文字识别

一、文字识别优势

通用文字识别（Universal Character Recognition）基于深度神经网络模型的端到端文字识别系统，将图片（来源如扫描仪或数码相机）中的印刷或手写文字转化为计算机可编辑的文字

- 高精度：笔声OCR基于深度学习技术，基于大规模数据集训练，在公开数据集上的识别效果已经超过了其他OCR系统，甚至可以与人类的识别能力相媲美。
- 多语种支持：笔声OCR支持多种语言文字的识别，包括中文、英文、日文、韩文等，可以满足不同场景下的需求。
- 鲁棒性强：笔声OCR能够在不同的光照、角度、噪声等环境下识别文字，且具有较好的鲁棒性，能够适应各种复杂的场景。
- 可定制性强：笔声OCR提供了丰富的预训练模型和模型组合方式，用户可以根据不同场景的需求自由组合选择合适的模型，从而实现更好的效果。
- 私有化：满足中大型B端G端要求，Docker化部署，支持K8S弹性拉伸; 私有化环境下确保不访问任何公网。
- 热词定义：支持热词定义，客户上传自己的热词，识别结果根据热词定义自动纠正。

总之，笔声OCR具有高精度、多语种支持、鲁棒性强、可定制性强和高效性等优势，是一个领先的OCR系统。

二、笔声OCR API文档

1、接口说明

- 通用文字识别（Universal Character Recognition），基于深度神经网络模型的端到端文字识别系统，将图片中印刷或手写的文字转化为计算机可编码的文字（目前支持中文、英文、日文、韩文等），可以满足不同场景下的需求。
- 集成通用文字识别时，需按照以下要求:

内容	说明
传输方式	http[s] (为提高安全性，强烈推荐https)
请求地址	http[s]: //api.abcpen.com/v1/ocr/file*注：服务器IP不固定，为保证您的接口稳定，请勿通过指定IP的方式调用接口，使用域名方式调用*
请求行	POST /v1/ocr/file HTTP/1.1
接口鉴权	签名机制，详情请参照下方[鉴权说明]

内容	说明
字符编码	UTF-8
响应格式	统一采用JSON格式
开发语言	任意，只要可以向笔声云服务发起HTTP请求的均可
适用范围	任意操作系统，但因不支持跨域不适用于浏览器
图片格式	jpg/jpeg/png/bmp
图片大小	大小不超过4M

2. 鉴权说明

在调用业务接口时，请求方需要对请求进行签名，服务端通过签名来校验请求的合法性。
鉴权根据application_id, application_secret, timestamp 和signature这几个参数做组合计算，其中application_id, application_secret请在笔声开放平台自主申请。鉴权如下

(1). Python示例代码

```python
def generate_signature(application_key: str, application_secret: str) -> str:
    timestamp: str = str(int(time.time()))
    message = f"{application_key}{timestamp}"
    signature = hmac.new(application_secret.encode("utf-8"), message.encode("utf-8"), hashlib.sha256).hexdigest()
    return signature
```

(2) . Java示例代码

```java
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.time.Instant;

public class SignatureGenerator {
    public static String generateSignature(String applicationKey, String applicationSecret) {
        String timestamp = String.valueOf(Instant.now().getEpochSecond());
        String message = applicationKey + timestamp;
        try {
            Mac sha256Hmac = Mac.getInstance("HmacSHA256");
            SecretKeySpec secretKey = new
SecretKeySpec(applicationSecret.getBytes(StandardCharsets.UTF_8), "HmacSHA256");
            sha256Hmac.init(secretKey);
            byte[] hmacDigest =
sha256Hmac.doFinal(message.getBytes(StandardCharsets.UTF_8));
            return bytesToHex(hmacDigest);
        } catch (NoSuchAlgorithmException | InvalidKeyException e) {
            e.printStackTrace();
            return null;
        }
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

(3). Kotlin示例代码

