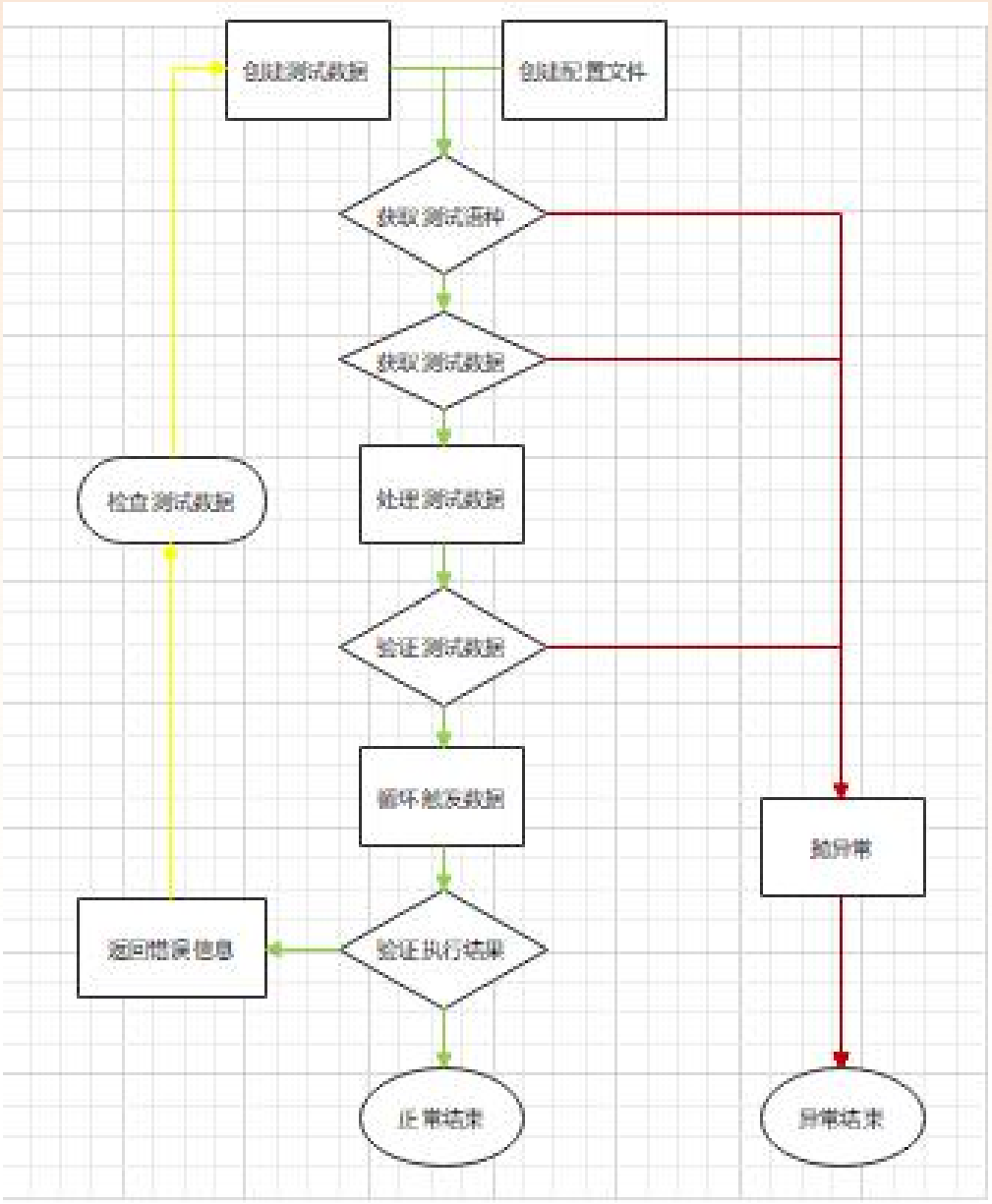
多语言

- 流程
- 整理
- 处理
- 触发
- 验证
- 总结

流程

1. 整理
   - 整理思路，创建测试数据
2. 处理
   - 处理测试数据，获得想要的数据
3. 触发
   - 依赖三方库触发测试数据
4. 验证
   - 获取执行结果进行验证



