Serverless 实战：如何结合 NLP 实现文本摘要和关键词提取？

原创 Anycodes Go Serverless 5月11日

本文已经被InfoQ收录：https://www.infoq.cn/article/d0JZjH0lpTTJqhj32VG1

前言

敏感词过滤是随着互联网社区发展一起发展起来的一种阻止网络犯罪和网络暴力的技术手段，通过对可能存在犯罪或网络暴力可能的关键词进行有针对性的筛查和屏蔽，很多时候我们能够防患于未然，把后果严重的犯罪行为扼杀于萌芽之中。

随着各种社交论坛等的日益火爆，敏感词过滤逐渐成了非常重要的也是值得重视的功能。那么在Serverless架构下，通过Python语言，敏感词过滤又有那些新的实现呢？我们能否是用最简单的方法，实现一个敏感词过滤的API呢？

了解敏感过滤的几种方法

Replace方法

如果说敏感词过滤，其实不如说是文本的替换，以Python为例，说到词汇替换，不得不想到 `replace` ，我们可以准备一个敏感词库，然后通过 `replace` 进行敏感词替换:

```
def worldFilter(keywords, text):
    for eve in keywords:
        text = text.replace(eve, "***")
    return text
keywords = ("关键词1", "关键词2", "关键词3")
content = "这是一个关键词替换的例子，这里涉及到了关键词1还有关键词2，最后还会有关键词3。"
print(worldFilter(keywords, content))
```

但是动动脑大家就会发现，这种做法在文本和敏感词库非常庞大的前提下，会有很严重的性能问题。例如我将代码进行修改，进行基本的性能测试:

```
import time

def worldFilter(keywords, text):
    for eve in keywords:
        text = text.replace(eve, "***")
```

```
    return text
keywords =[ "关键词" + str(i) for i in range(0,10000)]
content = "这是一个关键词替换的例子，这里涉及到了关键词1还有关键词2，最后还会有关键词3。" * 1(
startTime = time.time()
worldFilter(keywords, content)
print(time.time()-startTime)
```

此时的输出结果是： 0.12426114082336426 ，可以看到性能非常差。

正则表达方法

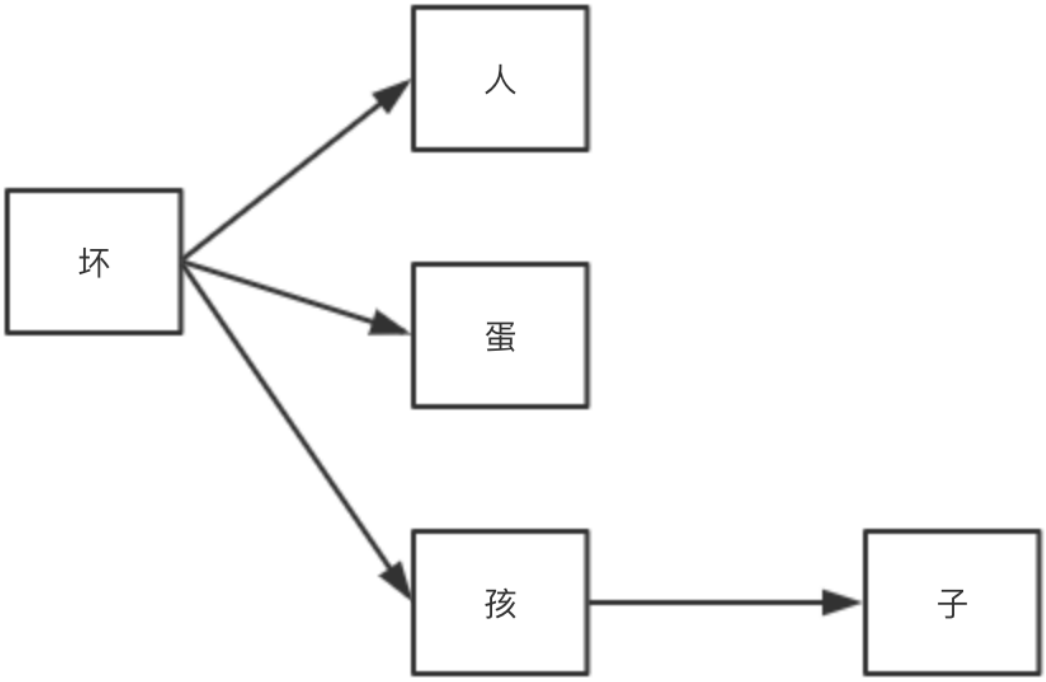
与其用 replace ，还不如通过正则表达 re.sub 来的更加快速。

```
import time
import re
def worldFilter(keywords, text):
    return re.sub("|".join(keywords), "***", text)
keywords =[ "关键词" + str(i) for i in range(0,10000)]
content = "这是一个关键词替换的例子，这里涉及到了关键词1还有关键词2，最后还会有关键词3。" * 1(
startTime = time.time()
worldFilter(keywords, content)
print(time.time()-startTime)
```