```
import java.security.MessageDigest
import javax.crypto.Mac
import javax.crypto.spec.SecretKeySpec
import java.time.Instant

fun generateSignature(applicationKey: String, applicationSecret: String): String {
    val timestamp = Instant.now().epochSecond.toString()
    val message = "$applicationKey$timestamp"
    val hmac = Mac.getInstance("HmacSHA256")
    hmac.init(SecretKeySpec(applicationSecret.toByteArray(Charsets.UTF_8),
"HmacSHA256"))
    val hmacDigest = hmac.doFinal(message.toByteArray(Charsets.UTF_8))
    return bytesToHex(hmacDigest)
}

private fun bytesToHex(bytes: ByteArray): String {
    val hexChars = CharArray(bytes.size * 2)
    for (i in bytes.indices) {
        val v = bytes[i].toInt() and 0xFF
        hexChars[i * 2] = hexArray[v.ushr(4)]
        hexChars[i * 2 + 1] = hexArray[v and 0x0F]
    }
    return String(hexChars)
}

private val hexArray = "0123456789abcdef".toCharArray()
```

(4). nodejs示例代码

```
const crypto = require('crypto');

function generateSignature(applicationKey, applicationSecret) {
  const timestamp = Math.floor(Date.now() / 1000).toString();
  const message = applicationKey + timestamp;
  const hmac = crypto.createHmac('sha256', applicationSecret);
  hmac.update(message);
  const signature = hmac.digest('hex');
  return signature;
}

// 示例使用
const applicationKey = 'your_application_key';
const applicationSecret = 'your_application_secret';
const signature = generateSignature(applicationKey, applicationSecret);
console.log(signature);
```

(5). go示例代码

```
package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "time"
)

func generateSignature(applicationKey string, applicationSecret string) string {
    timestamp := fmt.Sprintf("%d", time.Now().Unix())
    message := applicationKey + timestamp
    hmacKey := []byte(applicationSecret)
    hmacData := []byte(message)
    hmacSha256 := hmac.New(sha256.New, hmacKey)
    hmacSha256.Write(hmacData)
    signature := hex.EncodeToString(hmacSha256.Sum(nil))
    return signature
}
```

```
func main() {
    applicationKey := "your_application_key"
    applicationSecret := "your_application_secret"
    signature := generateSignature(applicationKey, applicationSecret)
    fmt.Println(signature)
}
```

(6) Rust示例代码

```
use hmac::{Hmac, Mac, NewMac};
use sha2::Sha256;
use std::time::{SystemTime, UNIX_EPOCH};

fn generate_signature(application_key: &str, application_secret: &str) -> String {
    let timestamp = SystemTime::now()
        .duration_since(UNIX_EPOCH)
        .unwrap()
        .as_secs()
        .to_string();
    let message = format!("{}{}", application_key, timestamp);
    let mut hmac =
        Hmac::<Sha256>::new_varkey(application_secret.as_bytes()).expect("HMAC initialization failed");
    hmac.update(message.as_bytes());
    let signature = hex::encode(hmac.finalize().into_bytes());
    signature
}

fn main() {
    let application_key = "your_application_key";
    let application_secret = "your_application_secret";
    let signature = generate_signature(application_key, application_secret);
    println!("{}", signature);
}
```

(7) vue示例代码

```
<template>
  <div>
    <button @click="generateSignature">Generate Signature</button>
    <p>Timestamp: {{ timestamp }}</p>
    <p>Signature: {{ signature }}</p>
  </div>
</template>

<script>
import crypto from 'crypto';
```

```
export default {
  name: 'SignatureGenerator',
  data() {
    return {
      applicationKey: 'test1',
      applicationSecret: '2258ACC4-199B-4DCB-B6F3-C2485C63E85A',
      timestamp: null,
      signature: null,
    };
  },
  methods: {
    generateSignature() {
      const timestamp = Math.floor(Date.now() / 1000).toString();
      const message = this.applicationKey + timestamp;
      const hmac = crypto.createHmac('sha256', this.applicationSecret);
      hmac.update(message);
      const signature = hmac.digest('hex');
      this.timestamp = timestamp;
      this.signature = signature;
    },
  },
};
</script>
```

2、鉴权访问

根据上述鉴权说明，生成X-App-Key， X-App-Signature和X-Timestamp这三个参数；将这三个参数放入http(s)请求体头部（header），向服务端发起请求。具体参考后面的示例代码。

3、接口访问

（1）上传热词文件

/v1/ocr/hotwords

- 常规参数，属于鉴权信息，生成X-App-Key， X-App-Signature和X-Timestamp这三个参数
- 具体参数定义

参数	类型	是否必须	默认值	备注
file	file	是	无	文本文件，扩展名是txt格式；每个热词一行，行与行之间用换行符分隔

- 说明，每个开发者一个热词文件，热词文件大小控制在4M以内。

(2) 上传图片识别

/v1/ocr/file

- 常规参数，属于鉴权信息，生成X-App-Key， X-App-Signature和X-Timestamp这三个参数
- 具体参数定义

参数	类型	是否必须	默认值	备注
file	file	是	无	图片文件； 支持jpg, jpeg, bmp, png, jp2等常规图像格式
confideceX	float	否	0.5	热词修正度； 在0~1之间，值越大，热词修正越高；值越小，热词修正度越少
use_hotwords	bool	否	False	为True，表示使用热词替换；为False，不使用热词替换

- 提交方式：multipart/form-data

(3) 示例代码

1). python示例

