

# 项目设计报告

项目名称：英语词汇量估算工具

专业：网络工程

成 员：杨舒越 项目组长

## 完成情况

	姓名	身份	任务	任务占比
	杨舒越	组长	估计算法、评价方法的设计	25%
	杨舒越	组员	编程实现，测试。	25%
	杨舒越	组员	数据库设计，界面实现	25%
	杨舒越	组员	总体设计与实现	25%

已实现基本功能： 词汇量估算功能，后台验证准确率功能，GUI 演示以及批处理功能

额外实现功能（扩展功能）：将用户测试结果发送至后台数据库功能

自评主要亮点：算法设计合理，平滑的梯度划分使代码具有稳定性

自评主要缺陷：代码应进一步优化

## 一、项目设计目的

1. **教育辅助**: 本项目设计从解决相关应用领域实际工程问题出发进行综合实践,帮助学习者了解自己的词汇掌握程度,从而更有针对性地进行学习。
2. **个性化学习计划**: 根据词汇量的结果,为学习者推荐适合其水平的学习材料和练习。
3. **进度跟踪**: 使学习者能够追踪自己的学习进度,看到自己的成长和需要改进的地方。
4. **自我评估**: 提供一个方便快捷的方式,让学习者可以自我评估,而不必依赖于教师或第三方评估。
5. **激励学习**: 通过量化词汇量,激发学习者的学习动力,增加学习的乐趣。
6. **语言能力认证**: 为学习者提供一个参考标准,帮助他们了解自己的英语水平是否达到了某种考试或认证的要求。

## 二、项目设计内容

- 设计题目：按照软件工程思路设计 英语词汇量估算工具；

提交内容：

数据（如词汇表等）、算法思路、具体设计文档（报告）、代码等；

有实际创新加分

有扩展功能加分

分组责任参考（1-8 人）：

总体设计；

算法设计（主要是词汇量测试算法和验证方法）；

前端选择和 UI 设计（web、桌面程序、app、小程序等都可以）。

简单数据库选择和设计；（不限定数据库）

演示测试：两种测试，一个 GUI 演示测试，一个是后台批处理测试。

● 主要功能：

收集词汇列表等不同辅助数据，设计一至多种用户词汇量估算算法；

设计验证方法：即 衡量你的算法，估算出来的词汇量到底有多准确？

可与业内产品做比较比如：

<http://testyourvocab.com/> （首选）

百词斩词汇测试

扇贝单次词汇测试

界面设计：可用 web 页面、桌面程序、app、小程序等。

后台批处理测试结果

可考虑 输入一个单词列表，直接算法后台计算结果

输入列表格式：词 A， 认识； 词 B， 认识； 词 C， 不认识；词 D， 不认识；.....。

输出结果：估算词汇量

界面实例测试结果

找不同学生，报告测试结果

主要数据：学号姓名（如有隐私考虑代号）、四级成绩、六级成绩、测试时间、测试结果。

● 扩展功能：

辅助数据和估算算法程序可以根据不同考虑放在服务器端或者客户端；

可用服务器端的数据、算法更新客户端相应的数据、算法；

**发送学生测试实例结果到服务器端数据库**

主要数据：学号姓名（如有隐私考虑代号）、四级成绩、六级成绩、测试时间、测试结果。

### 三、项目环境要求

1、硬件需求：普通 PC

2、软件需求：系统：Windows 系统或 linux 系统（Ubuntu）

3、开发工具：不限

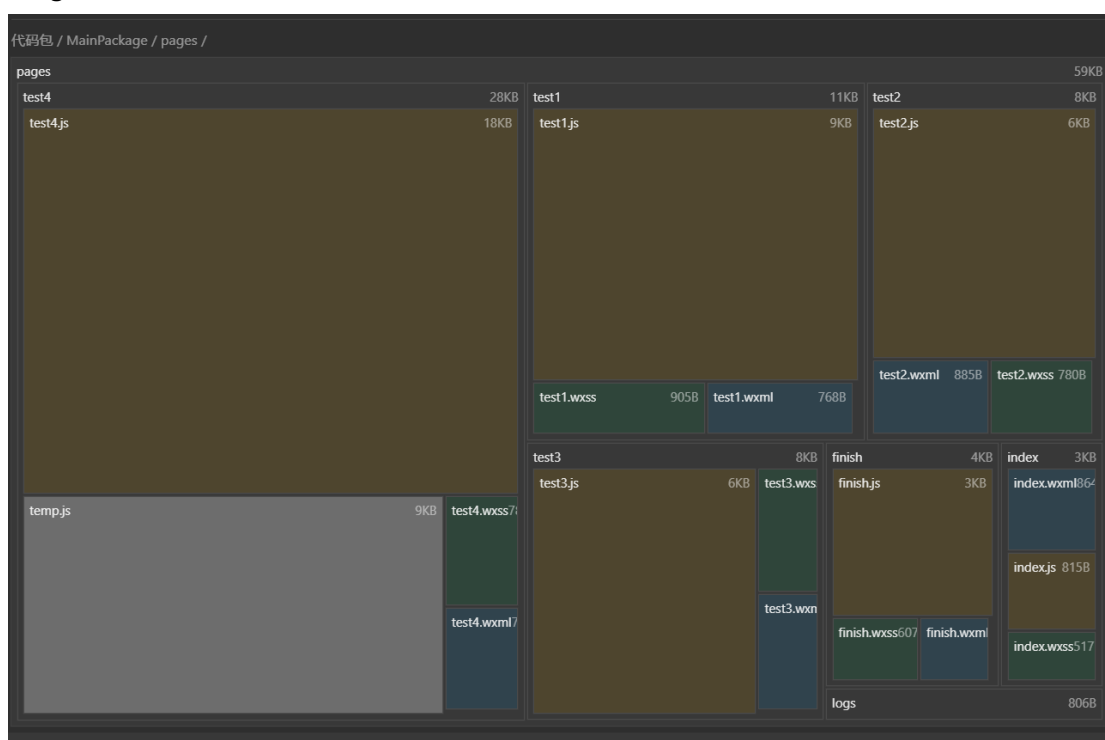
4、开发语言：不限

### 四、功能算法设计（模块设计、估计算法、评价方法）

该程序主要分为 5 个板块：

## 程序的前端 UI 设计与框架设计

1. 该程序选用微信小程序，使用微信开发者工具进行开发，同时使用了部分@vant 组件来实现复选框以及按钮的 UI
2. 该程序共有 5 个页面组成：一个 index page 四个 test page 以及一个 finish page 分别用于实现页面索引，梯度筛选以及最后的结果展示
3. Index page: 用于接收测试者姓名保存后并选择点击跳转到测试页面  
Test1 page: 用于调试参考用，没有实际用处  
Test2-4 page: 给出一定复选框使测试者选择认识的单词，并通过算法计算每次筛选的梯度范围从而进行更加精确的梯度筛选
4. Page 页面代码依赖关系



Index 页面核心代码:

```
// 跳转到 Test2 页面并保存数据
navigateToTest2() {
  if (this.saveName()) {
    wx.navigateTo({
      url: '/pages/test2/test2'
    });
  }
}
});
```