整理

裸键盘输入：瑞典语基础布局														
按键第一排	第一个	第二个	第三个	第四个	第五个	第六个	第七个	第八个	第九个	第十个	第十一个	第十二个	第十三个	第十四个
中文键盘	[`]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[0]	[-]	[=]	Backspace
瑞典语键盘（直接点击）	§	1	2	3	4	5	6	7	8	9	0	+	´	Backspace
Unicode														
瑞典语键盘（长按Shift）	½	!	"	#	¤	%	&	/	(	)	=	?	`	Backspace
Unicode														
瑞典语键盘（开启Capslock）	§	1	2	3	4	5	6	7	8	9	0	+	´	Backspace
Unicode														
瑞典语键盘（长按右Alt）			@	£	$	€		{	[	]	}	\		Backspace
Unicode														

§ 2.0.2 获取

首先，特殊按键在 `pynput.keyboard.Key` “模块”中可以直接找到。
比如 `ctrl` 对应 `pynput.keyboard.Key.ctrl` 还有 `.ctrl_l` 以及 `.ctrl_r` 。

然后，普通按键可以通过 `pynput.keyboard.KeyCode.from_char` 取得（特殊按键不可以，使用时会出现ArgumentError）。
如 `a` 可以运行 `pynput.keyboard.KeyCode.from_char('a')` 获得。

二者都可以用 `pynput.keyboard.KeyCode.from_vk` 通过按键的映射码取得。

- 根据微软输入法提前了解各类语言合适的上屏方式，方便后期验证；
- 根据产品提供布局表格了解各类语言的不同布局及映射结果；
- 不管各类语言对于物理键盘均是对应的四排按键映射后的结果。
- 根据三方库pynput的特性&按键映射码创建默认数据&测试数据。（y 89 XA59）

处理

- 获取语言、布局、上屏方式与测试数据
  1. 依赖configparser三方库操作ini文件，获取语言序号、布局、上屏方式；
  2. 根据语言序号获取该语言的整体数据列表；

```
hanyu = ['《韩语》数据',
        [[{'`': '192', '1': '49', '2': '50', '3': '51', '4': '52', '5': '53', '6': '54', '7': '55', '8': '56', '9': '57', '0': '48', '-': '189', '=': '187'},
            {'~': '192', '!': '49', '@': '50', '#': '51', '$': '52', '%': '53', '^': '54', '&': '55', '*': '56', '(': '57', ')': '48', '_': '189', '+': '187'},
            {'`': '192', '1': '49', '2': '50', '3': '51', '4': '52', '5': '53', '6': '54', '7': '55', '8': '56', '9': '57', '0': '48', '-': '189', '=': '187'}],
           [{'ㅂ': '81', 'ㅈ': '87', 'ㄷ': '69', 'ㄱ': '82', 'ㅅ': '84', 'ㅛ': '89', 'ㅕ': '85', 'ㅑ': '73', 'ㅐ': '79', 'ㅔ': '80'},
            {'ㅃ': '81', 'ㅉ': '87', 'ㄸ': '69', 'ㄲ': '82', 'ㅆ': '84', 'ㅛ': '89', 'ㅕ': '85', 'ㅑ': '73', 'ㅐ': '79', 'ㅔ': '80'},
            {'ㅂ': '81', 'ㅈ': '87', 'ㄷ': '69', 'ㄱ': '82', 'ㅅ': '84', 'ㅛ': '89', 'ㅕ': '85', 'ㅑ': '73', 'ㅐ': '79', 'ㅔ': '80'}],
           [{'ㅁ': '65', 'ㄴ': '83', 'ㅇ': '68', 'ㄹ': '70', 'ㅎ': '71', 'ㅗ': '72', 'ㅓ': '74', 'ㅏ': '75', 'ㅣ': '76'},
            {'ㅁ': '65', 'ㄴ': '83', 'ㅇ': '68', 'ㄹ': '70', 'ㅎ': '71', 'ㅗ': '72', 'ㅓ': '74', 'ㅏ': '75', 'ㅣ': '76'},
            {'ㅁ': '65', 'ㄴ': '83', 'ㅇ': '68', 'ㄹ': '70', 'ㅎ': '71', 'ㅗ': '72', 'ㅓ': '74', 'ㅏ': '75', 'ㅣ': '76'}],
           [{'ㅋ': '90', 'ㅌ': '88', 'ㅊ': '67', 'ㅍ': '86', 'ㅠ': '66', 'ㅜ': '78', 'ㅡ': '77'},
            {'ㅋ': '90', 'ㅌ': '88', 'ㅊ': '67', 'ㅍ': '86', 'ㅠ': '66', 'ㅜ': '78', 'ㅡ': '77'},
            {'ㅋ': '90', 'ㅌ': '88', 'ㅊ': '67', 'ㅍ': '86', 'ㅠ': '66', 'ㅜ': '78', 'ㅡ': '77'}]],
        ['基础布局', 'shift布局', 'capslock布局'], 32, False]
```