~~~
 ````python
 def test_ocr():
     application_key = "test1"
     application_secret = '2258ACC4-199B-4DCB-B6F3-C2485C63E85A'
     timestamp = str(int(time.time()))

     message = f"{application_key}{timestamp}"
     expected_signature = hmac.new(application_secret.encode("utf-8"),
message.encode("utf-8"), hashlib.sha256).hexdigest()
     headers = {
         "X-App-Key": application_key,
         "X-App-Signature": expected_signature,
         "X-Timestamp": timestamp,
     }

     file = {"file": open("./35.png", "rb")}
     payload = {"confidenceX": 0.5,
                "use_hotwords": False
                }

     response = requests.post(f"{BASE_URL}/ocr/file", files=file, headers=headers,
data=payload)

     print(response.json())
~~~

~~~~
 ````
~~~~

2). Java示例代码

~~~~
 ```java
 import java.io.File;
 import java.io.IOException;
 import java.nio.file.Files;
 import java.security.InvalidKeyException;
 import java.security.NoSuchAlgorithmException;
 import java.time.Instant;
 import javax.crypto.Mac;
 import javax.crypto.spec.SecretKeySpec;
 import okhttp3.MediaType;
 import okhttp3.MultipartBody;
 import okhttp3.OkHttpClient;
 import okhttp3.Request;
 import okhttp3.RequestBody;
 import okhttp3.Response;

 public class OcrTest {
     private static final String BASE_URL = "https://ocr.abcpen.com/v1";
     private static final String APPLICATION_KEY = "test1";
     private static final String APPLICATION_SECRET = "2258ACC4-199B-4DCB-B6F3-
C2485C63E85A";

     public static void main(String[] args) throws IOException,
NoSuchAlgorithmException, InvalidKeyException {
         String timestamp = String.valueOf(Instant.now().getEpochSecond());
         String message = APPLICATION_KEY + timestamp;
         Mac sha256Hmac = Mac.getInstance("HmacSHA256");
         SecretKeySpec secretKey = new SecretKeySpec(APPLICATION_SECRET.getBytes(),
"HmacSHA256");
         sha256Hmac.init(secretKey);
         String expectedSignature =
bytesToHex(sha256Hmac.doFinal(message.getBytes()));

         OkHttpClient client = new OkHttpClient();
         File file = new File("./35.png");
         RequestBody requestBody = new MultipartBody.Builder()
                 .setType(MultipartBody.FORM)
                 .addFormDataPart("file", file.getName(),
                         RequestBody.create(MediaType.parse("image/png"), file))
                 .addFormDataPart("confidencex", "0.5")
                 .addFormDataPart("use_hotwords", "false")
                 .build();
         Request request = new Request.Builder()
                 .url(BASE_URL + "/ocr/file")
                 .header("X-App-Key", APPLICATION_KEY)
                 .header("X-App-Signature", expectedSignature)
                 .header("X-Timestamp", timestamp)
~~~~

~~~
                .post(requestBody)
                .build();
        try (Response response = client.newCall(request).execute()) {
            System.out.println(response.body().string());
        }
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}

