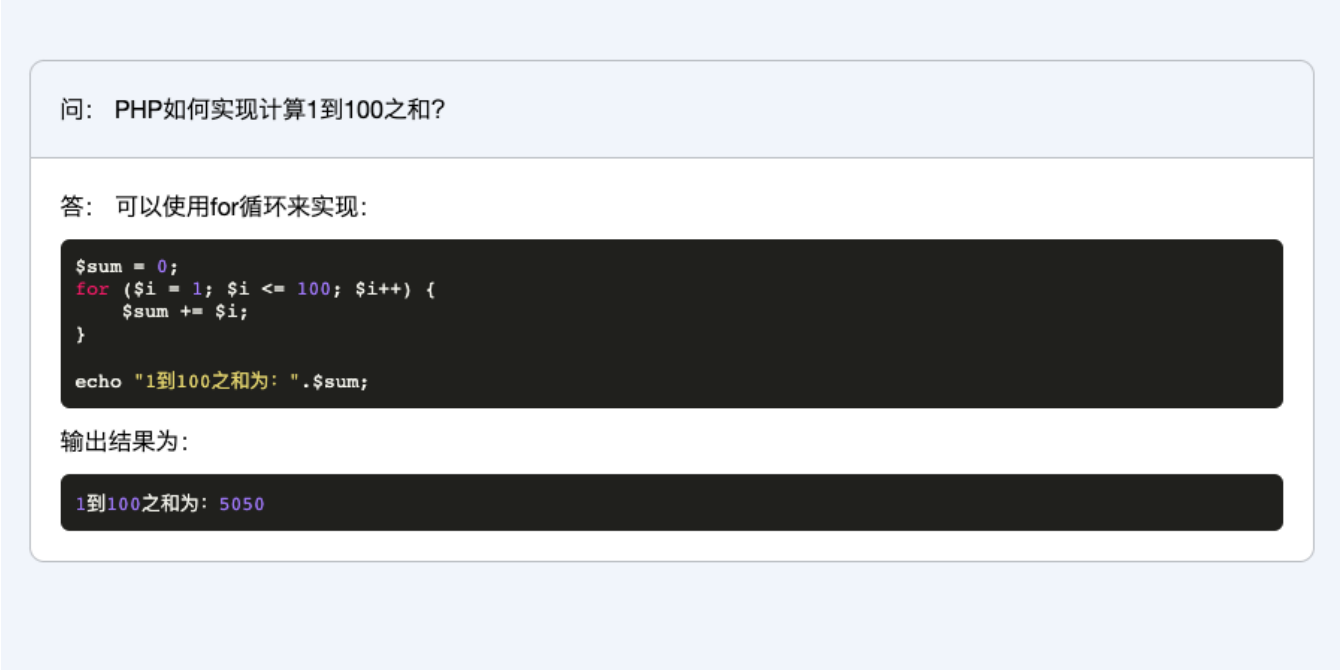
# 开源了一个不使用任何后端框架纯 php 实现流式调用 OpenAI gpt 接口的项目

原创 我是哥飞 哥飞 2023-03-24 10:19:04 广东




## php-openai-gpt-stream-chat-api-webui

由 @qiayue 开源的 纯 PHP 实现 GPT 流式调用和前端实时打印 webui 。

## 目录结构

```
/
├─ /class
│  ├─ Class.ChatGPT.php
│  ├─ Class.DFA.php
│  ├─ Class.StreamHandler.php
├─ /static
│  ├─ css
│  │  ├─ chat.css
│  │  ├─ monokai-sublime.css
│  ├─ js
│  │  ├─ chat.js
│  │  ├─ highlight.min.js
│  │  ├─ marked.min.js
├─ /chat.php
├─ /index.html
├─ /README.md
├─ /sensitive_words.txt
```