调用 `savename` 成功后跳转至 `test2` 页面

## Test2-4 核心代码:

```
Page({
  data: {
    gradientRange: [],
    checkboxes: [
      //由于范围更加精确，因此无需消除大权重的影响，直接计算平均权重即可
      //total=120
      [{ label: 'Option 1', checked: false ,weight:2 }, { label: 'Option 2', checked: false ,weight:2 }],
      [{ label: 'Option 3', checked: false ,weight:2 }, { label: 'Option 4', checked: false ,weight:2 }],
      [{ label: 'Option 5', checked: false ,weight:2 }, { label: 'Option 6', checked: false ,weight:2 }],
      [{ label: 'Option 7', checked: false ,weight:3 }, { label: 'Option 8', checked: false ,weight:3}],
      [{ label: 'Option 9', checked: false ,weight:3 }, { label: 'Option 10', checked: false ,weight:3}],
      [{ label: 'Option 11', checked: false ,weight:3 }, { label: 'Option 12', checked: false ,weight:3}],
      [{ label: 'Option 13', checked: false ,weight:4 }, { label: 'Option 14', checked: false ,weight:4}],
      [{ label: 'Option 15', checked: false ,weight:4 }, { label: 'Option 16', checked: false ,weight:4}],
      [{ label: 'Option 17', checked: false ,weight:4 }, { label: 'Option 18', checked: false ,weight:4}],
      [{ label: 'Option 19', checked: false ,weight:5 }, { label: 'Option 20', checked: false ,weight:5}],
      [{ label: 'Option 21', checked: false ,weight:5 }, { label: 'Option 22', checked: false ,weight:5}],
      [{ label: 'Option 23', checked: false ,weight:5 }, { label: 'Option 24', checked: false ,weight:5}],
      [{ label: 'Option 25', checked: false ,weight:6 }, { label: 'Option 26', checked: false ,weight:6}],
      [{ label: 'Option 27', checked: false ,weight:6 }, { label: 'Option 28', checked: false ,weight:6}],
      [{ label: 'Option 29', checked: false ,weight:6 }, { label: 'Option 30', checked: false ,weight:6}],
      //[{ label: 'Option 31', checked: false ,weight:3 }, { label: 'Option 32', checked: false ,weight:3}],
    ],
  },
},
```

设置复选框

```
77 >   getData() {
78     const db = wx.cloud.database();
79     const gradientRange = this.data.gradientRange;
80
81 >     const fetchWordData = (collection, size) => { ...
82     };
83
84     const wordPromises = gradientRange.map(gradient => fetchWordData(`word${gradient-1}`, 6)); //gradient-1
85
86     Promise.all(wordPromises).then(results => {
87 >       const options = results.flat().map((word, index) => ({
88         label: word,
89         checked: false
90       }));
91
92 >       if (options.length < 30) { ...
93       }
94
95       console.log('获取的选项数据:', options); // 添加调试信息
96
97 >       this.setData({ ...
98       });
99
100       console.log('复选框数据更新成功:', this.data.checkboxes); // 添加调试信息
101 >     }).catch(err => { ...
102     });
103   },
104 }
```

GetData()函数用于从数据库中获取单词数据填入复选框

Finish 页面核心代码:

```
1  onLoad: function (options) {  
2      if (options.score) {  
3          const score = options.score;  
4          console.log('接收到的 score:', score);  
5          this.setData({  
6              words: score,  
7          });  
8          if (options.rate) {  
9              const rate = options.rate;  
10             console.log('接收到的 rate:', rate);  
11             this.setData({  
12                 rates: rate * 100,  
13             });  
14         }  
15     }  
16 }
```

接收传递的 score 和 rate 参数并显示

## 词汇数据的提取与分类

1. 词汇来源于语料库 [https://www.wordfrequency.info/samples/words\\_219k.txt](https://www.wordfrequency.info/samples/words_219k.txt)，用现有的词频文件

	A	B	C	D	E	F
	rank	word	freq	#texts	%caps	blog
8	14365	kavanaugh	3363	493	1.00	
9	14375	dexter	3360	1137	0.97	
0	14385	darkest	3355	2976	0.06	
1	14395	cantor	3352	1046	0.93	
2	14405	mcveigh	3350	609	0.99	
3	14415	contingency	3348	2214	0.07	
4	14425	playboy	3346	2058	0.77	
5	14435	override	3342	2360	0.04	
6	14445	susie	3339	1111	0.98	
7	14455	murdoch	3336	1206	1.00	
8	14465	blames	3332	2992	0.03	
9	14475	amidst	3328	2835	0.15	
0	14485	moderates	3324	2167	0.05	
1	14495	hoc	3320	2106	0.11	
2	14505	heavyweight	3316	1903	0.10	
3	14515	libertarians	3314	1322	0.25	
4	14525	comrades	3310	2389	0.08	
5	14535	ramifications	3307	2966	0.02	
6	14545	encyclopedia	3303	2237	0.54	
7	14555	stash	3300	2665	0.08	
8	14565	rosenberg	3296	1345	0.99	
9	14575	modernization	3292	1800	0.10	
0	14585	iceland	3287	1419	0.99	
1	14595	wrinkles	3284	2494	0.05	
2	14605	hustle	3279	2474	0.21	

2. 用 python 算法将词频分类为 20 个等级构成合理的难度梯度，并从每个梯度随机抽取 300 个词汇组成测试梯度

核心代码：

```
levels["级别5-1"].append(word)
elif 40 < freq <= 53:
    levels["级别5-2"].append(word)
elif 31 < freq <= 40:
    levels["级别6-1"].append(word)
elif 24 < freq <= 31:
    levels["级别6-2"].append(word)
elif 20 < freq <= 24:
    levels["级别7-1"].append(word)
elif 17 < freq <= 20:
    levels["级别7-2"].append(word)
elif 14 < freq <= 17:
    levels["级别7-3"].append(word)
elif 12 < freq <= 14:
    levels["级别7-4"].append(word)
elif 10 < freq <= 12:
    levels["级别8-1"].append(word)
elif 9 < freq <= 10:
    levels["级别8-2"].append(word)
```

```
# 从每个级别中随机抽取300条数据
random_samples = defaultdict(list)
sample_size = 400

for level in levels:
    if len(levels[level]) > sample_size:
        random_samples[level] = random.sample(levels[level], sample_size)
    else:
        random_samples[level] = levels[level]
```

生成后词汇数据

```
26    nor
27    open
28    between
29    family
30    else
31    poor
32    fine
33    care
34    kind
35    read
36    answered
```

3. 将生成的词汇文本转化为 json 文件，以便导入微信云数据库

核心代码：

```

convert.py > ...
1  import json
2
3  def convert_txt_to_json(txt_file, json_file):
4      with open(txt_file, 'r', encoding='utf-8') as file:
5          words = file.read().splitlines()
6
7          json_data = [{"_id": f"n{i+8}", "headword": word} for i, word in enumerate(words)]
8
9          with open(json_file, 'w', encoding='utf-8') as file:
10             for entry in json_data:
11                 file.write(json.dumps(entry, ensure_ascii=False) + "\n")
12
13 # 遍历并转换 word0.txt 到 word20.txt
14 for i in range(21):
15     txt_file = f"word{i}.txt"
16     json_file = f"word{i}.json"
17     convert_txt_to_json(txt_file, json_file)
18
19 print("文件转换完成。")
20

```

生成后的 json 文件

```

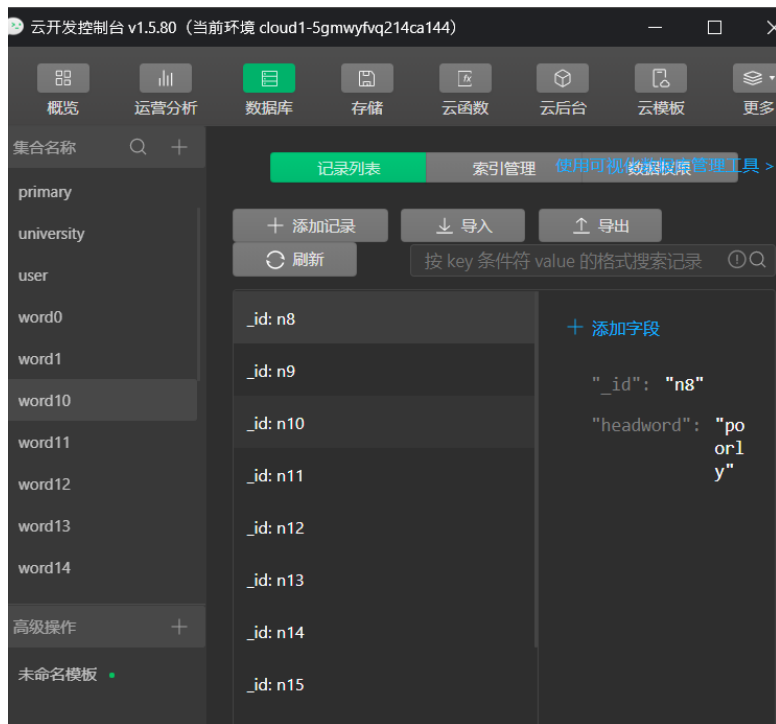
15 [{"_id": "n22", "headword": "either"}]
16 {"_id": "n23", "headword": "side"}
17 {"_id": "n24", "headword": "laughed"}
18 {"_id": "n25", "headword": "feet"}
19 {"_id": "n26", "headword": "hard"}
20 {"_id": "n27", "headword": "red"}
21 {"_id": "n28", "headword": "lay"}
22 {"_id": "n29", "headword": "carriage"}
23 {"_id": "n30", "headword": "high"}
24 {"_id": "n31", "headword": "since"}
25 {"_id": "n32", "headword": "neither"}
26 {"_id": "n33", "headword": "nor"}
27 {"_id": "n34", "headword": "open"}
28 {"_id": "n35", "headword": "between"}
29 {"_id": "n36", "headword": "family"}