```
~~~

3). Kotlin示例代码

```
import java.io.File
import java.net.URL
import java.security.MessageDigest
import java.text.SimpleDateFormat
import java.util.*

val BASE_URL = "https://ocr.abcpen.com"
val APPLICATION_KEY = "test1"
val APPLICATION_SECRET = "2258ACC4-199B-4DCB-B6F3-C2485C63E85A"

fun main() {
    val timestamp = System.currentTimeMillis() / 1000
    val expectedSignature = generateSignature(APPLICATION_KEY, APPLICATION_SECRET,
timestamp)
    val headers = mapOf(
        "X-App-Key" to APPLICATION_KEY,
        "X-App-Signature" to expectedSignature,
        "X-Timestamp" to timestamp.toString()
    )

    val file = File("./35.png")
    val payload = mapOf(
        "confidenceX" to "0.5",
        "use_hotwords" to "false"
    )

    val response = URL("${BASE_URL}/ocr/file").openConnection().apply {
```

```
        setRequestProperty("Content-Type", "multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW")
        setRequestProperty("User-Agent", "Mozilla/5.0")
        setRequestProperty("Accept-Language", "en-US,en;q=0.5")
        setRequestProperty("Accept-Encoding", "gzip, deflate")
        headers.forEach { (key, value) -> setRequestProperty(key, value) }
        doOutput = true
    }.outputStream.use { output ->
        val boundary = "----WebKitFormBoundary7MA4YWxkTrZu0gW"
        output.write("\r\n--$boundary\r\n".toByteArray())
        output.write("Content-Disposition: form-data; name=\"file\"; filename=\"${file.name}\"\r\n".toByteArray())
        output.write("Content-Type: application/octet-stream\r\n\r\n".toByteArray())
        output.write(file.readBytes())
        output.write("\r\n--$boundary--\r\n".toByteArray())
        payload.forEach { (key, value) ->
            output.write("\r\n--$boundary\r\n".toByteArray())
            output.write("Content-Disposition: form-data; name=\"$key\"\r\n\r\n".toByteArray())
            output.write(value.toByteArray())
        }
        output.flush()
        responseCode
    }
    if (response == 200) {
        URL("${BASE_URL}/ocr/file").openConnection().apply {
            headers.forEach { (key, value) -> setRequestProperty(key, value) }
        }.inputStream.use { input ->
            println(input.bufferedReader().readText())
        }
    }
}

fun generateSignature(applicationKey: String, applicationSecret: String, timestamp: Long): String {
    val message = "$applicationKey$timestamp"
    val signature = MessageDigest.getInstance("SHA-256").apply {
        update(applicationSecret.toByteArray())
        update(message.toByteArray())
    }.digest().fold("") { str, byte ->
        str + "%02x".format(byte)
    }
    return signature
}
```

4). go示例代码

```
package main

import (
    "bytes"
    "crypto/hmac"
    "crypto/sha256"
    "fmt"
    "io/ioutil"
    "net/http"
    "os"
    "path/filepath"
    "strconv"
    "strings"
    "time"
)

const (
    baseURL           = "https://ocr.abcpen.com/v1"
    applicationKey    = "test1"
    applicationSecret = "2258ACC4-199B-4DCB-B6F3-C2485C63E85A"
)

func generateSignature(applicationKey string, applicationSecret string, timestamp
string) string {
    message := applicationKey + timestamp
    mac := hmac.New(sha256.New, []byte(applicationSecret))
    mac.Write([]byte(message))
    expectedSignature := mac.Sum(nil)
    return fmt.Sprintf("%x", expectedSignature)
}

func main() {
    timestamp := strconv.FormatInt(time.Now().Unix(), 10)
    expectedSignature := generateSignature(applicationKey, applicationSecret,
timestamp)

    file, err := os.Open("./35.png")
    if err != nil {
        panic(err)
    }
    defer file.Close()

    body := &bytes.Buffer{}
    writer := multipart.NewWriter(body)
    part, err := writer.CreateFormFile("file", filepath.Base(file.Name()))
    if err != nil {
        panic(err)
    }
    _, err = io.Copy(part, file)
    if err != nil {
        panic(err)
    }
    writer.WriteField("confidenceX", "0.5")
    writer.WriteField("use_hotwords", "false")
    err = writer.Close()
    if err != nil {
        panic(err)
    }

    req, err := http.NewRequest("POST", baseURL+"/ocr/file", body)
    if err != nil {
        panic(err)
    }
    req.Header.Set("Content-Type", writer.FormDataContentType())
    req.Header.Set("X-App-Key", applicationKey)
    req.Header.Set("X-App-Signature", expectedSignature)
    req.Header.Set("X-Timestamp", timestamp)

    client := &http.Client{}
    resp, err := client.Do(req)
    if err != nil {
        panic(err)
    }
    defer resp.Body.Close()

    respBody, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        panic(err)
    }
    fmt.Println(string(respBody))
}
```

5). Rust代码