| 目录/文件 | 说明 |
| --- | --- |
| / | 程序根目录 |
| /class | php类文件目录 |
| /class/Class.ChatGPT.php | ChatGPT 类，用于处理前端请求，并向 OpenAI 接口提交请求 |
| /class/Class.DFA.php | DFA 类，用于敏感词校验和替换 |
| /class/Class.StreamHandler.php | StreamHandler 类，用于实时处理 OpenAI 流式返回的数据 |
| /static | 存放所有前端页面所需的静态文件 |
| /static/css | 存放前端页面所有的 css 文件 |
| /static/css/chat.css | 前端页面聊天样式文件 |
| /static/css/monokai-sublime.css | highlight 代码高亮插件的主题样式文件 |
| /static/js | 存放前端页面所有的 js 文件 |
| /static/js/chat.js | 前端聊天交互 js 代码 |
| /static/js/highlight.min.js | 代码高亮 js 库 |
| /static/js/marked.min.js | markdown 解析 js 库 |
| /chat.php | 前端聊天请求的后端入口文件，在这里引入 php 类文件 |
| /index.html | 前端页面 html 代码 |
| /README.md | 仓库描述文件 |
| /sensitive_words.txt | 敏感词文件，一行一个敏感词，需要你自己收集敏感词，也可以加我微信（同 GitHub id）找我要 |

## 使用方法

本项目代码，没有使用任何框架，也没有引入任何第三方后端库，前端引入了代码高亮库 highlight 和 markdown 解析库 marked 都已经下载项目内了，所以拿到代码不用任何安装即可直接使用。

唯二要做的就是把你自己的 api key 填进去。

获取源码后，修改

`chat.php`

，填写 OpenAI 的 api key 进去，具体请见：

```
$chat = new ChatGPT([
    'api_key' => '此处需要填入 openai 的 api key ',
]);
```