```

## 数据库的创建与实现

1. 首先创建单词集合：





在集合中创建集合 word0-19 并导入生成的 json 文件用于获取单词数据

## 2. 将用户数据上传后台数据

核心代码：

```
addUserToDatabase(username, finalValue) {
  const db = wx.cloud.database();
  // 使用云开发插入数据的方法
  db.collection('user').add({
    data: {
      username: username,
      final: finalValue,
      timestamp: db.serverDate() // 添加时间戳
    },
    success: res => {
      //console.log('用户数据存入数据库成功', res);
    },
    fail: err => {
      console.error('用户数据存入数据库失败', err);
    }
  });
},
```

### 3. 微信云开发的初始化工作

在主文件夹的 app.js 中初始化云函数

```
logs.unshift(Date.now())

wx.setStorageSync('logs', logs)

//初始化云开发
wx.cloud.init({
  env: 'cloud1-5gmwyfvq214ca144' //这里输入云开发id
})

// 云云
```

并在调用数据库时使用

```
Const db = wx.cloud.database()
```

调用

## 测试算法的设计与实现

算法使用梯度下降法来确定测试者所在合理区间 具体算法如下：

①在页面一从划分的 20 个梯度各随机取两个词汇组成 40 个复选框，并为每个梯度的复选框附上不同的权值，并且难度越大的梯度权值越大（但是要注意平衡大权值带来的影响），在测试者勾选完第一个页面的复选框后便利用公式计算平均权值来计算下一个梯度区间的范围

其中梯度 1-20 的复选框赋值为 1-20

平均权值计算公式如下：

$$\text{weightsum} = \text{weightsum} + \ln(\text{weight} + 1)$$

其中只计算由测试者勾选的复选框的权重值累加，采用  $\ln$  函数是为了消除大权值带来的不平衡影响， $\text{weight}+1$  是为了防止  $\ln$  函数内部为 0

因此可计算得出总权值 totalsum 值为

$$2 * \ln(21!) = 90.760277796953816052320947902151$$

因此由此可以将测试者的权值平均划分为 16 个区间，每个区间由 5 个更为精确的梯度构成  
划分算法如下

```

const total = 90.760277796953816052320947902151;
// 总权重值
// 设置梯度范围
let gradientRange = [];
const gradientSteps = 16;
for (let i = 1; i <= gradientSteps; i++) {
  if (totalWeight <= total * (i / gradientSteps)) {
    gradientRange = [i, i + 1, i + 2, i + 3, i + 4];
    break;
  }
}

```

由此可以计算出下一梯度的范围区间，并将 `gradient` 数组传递给下一个页面

```

wx.navigateTo({
  url: `/pages/test3/test3?gradientRange=${JSON.stringify(gradientRange)}`,
});
},

```

同时保存这个梯度的正确率 `rate1`

②在下一个页面中，将 `gradient` 数组范围作为接收数据的数据库来源并重复选择操作  
共有 30 个复选框，从 5 个梯度中各选 6 个数据组成，每个梯度赋值为 2-6  
核心代码实现如下：

```

getData() {
  const db = wx.cloud.database();
  const gradientRange = this.data.gradientRange;

  const fetchWordData = (collection, size) => { ...
};

const wordPromises = gradientRange.map(gradient => fetchWordData(`word${gradient-1}`, 6)); //gradient-1

Promise.all(wordPromises).then(results => { ...
}).catch(err => { ...
});
},

```

测试完成后计算平均权值

由于只有 5 个梯度，因此无需考虑极大权值带来的不平衡影响，因此直接计算平均权值即可

TotalWeight = 120

计算公式为：

```
const gradientRange = this.data.gradientRange;
let final = '';
const total=120; //经过平滑处理后的总权值
if (totalWeight < total*(1/5)) {
    final = gradientRange[0];
} else if (totalWeight < total*(2/5)) {
    final = gradientRange[1];
} else if (totalWeight < total*(3/5)) {
    final = gradientRange[2];
} else if (totalWeight < total*(4/5)) {
    final = gradientRange[3];
} else {
    final = gradientRange[4];
}
```

得出的 **final** 值即为最后一个梯度

同时保存这个梯度的正确率 **rate2**

③在最后一个梯度同样设置 40 个复选框选项给测试者测试，复选框数据从 **word{final-1}** 获取，测试结束后保存这个梯度的正确率 **rate3**

核心代码：

```
console.log('weightSum3',weightSum)
let total=40,
rate=weightSum/total;
console.log('rate3',rate);//得出正确率: 最终结果
},
```

④根据增长模型模拟不同梯度的词汇量递增水平来模拟不同的 **final** 梯度对应词汇量，并将其赋为每个 **final** 的对应权重

结合计算出的 **rate1,rate2,rate3** 以及 **final** 对应的权重值计算出估计词汇量

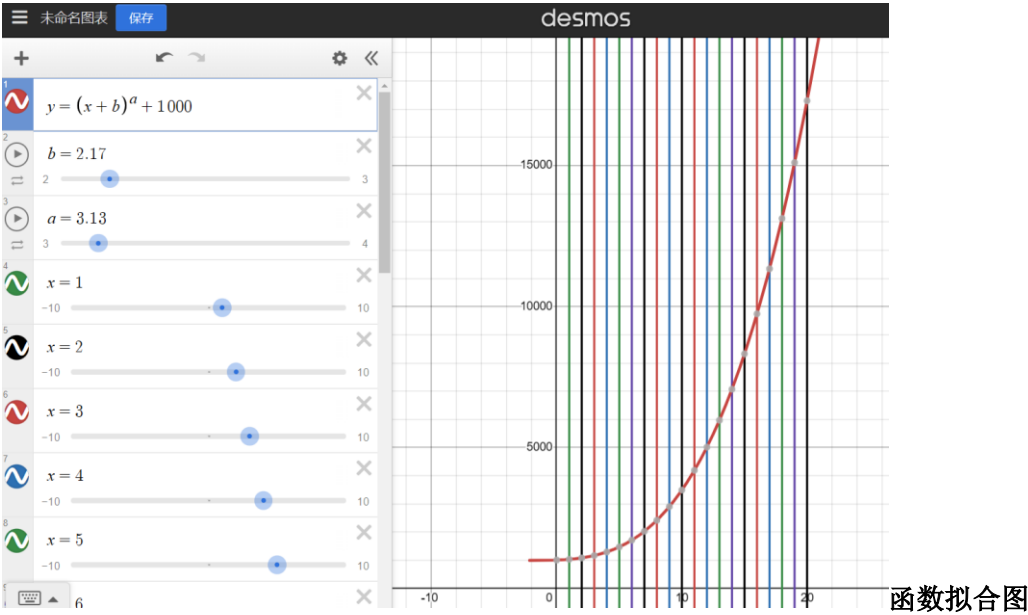
增长模型为函数拟合，拟合后得知函数模型

$$f(x) = (x + b)^a + 1000$$

较为符合实际情况的拟合，其中取 a=3.13, b=2.17  
则

$$f(x) = (x + 2.17)^{3.13} + 1000$$

近似为梯度与权重值的对应关系



因此可以求出对应权重值  
如下表：

Final	1	2	3	4	5
Weight	1011.301	1037.01	1087.302	1171.09	1297.572
Final	6	7	8	9	10
Weight	1716.564	2028.528	2422.049	2907.245	3494.362
Final	11	12	13	14	15
Weight	4193.769	5015.944	5971.471	7071.03	8325.395
Final	16	17	18	19	20
Weight	9475.425	11342.064	13126.332	15109.327	17302.215

