AudioSer优点

1. 模块化设计：代码被拆分成多个文件和函数，每个函数都专注于特定的任务，让代码可以易于维护和修改。
2. 缓存优化：通过Flask-Caching插件实现缓存，避免了重复计算和磁盘I/O，提高了API响应速度，支持多调用。
3. 异常处理：代码有异常处理，校验文件类型、处理文件不存在的情况下正常运行，增加代码运行的可靠性。
4. 日志记录：使用内置的logging模块，将异常信息写入日志文件，方便问题追踪和排查。
5. 安全性考虑：避免在前端或提交音频文件时，上传其它文件通过限制上传文件的类型，防止安全问题，如上传恶意可执行脚本文件（防护机制不安全但是必须要的）当服务器接收到音频识别后会立即删除服务器上传的文件，避免webshell存在的可能。
6. 严格文件类型检查：通过检查文件的扩展名、MIME类型以及文件内容的特征，来判断上传的文件是否为.wav格式的音频文件，防止了非法文件的上传。

   攻防无绝对安全，问题肯定会有的，我能做的就是亡羊补牢。

API接口

```
http://127.0.0.1:5620/voice
method: POST
file: 1.wav
// 音频以字节流的方式读取提交
```

curl

```
curl -F "file=@E:\Desktop\1.wav" http://127.0.0.1:5620/voice
```

Python

```python
import requests

url = 'http://127.0.0.1:5620/voice'
file = open('E:/Desktop/1.wav', 'rb')
files = {'file': ('1.wav', file)}
response = requests.post(url, files=files)
print(response.json()['message'])
file.close()
```

Golang

```go
package main

import (
    "bytes"
    "fmt"
```

```
    "io"
    "io/ioutil"
    "mime/multipart"
    "net/http"
    "os"
)

func main() {
    url := "http://127.0.0.1:5620/voice"
    file, err := os.Open("E:/Desktop/1.wav")
    if err != nil {
        fmt.Println(err)
        return
    }
    defer file.Close()
    requestBody := &bytes.Buffer{}
    writer := multipart.NewWriter(requestBody)
    part, err := writer.CreateFormFile("file", "1.wav")
    if err != nil {
        fmt.Println(err)
        return
    }
    _, err = io.Copy(part, file)
    if err != nil {
        fmt.Println(err)
        return
    }
    writer.Close()
    request, err := http.NewRequest("POST", url, requestBody)
    if err != nil {
        fmt.Println(err)
        return
    }
    request.Header.Set("Content-Type", writer.FormDataContentType())
    client := http.Client{}
    response, err := client.Do(request)
    if err != nil {
        fmt.Println(err)
        return
    }
    defer response.Body.Close()
    responseBody, err := ioutil.ReadAll(response.Body)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(string(responseBody))
}
```