~~~
 ```rust
 use hmac::{Hmac, Mac, NewMac};
 use reqwest::{header, Client, Response};
 use std::fs::File;
 use std::io::Read;
 use std::time::{SystemTime, UNIX_EPOCH};

 const BASE_URL: &str = "https://ocr.abcpen.com/v1";
 const APPLICATION_KEY: &str = "test1";
 const APPLICATION_SECRET: &str = "2258ACC4-199B-4DCB-B6F3-C2485C63E85A";

 type HmacSha256 = Hmac<sha2::Sha256>;

 fn generate_signature(application_key: &str, application_secret: &str, timestamp: &str) -> String {
~~~

```
     let message = format!("{}{}", application_key, timestamp);
     let mut mac = HmacSha256::new_varkey(application_secret.as_bytes()).unwrap();
     mac.update(message.as_bytes());
     let expected_signature = mac.finalize().into_bytes();
     hex::encode(&expected_signature)
 }

 fn main() -> Result<(), Box<dyn std::error::Error>> {
     let timestamp = SystemTime::now()
         .duration_since(UNIX_EPOCH)?
         .as_secs()
         .to_string();
     let expected_signature =
         generate_signature(APPLICATION_KEY, APPLICATION_SECRET, &timestamp);

     let file = File::open("./35.png")?;
     let mut buf_reader = std::io::BufReader::new(file);
     let mut file_content = Vec::new();
     buf_reader.read_to_end(&mut file_content)?;

     let client = Client::new();
     let response = client
         .post(&format!("{}/ocr/file", BASE_URL))
         .header(header::CONTENT_TYPE, "multipart/form-data")
         .header("X-App-Key", APPLICATION_KEY)
         .header("X-App-Signature", expected_signature)
         .header("X-Timestamp", timestamp)
         .multipart(
             reqwest::multipart::Form::new()
                 .part("file",
reqwest::multipart::Part::bytes(file_content).file_name("35.png"))
                 .part("confidenceX", reqwest::multipart::Part::text("0.5"))
                 .part("use_hotwords", reqwest::multipart::Part::text("false")),
         )
         .send()?;

     println!("{}", response.text()?);
     Ok(())
 }
```

6). nodejs示例代码

```
const crypto = require('crypto');
const fs = require('fs');
const path = require('path');
const FormData = require('form-data');
const fetch = require('node-fetch');

const BASE_URL = 'https://ocr.abcpen.com/v1';
const APPLICATION_KEY = 'test1';
const APPLICATION_SECRET = '2258ACC4-199B-4DCB-B6F3-C2485C63E85A';
```

```
function generateSignature(applicationKey, applicationSecret, timestamp) {
  const message = applicationKey + timestamp;
  const hmac = crypto.createHmac('sha256', applicationSecret);
  hmac.update(message);
  const expectedSignature = hmac.digest('hex');
  return expectedSignature;
}

(async function () {
  const timestamp = Math.floor(Date.now() / 1000).toString();
  const expectedSignature = generateSignature(
    APPLICATION_KEY,
    APPLICATION_SECRET,
    timestamp
  );

  const form = new FormData();
  const fileStream = fs.createReadStream('./35.png');
  form.append('file', fileStream, {
    filename: path.basename(fileStream.path),
  });
  form.append('confidenceX', '0.5');
  form.append('use_hotwords', 'false');

  const response = await fetch(`${BASE_URL}/ocr/file`, {
    method: 'POST',
    headers: {
      'X-App-Key': APPLICATION_KEY,
      'X-App-Signature': expectedSignature,
      'X-Timestamp': timestamp,
      ...form.getHeaders(),
    },
    body: form,
  });
  const result = await response.text();
  console.log(result);
})();
```