1.于是可以计算得出 rate1 位于的 gradient1 的对应权重 weight-gradient1=wg<sub>1</sub>

$$\frac{\sum_{i=1}^{20} \text{Weight}_i}{20} = 5755 = w_{g1}$$

这里为 rate1 赋予权重系数  $w_{r1} = 0.2$ ,  $w_i$  代表第 i 梯度的可信度, 越接近 1 可信度越高, 因为 rate1 为一轮筛选梯度, 不具备精确性, 因此可信度较低, 赋值为 0.3

2. 同时由于 rate2 对应的梯度为动态变换的, 因此 gradient2 的对应权重 weight-gradient2 是动态变化的

计算公式为 
$$\frac{\sum_{i=1}^5 \text{Weight}_i}{5} = w_{g2}$$

例如 gradient 数组值为 [5 6 7 8 9], 那么  $w_{g2}$  计算结果为 2074

例如 gradient 数组值为 [5 6 7 8 9], 那么  $w_{g2}$  计算结果为 13271

这里为 rate2 赋予权重系数  $w_{r2} = 0.5$ , 代表这个区间的结果 ‘中等可信’

3. 最后 rate3 对应的梯度即为最终确定的梯度 final, 因此 gradient3 的对应权重 weight-gradient3 等于 final 对应的 weight 值

$$f(\text{final}) = w_{g3}$$

这里为 rate3 赋予权重系数  $w_{r3} = 0.9$ , 代表这个区间的结果 ‘几乎可信’

因此可以得出最终词汇量测试结果计算公式为

$$\text{score} = r_1 \cdot w_{g1} \cdot w_{r1} + r_2 \cdot w_{g2} \cdot w_{r2} + r_3 \cdot w_{g3} \cdot w_{r3}$$

其中  $r_i$ ,  $w_g$ ,  $w_r$  分别对应三个测试大梯度的

正确率(rate), 梯度权重(weight-gradient), 梯度对应权重系数(weight-ratio)

## 验证算法的准确性实现

实现方式有两种：

- ① 将词库的词汇按照频率分布从不同的频率区间提取词汇，组成不同水平的词汇量是一种模拟词汇量思想

核心代码：

```
def main():
    # 示例使用
    file_path = 'words_frequency_2.txt' # 词汇频率文件路径
    word_freq = load_word_frequencies(file_path)

    # 定义频率区间及每个区间抽取的词汇数量
    intervals = [(0, 0.1), (0.1, 0.2), (0.2, 0.25), (0.25, 0.6), (0.6, 1.0)]
    samples_per_interval = [1258, 1258, 0, 0, 0]
    #考虑到样本词汇没有相应词根词缀的联想词，所以数量适当减少

    sampled_words = sample_words(word_freq, intervals, samples_per_interval)

    # 输出抽取的词汇样本并保存至文件
    with open('2500-3500.txt', 'w', encoding='utf-8') as f:
        for word, freq in sampled_words:
            f.write(f"{word} {freq}\n")
            #print(word, freq)

    # 输出每个区间的词汇数
    for i, (start, end) in enumerate(intervals):
        interval_size = int(len(word_freq) * (end - start))
        print(f"区间 {i+1} ({start*100}%-{end*100}%): {interval_size} 个词")

if __name__ == "__main__":
    main()
```

缺点：由于词汇量水平的定义过于主观，没有固定标准，因此不予采用

- ② 以行业内做的较好的产品做比较：例如 <https://preply.com/en/learn/english/test-your-vocab> 测试方式：利用 python 模拟这个网页的测试并且爬取测试结果，得出位于不同分数段的结果，该结果更具合理性

核心代码：

```

def run_test(driver_path, test_number):
    # 设置WebDriver路径
    service = Service(executable_path=driver_path)
    driver = webdriver.Chrome(service=service)

    # 导航到网页
    driver.get("https://preply.com/en/learn/english/test-your-vocab")

    # 等待页面加载
    sleep(5) # 根据需要调整等待时间

    # 定位所有复选框按钮
    checkboxes = driver.find_elements(By.CSS_SELECTOR, "button[role='checkbox']")
    words = {}

    # 记录每个复选框对应的词汇
    for checkbox in checkboxes:
        label = checkbox.find_element(By.XPATH, "./following-sibling::span")
        word = label.text
        words[checkbox] = word

    # 随机选择25个复选框进行点击
    selected_checkboxes = sample(list(words.keys()), min(25, len(words))) # 确保不会超过复选框的数量
    clicked_words = {}

    for checkbox in selected_checkboxes:
        try:
            driver.execute_script("arguments[0].click();", checkbox)
            clicked_words[words[checkbox]] = "会"
            sleep(0.5) # 随机等待, 模拟用户操作
        except Exception as e:
            print(f"测试 {test_number} - 点击时出错: ", e)

```

```

# 定位结果元素并打印其文本内容
result_css = "h3[data-preply-ds-component='Heading']"
try:
    result_element = driver.find_element(By.CSS_SELECTOR, result_css)
    result_text = result_element.text
    print(f"测试 {test_number} - Your vocabulary count is: {result_text}")
except Exception as e:
    print(f"测试 {test_number} - 定位结果元素时出错: ", e)

# 将未点击的词汇标记为不会
for checkbox, word in words.items():
    if word not in clicked_words:
        clicked_words[word] = "不会"

# 构建最终输出文本按列排放
output = [f'"{word}",{status}' for word, status in clicked_words.items()]
output.append(f'测试结果',{result_text})

output_text = "\n".join(output)

# 使用结果文本中的数字构建文件名
filename = f"stimulate_{result_text}.txt"

# 将结果写入指定文件名
with open(filename, "w", encoding="utf-8") as f:
    f.write(output_text)

# 关闭浏览器
driver.quit()

# 运行3个测试
driver_path = 'C:\\Program Files\\Google\\Chrome\\Application\\chromedriver.exe'

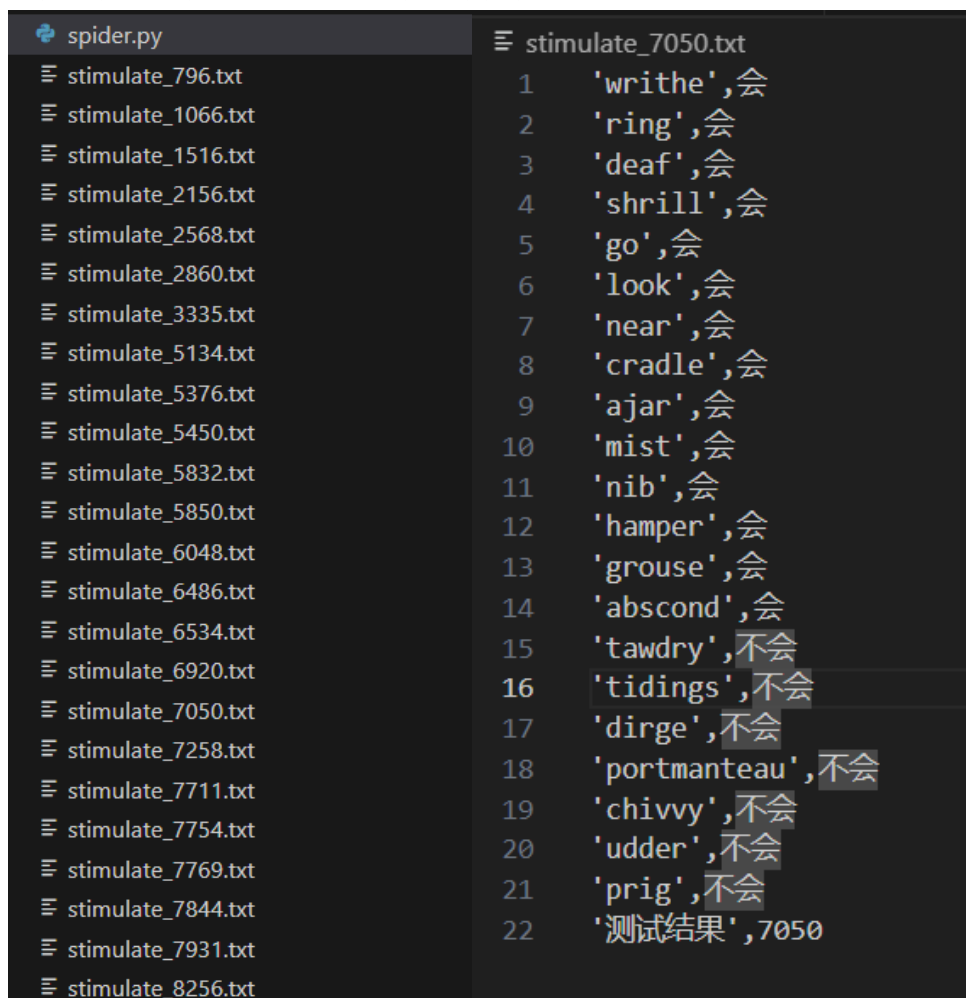
with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
    futures = [executor.submit(run_test, driver_path, i) for i in range(1,6)]

# 等待所有线程完成
concurrent.futures.wait(futures)

