1. 建立 Tokenizer 类

```python
import jieba
import pandas as pd
import re


class Tokenizer:
    def __init__(self, chars, coding='c', PAD=0):
        self.coding = coding
        self.chars = chars
        self.PAD = PAD
        d = {'[PAD]':PAD}
        code = 1
        if coding == 'c':
            for string in chars:
                for char in string:
                    if char not in d:
                        d[char] = code
                        code += 1
        elif coding == 'w':
            for string in chars:
                string = [word for word in jieba.lcut(string)]
                for word in string:
                    if word not in d:
                        d[word] = code
                        code += 1
        self.vocabulary = d

    def tokenize(self, sentence):
        if self.coding == 'c':
            list_of_chars = [char for char in sentence]
        elif self.coding == 'w':
            list_of_chars = [word for word in jieba.lcut(sentence)]
        return list_of_chars

    def encode(self, list_of_chars):
        return [self.vocabulary[i] for i in list_of_chars]

    def trim(self, tokens, seq_len):
        n = seq_len - len(tokens)
        if n < 0:  # 超出部分截断
            tokens = tokens[:seq_len]
        else:  # 不足部分补足
            for i in range(n):
                tokens.append(self.PAD)
        return tokens

    def decode(self, tokens):
        voc_reverse = {v:k for k,v in self.vocabulary.items()}
        return ''.join(voc_reverse[i] for i in tokens)

    def encode_all(self, seq_len):
        return [self.trim(self.encode(self.tokenize(i)), seq_len) for i in self.chars]
```

```python
def cut(str):  # 用正则表达式删去无意义字符
    c = re.findall(".+?(?=http|我在:|我在这里:)", str, re.S)
    if c == []:
        return str
    else:
        return c[0]


def main():
    D = pd.read_table(r'final_none_duplicate.txt',
                      encoding = 'utf-8', header = None, dtype = str)
    chars = D.iloc[:,1].dropna().apply(cut)  # 处理缺失值并删去结尾无效字符
    chars.drop_duplicates(keep = 'first', inplace = True)   # 删除重复项
    tokenizer_c = Tokenizer(chars, coding='c', PAD=0)
    tokenizer_w = Tokenizer(chars, coding='w', PAD=0)
    vectors_c = tokenizer_c.encode_all(64)
    print("按字分词：")
    print("编码：", vectors_c[44])
    print("解码：", tokenizer_c.decode(vectors_c[44]))
    print("分词：", tokenizer_c.tokenize(tokenizer_c.decode(vectors_c[44])))
    print("按词分词：")
    vectors_w = tokenizer_w.encode_all(64)
    print("编码：", vectors_w[0])
    print("解码：", tokenizer_w.decode(vectors_w[0]))
    print("分词：", tokenizer_w.tokenize(tokenizer_w.decode(vectors_w[0])))


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

句子长度 seq_len 的选择：句子长度应当选取在分布概率最大，即文档中句子长度频数最集中的区间，观察文档，本文设置 seq_len=64。

```
按字分词：
编码： [636, 637, 637, 48, 121, 638, 12, 10, 405, 62, 274, 41, 639, 48, 290, 235, 47, 48, 640, 99, 12, 116, 638,
47, 48, 640, 391, 12, 190, 641, 24, 636, 600, 642, 59, 48, 643, 166, 12, 13, 644, 645, 50, 24, 639, 637, 637, 48,
53, 25, 646, 195, 48, 327, 25, 647, 243, 22, 24, 648, 648, 12, 405, 649]
解码： 雨悄悄的下着，我就那么安静的走在你的背后，看着你的背影，周围是雨水落地的声音，心仿佛也是静悄悄的。一路上的第一句
话却是拜拜，就像
分词： ['雨', '悄', '悄', '的', '下', '着', '，', '我', '就', '那', '么', '安', '静', '的', '走', '在', '你', '的',
'背', '后', '，', '看', '着', '你', '的', '背', '影', '，', '周', '围', '是', '雨', '水', '落', '地', '的', '声',
'音', '，', '心', '仿', '佛', '也', '是', '静', '悄', '悄', '的', '。', '一', '路', '上', '的', '第', '一', '句',
'话', '却', '是', '拜', '拜', '，', '就', '像']
```

```
按词分词：
编码： [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
解码： 初中美人完全无脑啊我擦，心伤透 [PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD]
[PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD]
[PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD][PAD]
分词： ['初中', '美人', '完全', '无脑', '啊', '我', '擦', '，', '心', '伤透', ' ', '[', 'PAD', ']', '[', 'PAD', ']',
'[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD',
']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[',
'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']',
'[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD',
']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[',
'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']',
'[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD',
']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']', '[', 'PAD', ']']
```

2. 比较 tokenizer 与 one-hot

one-hot 编码当词表很大时，向量维度就会很高，使计算复杂度上升，且会失去语义信息。Tokenizer 编码的方式能有效降低向量维度，此外，可以 Tokenizer 在分词后根据词性词频编码，方便删去一些词频较低的词，提高自然语言处理任务的正确率。

3. BERT 与 Word2Vec

Word2Vec 虽然相比于 one-hot 解决了高维度和语义表示问题，但没能解决同名词不同义的问题。
BERT 使用双向 Transformer，利用单词的上下文信息来做特征提取，解决了同名不同义的问题。