我们同样增加性能测试，按照上面的方法进行改造测试，输出结果是 0.24773502349853516 。通过这样的例子，我们可以发现，其性能磕韩剧并不大，但是实际上随着文本量增加，正则表达这种做法在性能层面会变高很多。

DFA过滤敏感词

这种方法相对来说效率会更高一些。例如，我们认为坏人，坏孩子，坏蛋是敏感词，则他们的树关系可以表达：



用DFA字典来表示：

```
{
    '坏': {
        '蛋': {
            '\x00': 0
        },
        '人': {
            '\x00': 0
        },
        '孩': {
            '子': {
                '\x00': 0
            }
        }
    }
}
```

使用这种树表示问题最大的好处就是可以降低检索次数，提高检索效率，基本代码实现：

```
import time

class DFAFilter(object):
    def __init__(self):
        self.keyword_chains = {}  # 关键词链表
        self.delimit = '\x00'  # 限定

    def parse(self, path):
        with open(path, encoding='utf-8') as f:
            for keyword in f:
                chars = str(keyword).strip().lower()  # 关键词英文变为小写
                if not chars:  # 如果关键词为空直接返回
```

```
                    return
                level = self.keyword_chains
                for i in range(len(chars)):
                    if chars[i] in level:
                        level = level[chars[i]]
                    else:
                        if not isinstance(level, dict):
                            break
                        for j in range(i, len(chars)):
                            level[chars[j]] = {}
                            last_level, last_char = level, chars[j]
                            level = level[chars[j]]
                        last_level[last_char] = {self.delimit: 0}
                        break
                if i == len(chars) - 1:
                    level[self.delimit] = 0

    def filter(self, message, repl="*"):
        message = message.lower()
        ret = []
        start = 0
        while start < len(message):
            level = self.keyword_chains
            step_ins = 0
            for char in message[start:]:
                if char in level:
                    step_ins += 1
                    if self.delimit not in level[char]:
                        level = level[char]
                    else:
                        ret.append(repl * step_ins)
                        start += step_ins - 1
                        break
                else:
                    ret.append(message[start])
                    break
            else:
                ret.append(message[start])
            start += 1

        return ''.join(ret)



gfw = DFAFilter()
gfw.parse( "./sensitive_words")
content = "这是一个关键词替换的例子，这里涉及到了关键词1还有关键词2，最后还会有关键词3。" * 1(
startTime = time.time()
result = gfw.filter(content)
print(time.time()-startTime)
```

这里我们的字典库是：

```
with open("./sensitive_words", 'w') as f:
    f.write("\n".join( [ "关键词" + str(i) for i in range(0,10000)]))
```