```



多次更改在两个页面勾选的复选框个数并得出多种测试结果



```
spider.py
stimulate_796.txt
stimulate_1066.txt
stimulate_1516.txt
stimulate_2156.txt
stimulate_2568.txt
stimulate_2860.txt
stimulate_3335.txt
stimulate_5134.txt
stimulate_5376.txt
stimulate_5450.txt
stimulate_5832.txt
stimulate_5850.txt
stimulate_6048.txt
stimulate_6486.txt
stimulate_6534.txt
stimulate_6920.txt
stimulate_7050.txt
stimulate_7258.txt
stimulate_7711.txt
stimulate_7754.txt
stimulate_7769.txt
stimulate_7844.txt
stimulate_7931.txt
stimulate_8256.txt

stimulate_7050.txt
1 'writhe',会
2 'ring',会
3 'deaf',会
4 'shrill',会
5 'go',会
6 'look',会
7 'near',会
8 'cradle',会
9 'ajar',会
10 'mist',会
11 'nib',会
12 'hamper',会
13 'grouse',会
14 'abscond',会
15 'tawdry',不会
16 'tidings',不会
17 'dirge',不会
18 'portmanteau',不会
19 'chivvy',不会
20 'udder',不会
21 'prig',不会
22 '测试结果',7050
```

至此，并得到了测试用数据

- ③ 将以上数据进行格式转换后导入微信云存储  
核心代码

```
def extract_words_from_files(directory):
    # 匹配所有以 stimulate_ 开头的 txt 文件
    file_paths = glob.glob(os.path.join(directory, 'stimulate_*.txt'))

    for file_path in file_paths:
        with open(file_path, 'r', encoding='utf-8') as file:
            lines = file.readlines()

            # 初始化空列表来存储符合条件的单词
            words_only = []

            # 遍历每一行内容, 排除最后一行
            for line in lines[:-1]: # 使用切片去掉最后一行
                parts = line.split(',')
                if len(parts) > 1 and parts[1].strip() == '会': # 判断第二部分是否为“会”
                    word = parts[0].strip() # 获取单词部分并去除空格和换行符
                    clean_word = re.sub(r'^\w\s', '', word) # 使用正则表达式去除单词中的符号
                    words_only.append(clean_word) # 将处理后的单词部分添加到列表中

            # 输出结果到相应的 txt 文件
            output_file_name = os.path.basename(file_path).replace('stimulate_', '').replace('.txt', '_output.txt')
            output_file_path = os.path.join(directory, output_file_name)

            with open(output_file_path, 'w', encoding='utf-8') as output_file:
                for word in words_only:
                    output_file.write(word + '\n')

            # 输出结果
            print(f"Processed {file_path}, saved to {output_file_path}")
```

≡ 10051_output.txt	1	corner
≡ 10110_output.txt	2	clever
≡ 10253_output.txt	3	throttle
≡ 10370_output.txt	4	think
≡ 10408_output.txt	5	midget
≡ 10612_output.txt	6	soothsayer
≡ 10640_output.txt	7	shrivel
≡ 10776_output.txt	8	pay
≡ 10964_output.txt	9	ferry
≡ 11125_output.txt	10	box
≡ 11236_output.txt	11	go
	12	call
	13	after
	14	bury
	15	close
	16	melange

导入后

<input type="checkbox"/>	文件名称	File ID
<input type="checkbox"/>	1066.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/1066.txt
<input type="checkbox"/>	12375.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/12375.txt
<input type="checkbox"/>	2156.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/2156.txt
<input type="checkbox"/>	2860.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/2860.txt
<input type="checkbox"/>	3000.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/3000.txt
<input type="checkbox"/>	3335.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/3335.txt
<input type="checkbox"/>	5134.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/5134.txt
<input type="checkbox"/>	5376.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/5376.txt
<input type="checkbox"/>	5850.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/5850.txt
<input type="checkbox"/>	6486.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/6486.txt
<input type="checkbox"/>	6534.txt	cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/6534.txt

其中，每个文件的 File ID 为测试所需

验证算法代码嵌在 `page test4` 中，在 `onLoad()` 函数中通过去掉注释来启用，并通过控制台查看 `log` 来得知验证结果，共有三个函数用来模拟程序的各个梯度运行过程从而模拟程序的核心算法

`testAlgorithm()`      `testAlgorithm2()`      `testAlgorithm3()`

通过微信的 `Promise` 方法以及 `resolve()` 来实现函数之间的同步，确保在上一个算法得出结果后才会往下运行

`OnLoad()`函数实例

```
onLoad(options) {
  if (options.final) { ...
  }
  this.getData();

  this.testAlgorithm()
    .then(() => {
      return this.testAlgorithm2(); // 使用返回的 Promise 来等待 testAlgorithm2 完成
    })
    .then(() => {
      //console.log('test2已调用结束');
      // 确保 final2 被计算出来后调用 testAlgorithm3
      this.testAlgorithm3(this.data.final2);
    })
    .then(() => {
      //console.log('testAlgorithm3() 已调用');
    })
    .catch(err => {
      console.error('出错:', err);
    });
},
```

## TestAlgorithm 核心部分

### 1. 模拟复选框的数据填充

```
testAlgorithm() {  
  return new Promise((resolve, reject) => {  
    const db = wx.cloud.database();  
    const databaseArray = [];  
  
    const fetchWordData1 = (collection, size) => {  
      return db.collection(collection)  
        .aggregate()  
        .sample({ size: size })  
        .end()  
        .then(res => res.list.map(item => item.headword.trim()));  
    };  
  
    const fetchPromises = Array.from({ length: 20 }, (_, index) => {  
      const collectionName = `word${index}`;  
      return fetchWordData1(collectionName, 2).then(words => { ...  
    });  
  });  
});
```

### 2. 从微信云存储下载数据进行测试的计算逻辑

```
wx.cloud.downloadFile({  
  fileId: this.data.cloudpath,  
  timeout: '10000000',  
  success: res => {  
    if (res.statusCode === 200) {  
      const wordListContent = res.tempFilePath;  
      wx.getFileSystemManager().readFile({  
        filePath: wordListContent,  
        encoding: 'utf8',  
        success: res => {  
          const wordListContent = res.data;  
          const wordListArray = wordListContent.split('\n').map(word => word.trim());  
          //console.log('wordListArray',wordListArray);  
          let weightSum = 0;  
          let sum = 0;  
          wordListArray.forEach(word => {  
            databaseArray.forEach(database => {  
              database.words.forEach(dbWord => {  
                //if (this.isSubsequenceMatch(dbWord.word, word, 3)) { //若有匹配单词则权重累计增加  
                if(dbWord.word === word) {  
                  //console.log(`Match found between ${dbWord.word} and ${word}`);  
                  sum = sum+1;  
                  weightSum += Math.log1p(database.count - 1);  
                }  
              }  
            }  
          }  
        }  
      }  
    }  
  }  
});
```



