第一次了解了抓包和分析响应信息的思路方法，给我很大的启发，浏览器中的各种妙用也让我惊喜。因为值得学的东西，无论大的知识点还是小的 tips 都太多了，所以每时每刻都能接触到新鲜的内容，也每时每刻都可以因为一点小小的进步而开心。这种满足感对于持续的学习是很好的反馈和帮助。

•写报告的过程也是对编写程序过程的一个复习和总结。有些小知识点在学的时候，可能因为教程直接告诉你了就没有深入的了解，但是在写报告的时候就摸不着头脑，因此会接着去查找相关的知识，这样相当于又学到了新的东西。而且书面的内容不同于意识流，它要求思路相对条理，内容相对清晰，这样也能逼迫自己重新整理思路。

Part.6 总结

• 算法题从易到难，坚持练习，LeetCode 是很好的平台。坚持刷题一方面保持写代码的手感，提高自己对编程语言的熟悉程度，并且 LeetCode 中很多题都能学到编程的技巧；另一方面在做题的过程中学习之前不常用的数据结构和算法，也是对自己解决问题能力的很大提升以及思路的很好扩展。

• 接触一门新的编程语言或没有使用过的函数库的时候，主要依赖于自学，通过读文档、读代码实例、读博客论坛来快速上手学习，并且如果希望能长期使用并对它们有深入的理解，最好通过读文档的方式，文档是最好的参考资料，最后一次给我们讲座的老师说的一句话令我印象深刻，“你遇到的问题绝大多数都是因为对文档的快速阅读造成的。文档中的每句话一定都是有用的，遇到问题应该首先去仔细的回顾文档”。并且，首先去学习底层的方法是更好的选择，因为第三方函数库的编写都是依赖于底层代码的，以后实际的工作中如果需要自行编写封装函数库的话，还是一定要调用底层代码的。所以加深对底层和基础的理解一定是有价值的。

• 计算机专业的内容繁杂丰富，很多时候都是为了某个需求或者作业要临时去学习一项新内容，但可能使用完这一次后很长时间都不会再去用了，学的快，学的肤浅，忘得也快，只能说有所了解。这时候就体现出来笔记的重要性。例如无论使用编程语言或函数库做的任务复杂与否，一般配环境、下载安装等流程都是少不了的，而正是这些流程中可能会遇到很多奇奇怪怪的问题，这是由于第一次接触、各方面知识都不了解导致的。但这些问题又都是基本的，只需要有了第一次成功的经验后面就会顺利很多。所以，把第一次做这些事情的时候遇到的坑、解决的方法、搜索的不同的解决方案等记录下来，既是经验的积累，给自己下次用到的时候提供参考，也可以发到网上论坛、博客等等，给其他的学习者提供参考。知识要交流起来才更丰富，学习要交流起来才更能提高效率。

• LeetCode 刷题的学习还要坚持进行下去，日积月累，积少成多。就像如果不是这一次布置了将之前的打卡任务汇总起来的作业，我也没有意识到这一个月不知不觉已经写了不算少的内容。这都是对学习过程和知识的总结，这种记录是很有价值很有意义的。希望能够养成这种习惯，持续练习并保持记录下去。也希望这一份汇总文档能成为一个标志，成为我学习生涯的一个新的开始。