执行结果：

```
0.06450581550598145
```

可以看到性能进一步提升。

AC自动机过滤敏感词算法

接下来，我们来看一下 AC自动机过滤敏感词算法：

AC自动机：一个常见的例子就是给出n个单词，再给出一段包含m个字符的文章，让你找出有多少个单词在文章里出现过。

简单地讲，AC自动机就是字典树+kmp算法+失配指针

代码实现：

```
import time
class Node(object):
    def __init__(self):
        self.next = {}
        self.fail = None
        self.isWord = False
        self.word = ""


class AcAutomation(object):

    def __init__(self):
        self.root = Node()

    # 查找敏感词函数
    def search(self, content):
        p = self.root
        result = []
        currentposition = 0

        while currentposition < len(content):
            word = content[currentposition]
            while word in p.next == False and p != self.root:
                p = p.fail

            if word in p.next:
                p = p.next[word]
            else:
                p = self.root
```

```
            if p.isWord:
                result.append(p.word)
                p = self.root
            currentposition += 1
        return result

    # 加载敏感词库函数
    def parse(self, path):
        with open(path, encoding='utf-8') as f:
            for keyword in f:
                temp_root = self.root
                for char in str(keyword).strip():
                    if char not in temp_root.next:
                        temp_root.next[char] = Node()
                    temp_root = temp_root.next[char]
                temp_root.isWord = True
                temp_root.word = str(keyword).strip()

    # 敏感词替换函数
    def wordsFilter(self, text):
        """
        :param ah: AC自动机
        :param text: 文本
        :return: 过滤敏感词之后的文本
        """
        result = list(set(self.search(text)))
        for x in result:
            m = text.replace(x, '*' * len(x))
            text = m
        return text


acAutomation = AcAutomation()
acAutomation.parse('./sensitive_words')
startTime = time.time()
print(acAutomation.wordsFilter("这是一个关键词替换的例子，这里涉及到了关键词1还有关键词2，最
print(time.time()-startTime)
```

词库同样是：

```
with open("./sensitive_words", 'w') as f:
    f.write("\n".join( [ "关键词" + str(i) for i in range(0,10000)]))
```

使用上面的方法，测试结果为 0.017391204833984375 。

敏感词过滤方法小结

可以看到这个所有算法中，在上述的基本算法中DFA过滤敏感词性能最高，但是实际上，对于后两者算法，并没有谁一定更好，可能某些时候，AC自动机过滤敏感词算法会得到更高的性能，

所以在生产生活中，推荐时候用两者，可以根据自己的具体业务需要来做。

实现敏感词过滤API

将代码部署到Serverless架构上，可以选择API网关与函数计算进行结合，以AC自动机过滤敏感词算法为例：我们只需要增加是几行代码就好，完整代码如下：

```
# -*- coding:utf-8 -*-

import json, uuid


class Node(object):
    def __init__(self):
        self.next = {}
        self.fail = None
        self.isWord = False
        self.word = ""


class AcAutomation(object):

    def __init__(self):
        self.root = Node()

    # 查找敏感词函数
    def search(self, content):
        p = self.root
        result = []
        currentposition = 0

        while currentposition < len(content):
            word = content[currentposition]
            while word in p.next == False and p != self.root:
                p = p.fail

            if word in p.next:
                p = p.next[word]
            else:
                p = self.root

            if p.isWord:
                result.append(p.word)
                p = self.root
            currentposition += 1
        return result

    # 加载敏感词库函数
    def parse(self, path):
        with open(path, encoding='utf-8') as f:
            for keyword in f:
                temp_root = self.root
                for char in str(keyword).strip():
```

```
                    if char not in temp_root.next:
                        temp_root.next[char] = Node()
                    temp_root = temp_root.next[char]
                temp_root.isWord = True
                temp_root.word = str(keyword).strip()

    # 敏感词替换函数
    def wordsFilter(self, text):
        """
        :param ah: AC自动机
        :param text: 文本
        :return: 过滤敏感词之后的文本
        """
        result = list(set(self.search(text)))
        for x in result:
            m = text.replace(x, '*' * len(x))
            text = m
        return text


def response(msg, error=False):
    return_data = {
        "uuid": str(uuid.uuid1()),
        "error": error,
        "message": msg
    }
    print(return_data)
    return return_data


acAutomation = AcAutomation()
path = './sensitive_words'
acAutomation.parse(path)


def main_handler(event, context):
    try:
        sourceContent = json.loads(event["body"])["content"]
        return response({
            "sourceContent": sourceContent,
            "filtedContent": acAutomation.wordsFilter(sourceContent)
        })
    except Exception as e:
        return response(str(e), True)
```