## 五、模块展示说明（界面截图、测试过程、结果讨论）

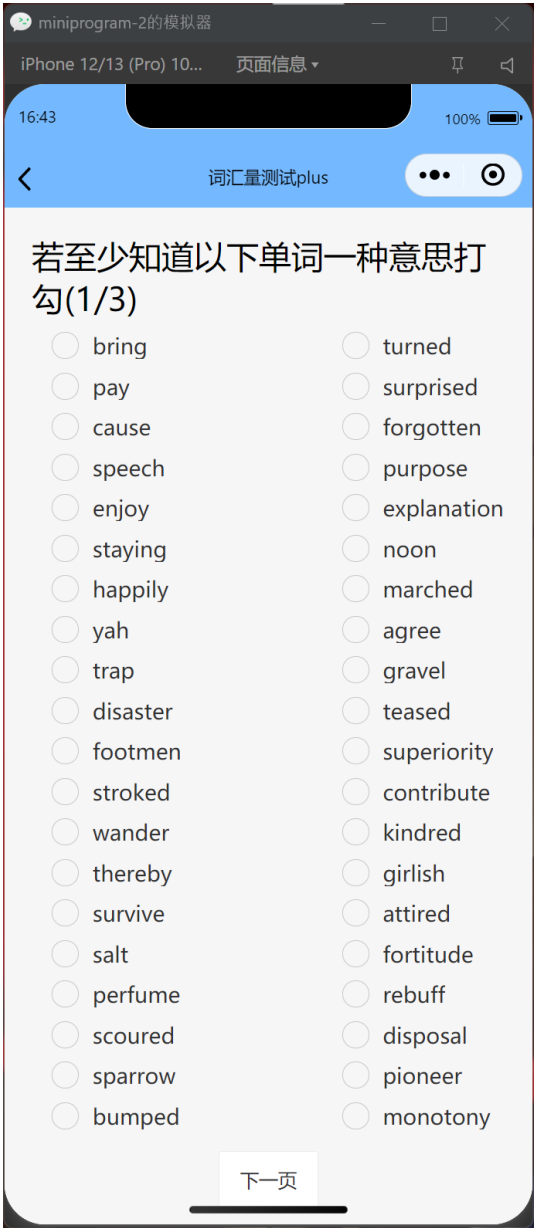
### 一、UI 页面：

Index page:

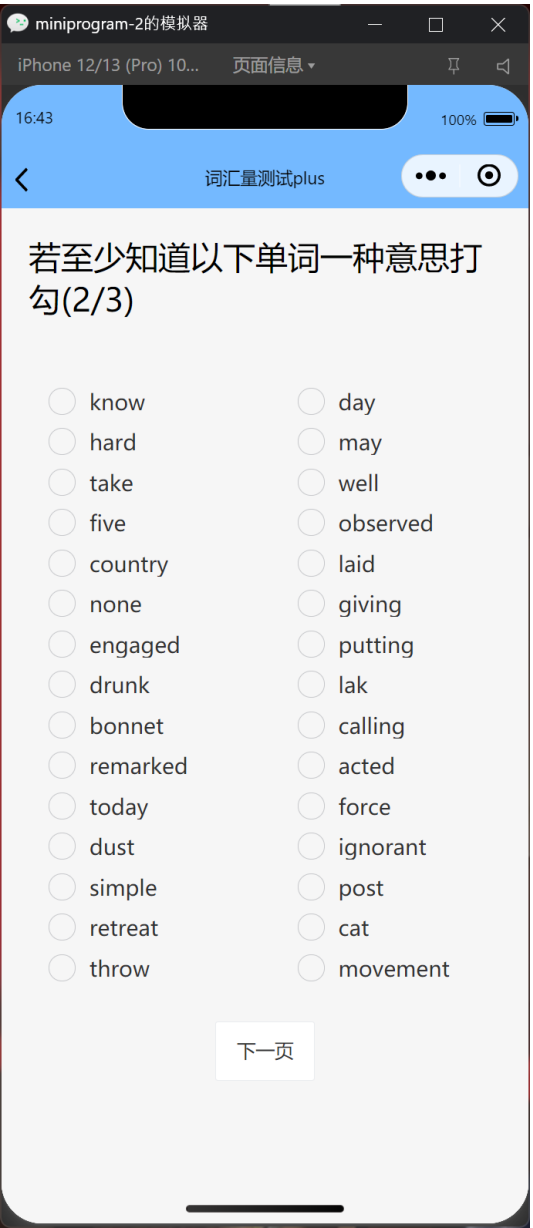


\*只有在输入测试者姓名后才会跳转页面

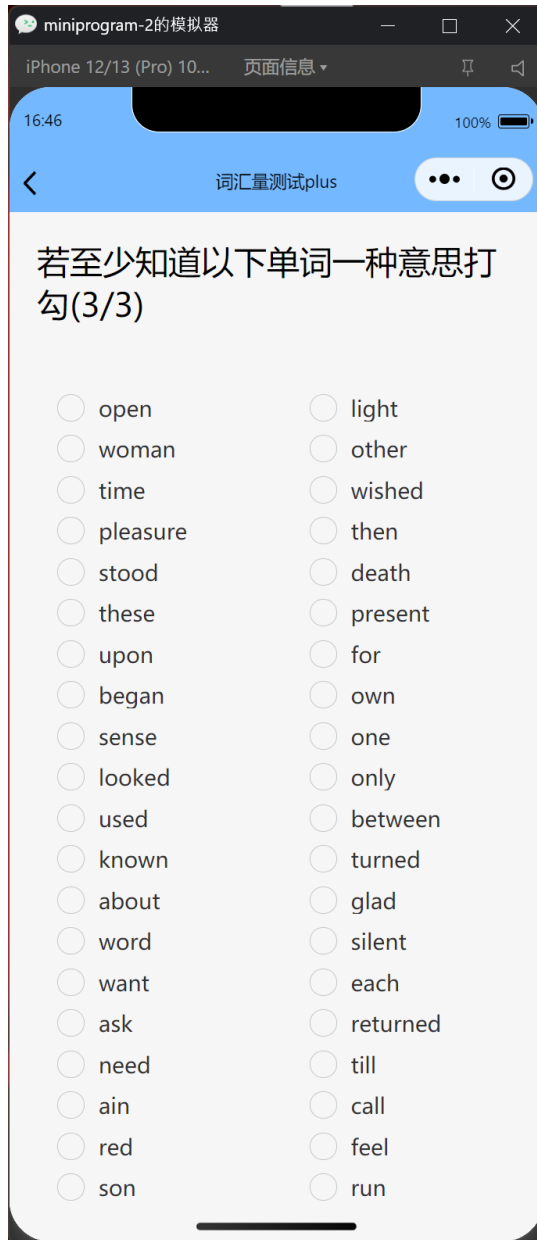
第一梯度测试 page



第二梯度测试 page



### 第三梯度测试 page



### Finish page



## 二、GUI 演示测试:

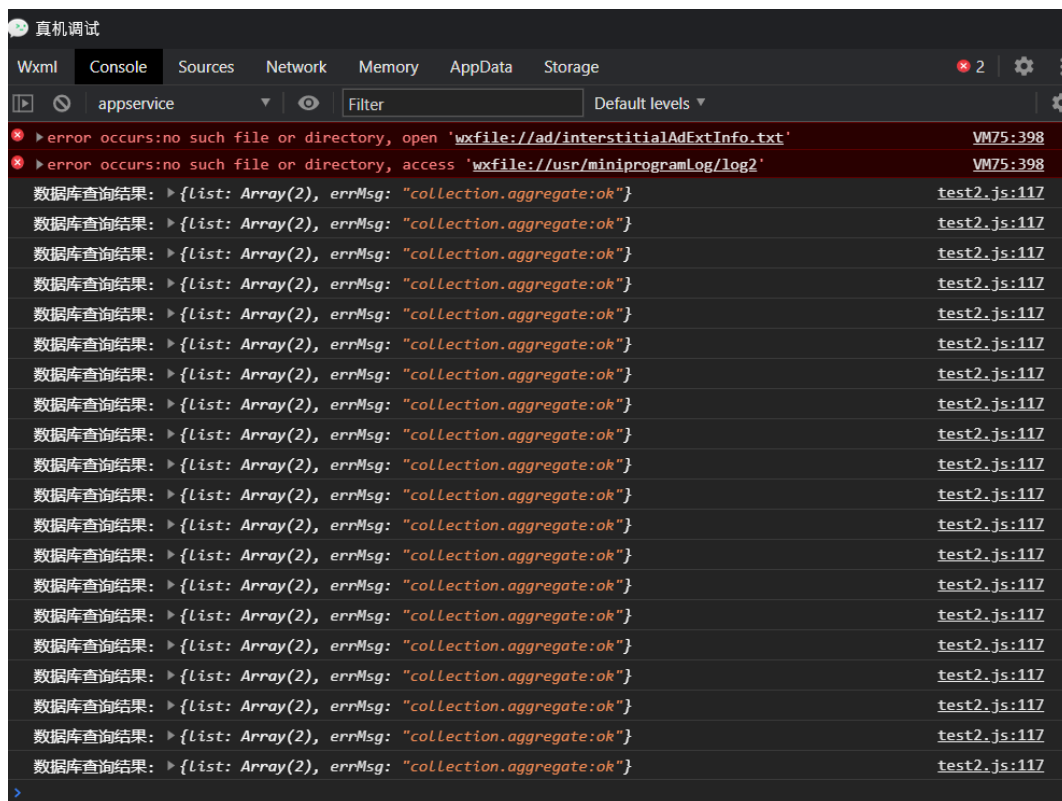
勾选复选框一步步测试即可，测试结束时可在后台的 `user` 集合中找到测试者的测试数据

启用真机调试，在电脑的调试段中查看日志

现在模拟正常测试者在手机上的测试效果

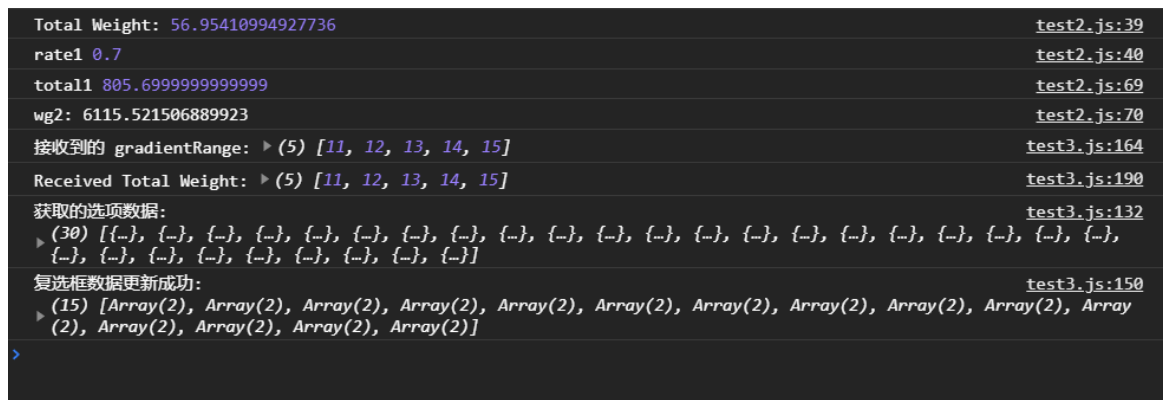


① 在 index 页面输入姓名并跳转下一页时，调试台显示日志信息



代表已成功从数据库选取单词数据并填入到复选框中

② 在测试页面一选完并跳转下一页时，调试台显示相关日志信息



其中 **Total Weight** 代表在这一梯度的权重值，用于确定下一梯度的 **gradient** 数组范围 **Total Weight: 56.95410994927736**

计算得出 **gradientRange: (5) [11, 12, 13, 14, 15]** 因此下一梯度从 word10-14 选取数据

**Rate1=0.7** 代表这一梯度的正确率，可以看到调试台有

```
total|1 805.6999999999999
```

**wg2: 6115.521506889923**

分别为这个梯度所得到的**总分**和下一梯度的**权重值**

③ 在测试页面二选完后进入最后梯度，此时调试台打印信息：

```
最后收到的total1为 805
最后收到的total2为 2140
最后收到的wg3为 7071.029767419511
Received Final: 14
```

前面不在赘述，**Received Final: 14** 代表进入最终梯度 14，对应 wg3 为 7071  
查表可知 wg3 没有错误

④ 在最后页面做完测试后，可以得出调试信息，同时 finish page 也显示结果：

中国联通 17:09 2.2 K/s 32%

< 词汇量测试plus ...

● 已连接 展开

<input checked="" type="checkbox"/> blunder	<input checked="" type="checkbox"/> meddle
<input checked="" type="checkbox"/> fascinating	<input checked="" type="checkbox"/> romance
<input checked="" type="checkbox"/> rejected	<input checked="" type="checkbox"/> policeman
<input checked="" type="checkbox"/> outright	<input checked="" type="checkbox"/> wrapping
<input checked="" type="checkbox"/> recognise	<input type="checkbox"/> ruffle
<input type="checkbox"/> perched	<input checked="" type="checkbox"/> beckoned
<input checked="" type="checkbox"/> feeding	<input checked="" type="checkbox"/> consciously
<input checked="" type="checkbox"/> pledge	<input checked="" type="checkbox"/> unreal
<input type="checkbox"/> muff	<input checked="" type="checkbox"/> hypocrisy
<input type="checkbox"/> plaid	<input checked="" type="checkbox"/> criticism
<input checked="" type="checkbox"/> tilted	<input checked="" type="checkbox"/> destruction
<input type="checkbox"/> foreman	<input checked="" type="checkbox"/> crouched
<input checked="" type="checkbox"/> encountered	<input checked="" type="checkbox"/> pounding
<input checked="" type="checkbox"/> rocked	<input checked="" type="checkbox"/> cooking
<input checked="" type="checkbox"/> shabby	<input checked="" type="checkbox"/> glazed
<input type="checkbox"/> merriment	<input type="checkbox"/> swoon
<input checked="" type="checkbox"/> spit	<input checked="" type="checkbox"/> welfare
<input checked="" type="checkbox"/> allowing	<input type="checkbox"/> gnawing
<input checked="" type="checkbox"/> drawl	<input checked="" type="checkbox"/> drifted
<input checked="" type="checkbox"/> leap	<input checked="" type="checkbox"/> breadth

下一页

中国联通 17:09 12 K/s 32%

< 词汇量测试plus ...

● 已连接 展开

你的词库词汇量约是

8036

再测一次

返回首页

Received Final: 14
rate3 0.8
Score: 8036
接收到的 score: 8036

代表在第三梯度正确率为 0.8  
 利用公式计算后可得出 **Score: 8036**  
 至此，GUI 演示完成

\*最后一梯度正确率为 **0.8**，证明是符合测试者的最佳梯度，侧面佐证梯度下降筛选的算法稳定性

### 三、验证算法的准确性

由于算法的特殊性，因此将后台批处理验证与验证准确性相结合

- ① 在 test4 page 的 onLoad()函数中启用验证算法，并将 cloudpath 替换为微信云存储中不同模拟数据的 File ID，文件为处理过的词汇数据，记录为纯单词文本，代表测试者所认识的单词，也代表在模拟抓取数据中勾选的单词

```

9      'next',会
10     'hint',会
11     'toss',会
12     'glide',会
13     'warranty',会
14     'clutch',会
15     'ceiling',会
16     'hamper',会
17     'hedge',会
18     'tinker',不会
19     'drought',不会
20     'dangle',不会
21     'outright',不会
22     'beware',不会
23     'wriggle',不会
24     'cradle',不会
25     'ledge',不会
26     'loot',不会
27     '测试结果',2568

```

```

File Edit View
him
sapling
like
suit
clay
ajar
ask
close
fish
hope
nest
embankment
slough
thrust
extol
respite
tan
tipple
feeble
forbid
warranty
ballast