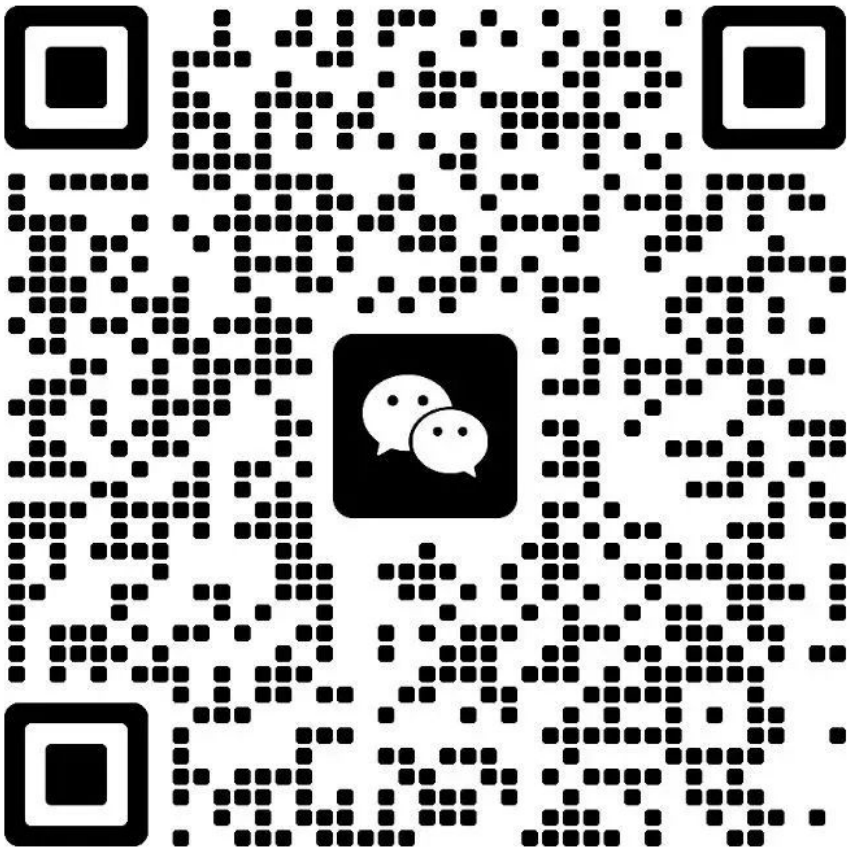
如果开启敏感词检测功能，需要把敏感词一行一个放入

`sensitive_words.txt`

文件中。

开了一个微信群，欢迎入群交流：

群聊：代码释疑群



该二维码7天内(3月31日前)有效，重新进入将更新

原理说明

## 流式接收 OpenAI 的返回数据

后端 Class.ChatGPT.php 中用 curl 向 OpenAI 发起请求，使用 curl 的

`CURLOPT_WRITEFUNCTION`

设置回调函数，同时请求参数里

`'stream' => true`

告诉 OpenAI 开启流式传输。

我们通过

`curl_setopt($ch, CURLOPT_WRITEFUNCTION, [$this->streamHandler, 'callback']);`

设置使用 StreamHandler 类的实例化对象

`$this->streamHandler`

的

`callback`

方法来处理 OpenAI 返回的数据。

OpenAI 会在模型每次输出时返回

`data: {"id":"","object":"","created":1679616251,"model":"","choices":[{"delta":{"content":""},"index":0,"fir`

格式字符串，其中我们需要的回答就在

`choices[0]['delta']['content']`

里，当然我们也要做好异常判断，不能直接这样获取数据。

另外，实际因为网络传输问题，每次

`callback`

函数收到的数据并不一定只有一条

`data: {"key":"value"}`

格式的数据，有可能只有半条，也有可能有多条，还有可能有N条半。

所以我们在

`StreamHandler`

类中增加了

`data_buffer`

属性来存储无法解析的半条数据。

这里根据 OpenAI 的返回数据格式，做了一些特殊处理，具体代码如下：

```
public function callback($ch, $data) {
        $this->counter += 1;
        file_put_contents('./log/data.'.$this->qmd5.'.log', $this->counter.'=='.$data.PHP_EOL.'--------------------'.

        $result = json_decode($data, TRUE);
        if(is_array($result)){
         $this->end('openai 请求错误: '.json_encode($result));
         return strlen($data);
        }

        /*
            此处步骤仅针对 openai 接口而言
            每次触发回调函数时，里边会有多条data数据，需要分割
            如某次收到 $data 如下所示：
```

```
            data: {"id":"chatcmpl-6wimHHBt4hKFHEpFnNT2ryUeuRRJC","object":"chat.completion.chunk","created":167945316

            最后两条一般是这样的：
            data: {"id":"chatcmpl-6wimHHBt4hKFHEpFnNT2ryUeuRRJC","object":"chat.completion.chunk","created":167945316

            根据以上 openai 的数据格式，分割步骤如下：
        */

        // 0、把上次缓冲区内数据拼接上本次的data
        $buffer = $this->data_buffer.$data;

        // 1、把所有的 'data: {' 替换为 '{' ，'data: [' 换成 '['
        $buffer = str_replace('data: {', '{', $buffer);
        $buffer = str_replace('data: [', '[', $buffer);

        // 2、把所有的 '}\n\n{' 替换维 '}[br]{' ，  '}\n\n[' 替换为 '}[br]['
        $buffer = str_replace('}'.PHP_EOL.PHP_EOL.'{', '}[br]{', $buffer);
        $buffer = str_replace('}'.PHP_EOL.PHP_EOL.'[', '}[br][', $buffer);

        // 3、用 '[br]' 分割成多行数组
        $lines = explode('[br]', $buffer);

        // 4、循环处理每一行，对于最后一行需要判断是否是完整的json
        $line_c = count($lines);
        foreach($lines as $li=>$line){
            if(trim($line) == '[DONE]'){
                //数据传输结束
                $this->data_buffer = '';
                $this->counter = 0;
                $this->sensitive_check();
                $this->end();
                break;
            }
            $line_data = json_decode(trim($line), TRUE);
            if( !is_array($line_data) || !isset($line_data['choices']) || !isset($line_data['choices'][0]) ){
                if($li == ($line_c - 1)){
                    //如果是最后一行
                    $this->data_buffer = $line;
                    break;
                }
                //如果是中间行无法json解析，则写入错误日志中
                file_put_contents('./log/error.'.$this->qmd5.'.log', json_encode(['i'=>$this->counter, 'line'=>$line,
                continue;
            }

            if( isset($line_data['choices'][0]['delta']) && isset($line_data['choices'][0]['delta']['content']) ){
             $this->sensitive_check($line_data['choices'][0]['delta']['content']);
            }
        }

        return strlen($data);
    }
```