7). vue示例代码

~~~
 ```vue
<template>
  <div>
    <input type="file" @change="onFileSelected">
    <button @click="onUploadClick">Upload</button>
  </div>
</template>

<script>
~~~

```
import crypto from 'crypto';
import FormData from 'form-data';
import axios from 'axios';

const BASE_URL = 'https://ocr.abcpen.com/v1';
const APPLICATION_KEY = 'test1';
const APPLICATION_SECRET = '2258ACC4-199B-4DCB-B6F3-C2485C63E85A';

export default {
  name: 'UploadForm',
  data() {
    return {
      selectedFile: null,
      confidenceX: 0.5,
      useHotwords: false,
    };
  },
  methods: {
    async onUploadClick() {
      if (!this.selectedFile) {
        return;
      }
      const timestamp = Math.floor(Date.now() / 1000).toString();
      const expectedSignature = this.generateSignature(
        APPLICATION_KEY,
        APPLICATION_SECRET,
        timestamp
      );
      const form = new FormData();
      form.append('file', this.selectedFile);
      form.append('confidenceX', this.confidenceX.toString());
      form.append('use_hotwords', this.useHotwords.toString());
      try {
        const response = await axios.post(`${BASE_URL}/ocr/file`, form, {
          headers: {
            'X-App-Key': APPLICATION_KEY,
            'X-App-Signature': expectedSignature,
            'X-Timestamp': timestamp,
            'Content-Type': 'multipart/form-data',
          },
        });
        console.log(response.data);
      } catch (error) {
        console.error(error);
      }
    },
    onFileSelected(event) {
      this.selectedFile = event.target.files[0];
    },
    generateSignature(applicationKey, applicationSecret, timestamp) {
      const message = applicationKey + timestamp;
      const hmac = crypto.createHmac('sha256', applicationSecret);
```

~~~
      hmac.update(message);
      const expectedSignature = hmac.digest('hex');
      return expectedSignature;
    },
  },
};
</script>

```
~~~

4、错误码

错误码	描述	说明	处理方式
0	success	成功	
-1	in progress	识别中	请继续重试
-2	audio encode error	音频编码错误	请编码成正确的格式，再提交请求
10105	illegal access	没有权限	检查apiKey，ip，ts等授权参数是否正确
10106	invalid parameter	无效参数	上传必要的参数， 检查参数格式以及编码
10107	illegal parameter	非法参数值	检查参数值是否超过范围或不符合要求
10110	no license	无授权许可	检查参数值是否超过范围或不符合要求
10700	engine error	引擎错误	提供接口返回值，向服务提供商反馈
16003	basic component error	基础组件异常	重试或向服务提供商反馈
10800	over max connect limit	超过授权的连接数	确认连接数是否超过授权的连接数

三、应用场景

1. 手写体识别：笔声OCR可以对手写体进行识别，如手写数字、手写汉字等。
2. 印刷体识别：笔声OCR可以识别印刷体文字，包括图书、文件、票据、合同、名片等。
3. 表格识别：笔声OCR可以识别表格中的文字、数字和符号等信息，并能够将其转换为电子表格或数据库格式。
4. 车牌识别：笔声OCR可以对车牌号码进行识别，并可应用于停车场、高速公路等场景。
5. 人脸识别：笔声OCR可以识别人脸上的文字、数字和符号等信息，可用于签到、门禁等场景。
6. 视频识别：笔声OCR可以识别视频中的文字信息，如广告牌、字幕等。

四、价格套餐

	免费套餐	套餐一	套餐二	套餐三
服务量	10万	1万	10万	100万
有效期	90天	一年	一年	一年
单价（万次）	免费	￥175.00	￥1120.00	￥10500.00
立即购买	申请链接	购买链接	购买链接	购买链接

注意：免费包的服务量为10万次，有效期为90天，单价为免费。免费包的服务量为10万次，有效期为90天，单价为免费。套餐一、二、三的服务量分别为1万次、10万次、100万次，有效期均为一年，单价分别为￥175.00、￥1120.00、￥10500.00，用户可以根据自身需求选择合适的套餐。