```

处理代码：

```
def extract_words_from_files(directory):
    # 匹配所有以 stimulate_ 开头的 txt 文件
    file_paths = glob.glob(os.path.join(directory, 'stimulate_*.txt'))
    for file_path in file_paths:
        with open(file_path, 'r', encoding='utf-8') as file:
            lines = file.readlines()
            # 初始化空列表来存储符合条件的单词
            words_only = []
            # 遍历每一行内容，排除最后一行
            for line in lines[:-1]: # 使用切片去掉最后一行
                parts = line.split(',')
                if len(parts) > 1 and parts[1].strip() == '会': # 判断第二部分是否为“会”
                    word = parts[0].strip() # 获取单词部分并去除空格和换行符
                    clean_word = re.sub(r'^\w\s', '', word) # 使用正则表达式去除单词中的符号
                    words_only.append(clean_word) # 将处理后的单词部分添加到列表中
```

②将 cloudpath 替换为 cloud://cloud1-5gmwyfvq214ca144.636c-cloud1-5gmwyfvq214ca144-1327334961/7931\_output.txt，也就是估值为 7931 的词汇量后，验证出的 score 分别为

weightSum3 32	weightSum3 28
rate3 0.8	rate3 0.7
wg3 7071.029767419511	wg3 5971.4705131169585
score 7862.198243057191	score 6329.2325168824955

weightSum3 31	weightSum3 25
rate3 0.775	rate3 0.625
wg3 7071.029767419511	wg3 8325.394597984712
score 7674.325073290251	score 8075.3329258683225

取平均值可知 score=7485，相差 446  
关于更多梯度的验证见附表

## 四、关于测试结果的讨论

①关于 GUI 测试的结果

测试者 1：杨舒越

测试者 1 的百词斩测试结果为：8306

测试者 1 的 Vocabulary test 测试结果为：8340

测试者\测试次数	第一次测试	第二次测试	第三次测试	平均结果
测试者 1	8065	8036	8574	8225
测试者 2	1304	1578	1526	1469
测试者 3	6898	7138	7117	7051

② 关于后台验证算法准确性的结果：

标准值\测试次数	第一次	第二次	第三次	平均值	平均值和标准值差值
3355	2962	3745	3428	3378.33	<b>23.33</b>
5450	5110	5100	5475	5228.33	<b>-221.67</b>
5832	5840	5991	5479	5770.00	<b>-62.00</b>
6534	6448	6395	6661	6501.33	<b>-32.67</b>
6920	7178	6921	7028	7042.33	<b>122.33</b>
7931	7844	7736	8196	7925.33	<b>-5.67</b>
8586	8222	8406	8624	8417.33	<b>-168.67</b>
9169	8975	9550	8942	9155.67	<b>-13.33</b>
9960	9597	10339	9630	9855.33	<b>-104.67</b>
10964	11330	10740	10752	10940.67	<b>-23.33</b>
11236	11531	11251	11539	11440.33	<b>204.33</b>
11697	11933	11923	11461	11772.33	<b>75.33</b>
13216	12945	13499	13245	13229.67	<b>13.67</b>

## 五、程序的扩展实现

该程序实现了将用户的测试结果发送至后台数据库的实现  
具体实现思路如下：首先在 index 跳转页面时，用微信自带的 `setStorageSync` 方法将 `nameValue` 保存至本地缓存，代码实现如下

```
// 保存数据
saveName() {
  const { name } = this.data;
  if (name) {
    wx.setStorageSync('userName', name);

    return true;
  } else {
    wx.showToast({
      title: '请输入姓名',
      icon: 'none',
      duration: 2000
    });
    return false;
  }
},
```

```
// 跳转到 Test2 页面并保存数据
navigateToTest2() {
  if (this.saveName()) {
    wx.navigateTo({
      url: '/pages/test2/test2'
    });
  }
}
});
```

在最后一个测试页面，当跳转至 finish 时调用 `saveUserResult` 函数，将 `score` 保存至数据库中

`saveUserResult()`的代码实现如下

```

saveUserResult() {
  const username = wx.getStorageSync('userName'); // 从本地缓存获取用户名
  const finalResult = this.data.score; // 假设 final 存储在页面数据中
  if (username && finalResult) {
    this.addUserToDatabase(username, finalResult);
  } else {
    console.error('用户名或 final 值为空，无法保存到数据库');
  }
},

```

在这段函数中同时调用 addUserToDatabase 函数为用户添加一张表，addUserToDatabase()的实现方法如下

```

addUserToDatabase(username, Result) {
  const db = wx.cloud.database();
  // 使用云开发插入数据的方法
  db.collection('user').add({
    data: {
      username: username,
      result: Result,
      timestamp: db.serverDate() // 添加时间戳
    },
    success: res => {
      //console.log('用户数据存入数据库成功', res);
    },
    fail: err => {
      console.error('用户数据存入数据库失败', err);
    }
  });
},

```

经过一系列调用后即可实现将用户成绩和信息以及测试的时间保存至数据库

实例:

测试结果以及数据库接收信息(手机端和电脑调试端均可以接收)



```
+ 添加字段

"_id": "e2764d2d667fcf4e03cb161070b9f67a"
"_openid": "oqG3U4pCFU8nI6LkzIBwzzFEDsyw"
"result": 8036
"timestamp": "Sat Jun 29 2024 17:09:34 GMT+0800 (China Standard Time)"
"username": "ysy"
```



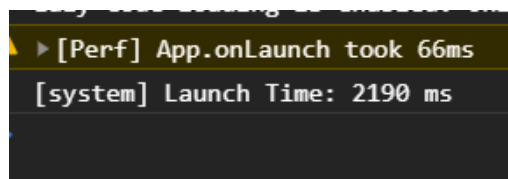


## 六、课程设计体会（团队体会、个人体会）

在本次软件系统课程设计中，我初步学会了微信小程序的开发，同时在开发过程中，我学会了以下几点：

- ① 程序的模块化设计：我学会了在程序中通过组织不同的函数来实现不同的逻辑，例如 `getdata()`, `navigateToNextPage()`, `calculateTotalWeight()`, 等函数同时还注意到同步的重要性，因此使用 `Promise-resolve` 方法来实现了函数之间的同步
- ② 数据处理：使用 `calculateWeight()` 来处理权重以及计算比例系数等，同时还通过 `Math` 方法构建对数模型来合理化权重计算方式
- ③ 用户交互：使用 `onChange` 函数处理复选框的变化，`onLoad` 函数处理页面被加载时的初始化工作，`getData` 方法用于从数据库获取数据，`setData` 方法用于更新页面数据，初步了解了响应式编程的特点
- ④ 异步编程：通过使用 `Promise` 和 `async/await` 方法，用来处理异步操作，如数据库查询、参数传递和文件读取，让我简单学会了管理复杂的异步逻辑
- ⑤ 错误处理：我在多个可能会出现异常的地方使用了 `try/catch` 结构来捕获和处理可能发生的错误，同时可以提高程序的健壮性。
- ⑥ 调试和日志记录：与错误处理相结合，在代码中广泛使用了 `console.log` 来记录关键信息，让我更进一步学会了调试程序，帮助我在开发过程中找到了许多问题，例如变量未被正确声明、函数表达错误、参数获取错误、同步问题等代码异常
- ⑦ 数据库操作：学会了如何使用微信云开发数据库进行数据的增删改查，包括聚合查询、文件下载、新建集合并往其中写入数据等
- ⑧ 网页样式的设计：通过微信的 `wxss`、`wxml`、等样式文件，对于网页布局以及元素分布排列有了更进一步的了解，初步学会使用 `@vant` 组件
- ⑨ 配置提取：将 `cloudpath` 等变量提取到 `data` 内，便可以实现管理外部资源，便于更改全局代码

不过程序也有一些缺点需要改进，例如性能优化：



可以看出程序加载时间略长，可以优化使之加载更快

还有代码重用问题：在 `testAlgorithm2()`, `testAlgorithm2()` 和 `testAlgorithm3()` 中，可以看出代码有重用的趋势，因此代码可以被进一步抽象和重用