最后，为了方便本地测试，我们可以增加：

```
def test():
    event = {
        "requestContext": {
            "serviceId": "service-f94sy04v",
            "path": "/test/{path}",
            "httpMethod": "POST",
            "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
            "identity": {
                "secretId": "abdcdxxxxxxxsdfs"
```

```
            },
            "sourceIp": "14.17.22.34",
            "stage": "release"
        },
        "headers": {
            "Accept-Language": "en-US,en,cn",
            "Accept": "text/html,application/xml,application/json",
            "Host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqloud.com",
            "User-Agent": "User Agent String"
        },
        "body": "{\"content\":\"这是一个测试的文本，我也就呵呵了\"}",
        "pathParameters": {
            "path": "value"
        },
        "queryStringParameters": {
            "foo": "bar"
        },
        "headerParameters": {
            "Refer": "10.0.2.14"
        },
        "stageVariables": {
            "stage": "release"
        },
        "path": "/test/value",
        "queryString": {
            "foo": "bar",
            "bob": "alice"
        },
        "httpMethod": "POST"
    }
    print(main_handler(event, None))


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

完成之后，我们就可以测试运行一下，例如我的字典是：

```
呵呵
测试
```

执行之后结果：

```
{'uuid': '9961ae2a-5cfc-11ea-a7c2-acde48001122', 'error': False, 'message': {'sourceCor
```

接下来，我们将代码部署到云端，新建 serverless.yaml ：

```
sensitive_word_filtering:
  component: "@serverless/tencent-scf"
  inputs:
    name: sensitive_word_filtering
```

```
    codeUri: ./
    exclude:
      - .gitignore
      - .git/**
      - .serverless
      - .env
    handler: index.main_handler
    runtime: Python3.6
    region: ap-beijing
    description: 敏感词过滤
    memorySize: 64
    timeout: 2
    events:
      - apigw:
          name: serverless
          parameters:
            environment: release
            endpoints:
              - path: /sensitive_word_filtering
                description: 敏感词过滤
                method: POST
                enableCORS: true
                param:
                  - name: content
                    position: BODY
                    required: 'FALSE'
                    type: string
                    desc: 待过滤的句子
```

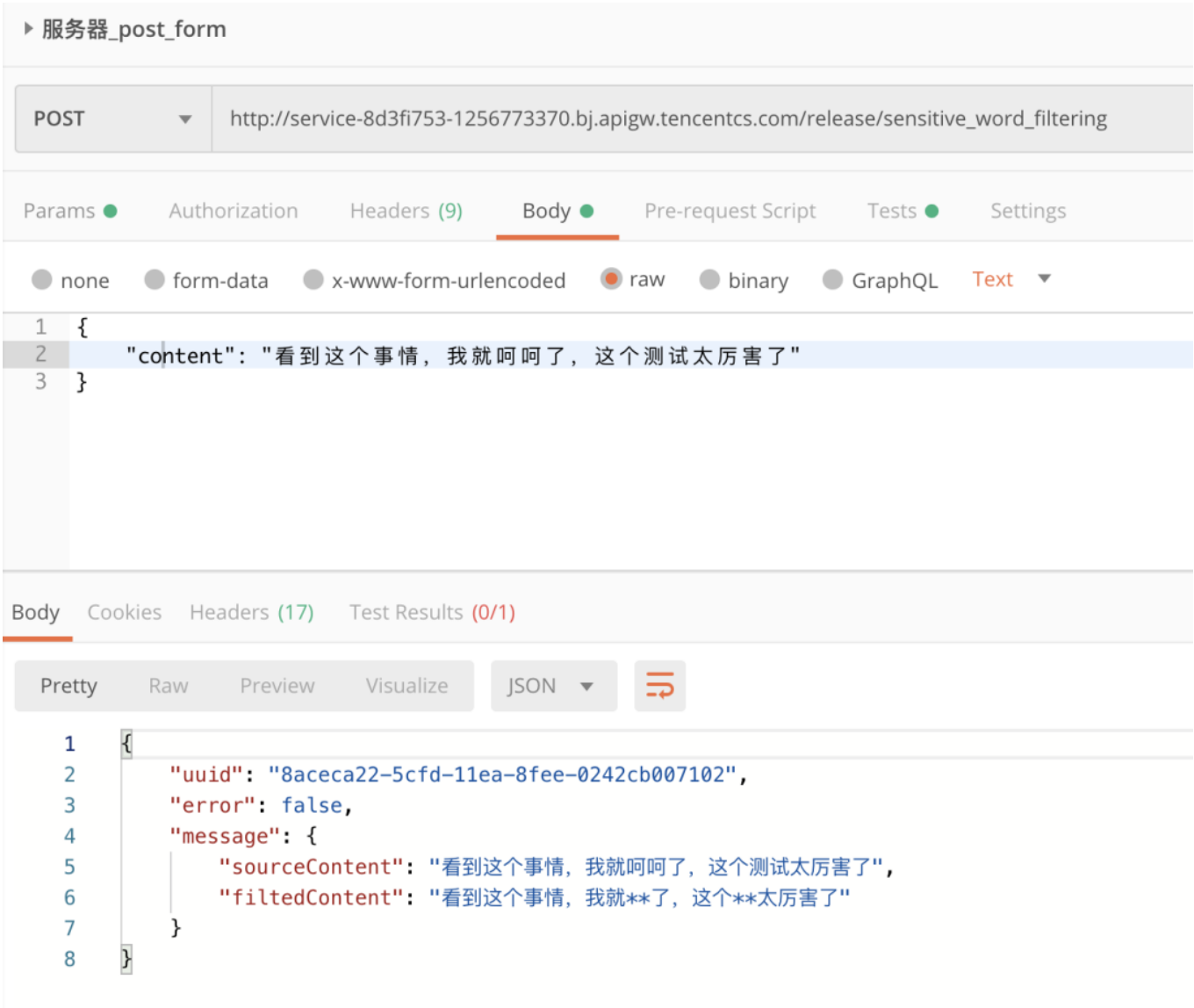
然后通过 `sls --debug` 进行部署，部署结果：