## 敏感词检测

我们使用了 DFA 算法来实现敏感词检测，按照 ChatGPT 的解释，

`"DFA"是指"确定性有限自动机"（Deterministic Finite Automaton）`

，

`DfaFilter（确定有限自动机过滤器）通常是指一种用于文本处理和匹配的算法`

。

Class.DFA.php 类代码是 GPT4 写的，具体实现代码见源码。

这里介绍一下使用方法，创建一个 DFA 实例需要传入敏感词文件路径：

```
$dfa = new DFA([
    'words_file' => './sensitive_words.txt',
]);
```

之后就可以用

`$dfa->containsSensitiveWords($inputText)`

来判断

`$inputText`

是否包含敏感词，返回值是

`TRUE`

或

`FALSE`

的布尔值，也可以用

`$outputText = $dfa->replaceWords($inputText)`

来进行敏感词替换，所有在

`sensitive_words.txt`

中指定的敏感词都会被替换为三个

`*`

号。

如果不想开启敏感词检测，把

`chat.php`

中的以下三句注释掉即可：

```
$dfa = new DFA([
    'words_file' => './sensitive_words.txt',
]);
$chat->set_dfa($dfa);
```

如果没有开启敏感词检测，那么每次 OpenAI 的返回都会实时返回给前端。

如果开启了敏感词检测，会查找 OpenAI 返回中的换行符和停顿符号

`[', ', '。', '; ', '? ', '! ', '……']`

等来进行分句，每一句都使用

`$outputText = $dfa->replaceWords($inputText)`

来替换敏感词，之后整句返回给前端。

开启敏感词后，加载敏感词文件需要时间，每次检测时也是逐句检测，而不是逐词检测，也会导致返回变慢。

所以如果是自用，可以不开启敏感词检测，如果是部署出去给其他人用，为了保护你的域名安全和你的安全，最好开启敏感词检测。

## 流式返回给前端

直接看

`chat.php`

的注释会更清楚：

```
/*
以下几行注释由 GPT4 生成
*/

// 这行代码用于关闭输出缓冲。关闭后，脚本的输出将立即发送到浏览器，而不是等待缓冲区填满或脚本执行完毕。
ini_set('output_buffering', 'off');

// 这行代码禁用了 zlib 压缩。通常情况下，启用 zlib 压缩可以减小发送到浏览器的数据量，但对于服务器发送事件来说，实时性更重要，因此需要禁
ini_set('zlib.output_compression', false);

// 这行代码使用循环来清空所有当前激活的输出缓冲区。ob_end_flush() 函数会刷新并关闭最内层的输出缓冲区，@ 符号用于抑制可能出现的错误或警
while (@ob_end_flush()) {}

// 这行代码设置 HTTP 响应的 Content-Type 为 text/event-stream，这是服务器发送事件（SSE）的 MIME 类型。
header('Content-Type: text/event-stream');

// 这行代码设置 HTTP 响应的 Cache-Control 为 no-cache，告诉浏览器不要缓存此响应。
header('Cache-Control: no-cache');

// 这行代码设置 HTTP 响应的 Connection 为 keep-alive，保持长连接，以便服务器可以持续发送事件到客户端。
header('Connection: keep-alive');

// 这行代码设置 HTTP 响应的自定义头部 X-Accel-Buffering 为 no，用于禁用某些代理或 Web 服务器（如 Nginx）的缓冲。
// 这有助于确保服务器发送事件在传输过程中不会受到缓冲影响。
header('X-Accel-Buffering: no');
```