- 处理数据集合重整为预期的数据列表
  1. 将该语言各布局下对应的四排按键基于布局进行归类

```
'[2021-08-04 19:45:16.995314] [INFO] [handle_keyboard.py] [key_mapping] [18] 布局：基础布局'
buju = [[{'`': '192', '1': '49', '2': '50', '3': '51', '4': '52', '5': '53', '6': '54', '7': '55', '8': '56', '9': '57', '0': '48', '-': '189', '=': '187'},
        {'ㅂ': '81', 'ㅈ': '87', 'ㄷ': '69', 'ㄱ': '82', 'ㅅ': '84', 'ㅛ': '89', 'ㅕ': '85', 'ㅑ': '73', 'ㅐ': '79', 'ㅔ': '80'},
        {'ㅁ': '65', 'ㄴ': '83', 'ㅇ': '68', 'ㄹ': '70', 'ㅎ': '71', 'ㅗ': '72', 'ㅓ': '74', 'ㅏ': '75', 'ㅣ': '76'},
        {'ㅋ': '90', 'ㅌ': '88', 'ㅊ': '67', 'ㅍ': '86', 'ㅠ': '66', 'ㅜ': '78', 'ㅡ': '77'}],
       [{'~': '192', '!': '49', '@': '50', '#': '51', '$': '52', '%': '53', '^': '54', '&': '55', '*': '56', '(': '57', ')': '48', '_': '189', '+': '187'},
        {'ㅃ': '81', 'ㅉ': '87', 'ㄸ': '69', 'ㄲ': '82', 'ㅆ': '84', 'ㅛ': '89', 'ㅕ': '85', 'ㅑ': '73', 'ㅐ': '79', 'ㅔ': '80'},
        {'ㅁ': '65', 'ㄴ': '83', 'ㅇ': '68', 'ㄹ': '70', 'ㅎ': '71', 'ㅗ': '72', 'ㅓ': '74', 'ㅏ': '75', 'ㅣ': '76'},
        {'ㅋ': '90', 'ㅌ': '88', 'ㅊ': '67', 'ㅍ': '86', 'ㅠ': '66', 'ㅜ': '78', 'ㅡ': '77'}],
       [{'`': '192', '1': '49', '2': '50', '3': '51', '4': '52', '5': '53', '6': '54', '7': '55', '8': '56', '9': '57', '0': '48', '-': '189', '=': '187'},
        {'ㅂ': '81', 'ㅈ': '87', 'ㄷ': '69', 'ㄱ': '82', 'ㅅ': '84', 'ㅛ': '89', 'ㅕ': '85', 'ㅑ': '73', 'ㅐ': '79', 'ㅔ': '80'},
        {'ㅁ': '65', 'ㄴ': '83', 'ㅇ': '68', 'ㄹ': '70', 'ㅎ': '71', 'ㅗ': '72', 'ㅓ': '74', 'ㅏ': '75', 'ㅣ': '76'},
        {'ㅋ': '90', 'ㅌ': '88', 'ㅊ': '67', 'ㅍ': '86', 'ㅠ': '66', 'ㅜ': '78', 'ㅡ': '77'}]]
```