```
Login successful for TencentCloud.
  DEBUG - Compressing function sensitive_word_filtering file to /Users/dfounderliu/Documents/code/MyAPICenter/sen
  DEBUG - Compressed function sensitive_word_filtering file successful
  DEBUG - Uploading service package to cos[sls-cloudfunction-ap-beijing-code]. sls-cloudfunction-default-sensitiv
  DEBUG - Uploaded package successful /Users/dfounderliu/Documents/code/MyAPICenter/sensitiveWordFiltering/.serve
  DEBUG - Creating function sensitive_word_filtering
  sensitive_word_filtering [                                        ] 100% | ETA: 0s | Speed: 9.54k/s
  DEBUG - Created function sensitive_word_filtering successful
  DEBUG - Setting tags for function sensitive_word_filtering
  DEBUG - Creating trigger for function sensitive_word_filtering
  DEBUG - Starting API-Gateway deployment with name sensitive_word_filtering.serverless in the ap-beijing region
  DEBUG - Updating service with serviceId service-8d3fi753.
  DEBUG - API with id api-9ncafpwg created.
  DEBUG - Deploying service with id service-8d3fi753.
  DEBUG - Deployment successful for the api named sensitive_word_filtering.serverless in the ap-beijing region.
  DEBUG - Deployed function sensitive_word_filtering successful

  sensitive_word_filtering:
    Name:        sensitive_word_filtering
    Runtime:     Python3.6
    Handler:     index.main_handler
    MemorySize:  64
    Timeout:     2
    Region:      ap-beijing
    Namespace:   default
    Description: 敏感词过滤
    APIGateway:
      - serverless - http://service-8d3fi753-1256773370.bj.apigw.tencentcs.com/release

  32s › sensitive_word_filtering › done
```

最后，通过PostMan进行测试：



总结

敏感词过滤是目前非常常见的需求/技术，通过敏感词过滤，我们可以在一定程度上降低恶意言语或者违规言论的出现，在上述实践过程，有以下两点内容：

- 对于敏感词库额获得问题：Github上有很多，可以自行搜索下载，因为敏感词词库里面有很多敏感词，所以我也不能直接放在这个上面供大家使用，所以还需要大家自行在Github上搜索使用；
- 这个API使用场景的问题：完全可以放在我们的社区跟帖系统/留言评论系统/博客发布系统中，防止出现敏感词汇，可以降低不必要的麻烦出现。

阅读原文