之后我们每次想给前端返回数据，用以下代码即可：

```
echo 'data: '.json_encode(['time'=>date('Y-m-d H:i:s'), 'content'=>'答:  ']).PHP_EOL.PHP_EOL;
flush();
```

这里我们定义了我们自己使用的一个数据格式，里边只放了 time 和 content，不用解释都懂，time 是时间， content 就是我们要返回给前端的内容。

注意，回答全部传输完毕后，我们需要关闭连接，可以用以下代码：

```
echo 'retry: 86400000'.PHP_EOL; // 告诉前端如果发生错误，隔多久之后才轮询一次
echo 'event: close'.PHP_EOL; // 告诉前端，结束了，该说再见了
echo 'data: Connection closed'.PHP_EOL.PHP_EOL; // 告诉前端，连接已关闭
flush();
```

## EventSource

前端 js 通过

```
const eventSource = new EventSource(url);
```

开启一个 EventSource 请求。

之后服务器按照

```
data: {"kev1":"value1","kev2":"value2"}
```

格式向前端发送数据，前端就可以在 EventSource 的 message 回调事件中的

```
event.data
```

里获取

```
{"kev1":"value1","kev2":"value2"}
```

字符串形式 json 数据，再通过

```
JSON.parse(event.data)
```

就可以得到 js 对象。

具体代码在 getAnswer 函数中，如下所示：

```
function getAnswer(inputValue){
    inputValue = inputValue.replace('+', '{[$add$]}');
    const url = "./chat.php?q="+inputValue;
    const eventSource = new EventSource(url);

    eventSource.addEventListener("open", (event) => {
        console.log("连接已建立", JSON.stringify(event));
    });

    eventSource.addEventListener("message", (event) => {
        //console.log("接收数据: ", event);
        try {
            var result = JSON.parse(event.data);
            if(result.time && result.content ){
                answerWords.push(result.content);
                contentIdx += 1;
            }
        } catch (error) {
            console.log(error);
        }
    });

    eventSource.addEventListener("error", (event) => {
        console.error("发生错误: ", JSON.stringify(event));
    });

    eventSource.addEventListener("close", (event) => {
        console.log("连接已关闭", JSON.stringify(event.data));
        eventSource.close();
        contentEnd = true;
        console.log((new Date().getTime()), 'answer end');
    });
}
```

## 打字机效果

对于后端返回的所有回复内容，我们需要用打字机形式打印出来。

最初的方案是

```
function typingWords(){
    if(contentEnd && contentIdx==typingIdx){
        clearInterval(typingTimer);
        answerContent = '';
        answerWords = [];
        answers = [];
        qaIdx += 1;
        typingIdx = 0;
        contentIdx = 0;
        contentEnd = false;
        lastWord = '';
        lastLastWord = '';
        input.disabled = false;
        sendButton.disabled = false;
        console.log((new Date().getTime()), 'typing end');
        return;
    }
    if(contentIdx<=typingIdx){
        return;
    }
    if(typing){
        return;
    }
    typing = true;

    if(!answers[qaIdx]){
        answers[qaIdx] = document.getElementById('answer-'+qaIdx);
    }

    const content = answerWords[typingIdx];
    if(content.indexOf('`') != -1){
        if(content.indexOf('```') != -1){
            codeStart = !codeStart;
        }else if(content.indexOf('``') != -1 && (lastWord + content).indexOf('```') != -1){
            codeStart = !codeStart;
        }else if(content.indexOf('`') != -1 && (lastLastWord + lastWord + content).indexOf('```') != -1){
            codeStart = !codeStart;
        }
    }

    lastLastWord = lastWord;
    lastWord = content;

    answerContent += content;
    answers[qaIdx].innerHTML = marked.parse(answerContent+(codeStart?'\n\n```':''));

    typingIdx += 1;
    typing = false;
}
```

## 其它

更多其它细节请看代码，如果对代码有疑问的，请加我微信（同 GitHub id）

## License

BSD 2-Clause

点击阅读原文，查看开源代码

阅读原文