触发

- § 2.2.1

先获取控件： `ctr = pynput.keyboard.Controller()` 。

通过方法 `press` **来按下按键，**

你需要提供一个“*长度为1的字符*”或是“*前文所说的按键对象*”。

通过 `release` **释放按键，**

和press方法一样，需要提供一个“*长度为1的字符*”或是“*前文所说的按键对象*”。

```
self.ctr.press(pynput.keyboard.KeyCode.from_vk(int(values)))
self.ctr.release(pynput.keyboard.KeyCode.from_vk(int(values)))
```

`pressed` 方法就是我说的“更简单、优雅”的方法。

使用时提供要按下的键，再用 `with` 语句“封装”上。

效果是**进入语句块时***顺序*按下提供按键，**退出语句块时***逆序*释放按键。

```
import pynput, time
ctr = pynput.keyboard.Controller()
with ctr.pressed(pynput.keyboard.Key.ctrl,pynput.keyboard.Key.shift,'s'):
    pass
with ctr.pressed(pynput.keyboard.Key.esc):
    pass
```

- 了解各功能按键与特殊按键的操作逻辑：按下&释放
- 针对各布局下功能按键与特殊按键搭配组合
- 获取各布局下需要验证的功能按键
- 通过pynput的keyboard.Controller操作各按键配合上屏方式上屏内容

验证

```
def get_txt(self,key,values):
    '''获取文件内容，进行比对校验'''
    Handle_txt().save_data()
    Handle_txt().clear_data()
    with open('ceshi.txt','r',encoding='utf-8') as f:
        data = f.read()
        default = Testdata().default_data()
        if key == data.split(' ')[0]:
            i = [i for i, j in default.items() if values == j]
            print('键盘：{0} 键映射正常：预期结果：{1} -> 实际结果{2}'.format(i,key,data))
            # logger.info('键盘：{0} 键映射正常：预期结果：{1} -> 实际结果{2}'.format(i,key,data))

        else:
            i= [i for i, j in default.items() if values == j]
            logger.error('键盘{0} 键映射异常：预期结果{1} -> 实际结果{2}'.format(i, key, data))
```

- 上屏后，保存并获取文件内容，获取后清空
- 处理文件内容为长度1的字符串，返回屏上结果
- 验证屏上结果与预期结果是否一致
  - 一致：判断硬编码对应的物理按键，返回验证成功
  - 不一致：判断硬编码对应的物理按键，输出该按键的异常信息

总结

新增语言时：

- 在test_data ＞ data.py 文件内新建该语言的测试数据；
- 在Config ＞ config.ini 文件内新建该语言的布局类型及上屏方式；
- 在Util ＞ Data_language.py 文件内新增判断用于获取该语言测试数据
- 运行

问题：

- 字符型的list、dict、tuple转换为原格式类型
  - ast.literal_eval()可以解决
- 获取文件内容
  - 按正常逻辑应为：保存-获取-删除，但脚本中保存-删除-获取 的结果却是正确的?
- 按键未释放
  - 先确保熟悉各按键的操作逻辑，有按下必须有释放，否则就要重启电脑恢复按键状态
- 上屏结果与预期结果不一致
  - 特殊语言的某些按键硬编码在该语言下是经过修改的；
  - 特殊语言的某些状态致使上屏错误，例日语罗马全角片假名是因为预上屏状态时候选的问题影响
- 支持系统
  - 暂仅支持windows系统运行，linux运行因某些原因会报X11段错误，初步定位.from_vk()操作特殊按键导致

Ending ...