

身份证OCR识别服务

这是一个基于Python的身份证OCR识别服务端，提供REST API接口，支持高并发处理和性能优化。

项目介绍

这是一个基于Python开发的身份证OCR识别服务，可以通过API接口识别身份证上的文字信息，支持并发处理多个请求。

主要功能：

- 身份证正面识别（姓名、性别、民族、出生日期、住址、身份证号）
- 身份证背面识别（签发机关、有效期限）
- 支持批量识别多张身份证图片
- 提供REST API接口，方便集成到其他系统
- 支持Base64编码和文件上传两种方式提交图片

环境要求

🔧 基础要求

- Python 3.7或更高版本
- Windows、Linux或macOS操作系统

💻 硬件要求（根据使用场景选择）

使用场景	CPU	内存	并发能力	适用说明
🧪 开发测试	2核+	2-4GB	2并发	功能验证、开发调试
💾 内存受限	4核+	4-8GB	4并发	小型部署、内存有限
🔧 生产推荐	4-8核	8-16GB	8并发	一般生产环境
🚀 高性能	8核+	16GB+	16并发+	高并发生产环境

💡 配置优化提示：详细的性能优化配置请参考 [📖 配置优化指南](#)

功能特点

- 🎯 **高精度识别**：基于PaddleOCR的专业身份证识别
- 🚀 **高并发支持**：多进程架构，支持高并发请求处理
- 🔧 **性能可调**：丰富的配置选项，适应不同硬件环境
- 📄 **RESTful API**：标准化接口，易于集成
- 📊 **详细日志**：完整的请求日志和错误跟踪
- 💾 **内存优化**：智能内存管理，适合资源受限环境
- 🔒 **安全认证**：支持API密钥验证

-  **多种格式**: 支持Base64和文件上传两种方式

技术栈

- **FastAPI**: 现代高性能Web框架
- **PaddleOCR**: 百度开源OCR引擎
- **OpenCV**: 图像处理库
- **Uvicorn**: 高性能ASGI服务器
- **Loguru**: 现代Python日志库

快速开始

1. 安装Python环境

如果您还没有安装Python，请先从[Python官网](#)下载并安装Python 3.7或更高版本。

2. 下载项目代码

将项目代码下载到本地，或者直接使用git克隆：

```
git clone https://github.com/yourusername/sfzocr.git
cd sfzocr
```

3. 安装依赖包

在项目根目录下执行以下命令安装所需的依赖包：

```
pip install -r requirements.txt
```

 **安装提示**：安装过程可能需要几分钟时间，特别是PaddlePaddle和PaddleOCR这两个包比较大。如果安装PaddlePaddle时遇到问题，可以参考[PaddlePaddle官方安装指南](#)。

4. 配置优化（推荐）

根据您的服务器性能选择合适的配置：

```
# 查看当前配置和建议
python app/config.py

# 复制配置模板
cp config_template.env .env

# 编辑配置文件，选择适合您服务器的配置方案
# 详细说明请参考：CONFIG_OPTIMIZATION.md
```

5. 启动服务

开发环境启动

```
python run.py --debug
```

生产环境启动

```
# 使用默认配置（推荐让配置文件决定）
python run.py

# 或者手动指定参数
python run.py --host 0.0.0.0 --port 8080 --workers 4
```

启动参数说明：

- `--host`：服务监听的IP地址，0.0.0.0表示监听所有网络接口
- `--port`：服务监听的端口号
- `--workers`：工作进程数量（会覆盖配置文件设置）
- `--debug`：是否启用调试模式（开发环境使用）
- `--log-level`：日志级别（DEBUG/INFO/WARNING/ERROR/CRITICAL）

⚙️ 配置说明

本项目支持灵活的配置方式，可以通过环境变量或配置文件进行定制。

🎯 快速配置

根据您的服务器性能，选择预设的配置方案：

```
# 方案A：高性能服务器（16GB+内存，8核+CPU）
export WORKERS=8
export OCR_PROCESS_POOL_SIZE=8
export MAX_CONCURRENT_REQUESTS=16
export MEMORY_OPTIMIZATION=False

# 方案B：内存受限服务器（4-8GB内存，4核CPU）
export WORKERS=2
export OCR_PROCESS_POOL_SIZE=2
export MAX_CONCURRENT_REQUESTS=4
export MEMORY_OPTIMIZATION=True

# 方案C：生产环境推荐（8-16GB内存，4-8核CPU）
export WORKERS=4
export OCR_PROCESS_POOL_SIZE=4
export MAX_CONCURRENT_REQUESTS=8
export MEMORY_OPTIMIZATION=True
export API_KEYS="your-secret-key-1,your-secret-key-2"
```

核心性能参数

参数	说明	推荐值	性能影响
<code>WORKERS</code>	Uvicorn工作进程数	CPU核心数	影响并发处理能力
<code>OCR_PROCESS_POOL_SIZE</code>	OCR处理进程池大小	2-8	每个进程约占1GB内存
<code>MAX_CONCURRENT_REQUESTS</code>	最大并发请求数	OCR进程数×2	防止内存溢出
<code>OCR_TASK_TIMEOUT</code>	OCR任务超时时间(秒)	15-60	防止任务卡死

内存优化参数

参数	说明	推荐场景
<code>MEMORY_OPTIMIZATION</code>	启用内存优化	内存<16GB时启用
<code>ENABLE_GC_AFTER_REQUEST</code>	请求后强制垃圾回收	生产环境推荐启用

安全配置

参数	说明	示例
<code>API_KEYS</code>	API密钥列表（逗号分隔）	<code>"key1, key2, key3"</code>
<code>DEBUG</code>	调试模式	生产环境设为False

日志配置

参数	说明	推荐值
<code>LOG_LEVEL</code>	日志级别	生产:WARNING, 开发:DEBUG
<code>LOG_ROTATION</code>	日志轮转大小	20-50MB
<code>LOG_RETENTION</code>	日志保留时间	1 week - 1 month

 详细配置指南：更多配置选项和优化建议请参考 [CONFIG_OPTIMIZATION.md](#)

配置工具

配置检查和摘要

查看当前配置摘要和优化建议
`python app/config.py`

配置模板

项目提供了 `config_template.env` 配置模板，包含不同场景的预设配置：

```
# 复制并编辑配置模板
cp config_template.env .env
# 编辑 .env 文件，选择适合的配置方案
```



使用API接口

服务启动后，可以通过以下方式使用API：

1. 查看API文档

在浏览器中访问：`http://服务器IP:端口/docs`

例如：`http://localhost:8000/docs`

这里提供了交互式的API文档，使用全中文界面，可以直接在页面上测试API功能。文档包含了所有API的详细说明、请求参数和响应格式。

如果您喜欢另一种文档风格，也可以访问：`http://服务器IP:端口/redoc`

2. ID 身份证识别API

单张身份证识别 (Base64方式)

请求地址：`/api/v1/ocr/idcard`

请求方法：POST

请求参数：

```
{
  "image": "base64编码的图片数据",
  "side": "front" // 可选值: front (正面)、back (背面)
}
```

响应结果：

```
{
  "code": 0,
  "message": "识别成功",
  "data": {
    "name": "张三",
    "sex": "男",
    "nation": "汉",
    "birth": "1990-01-01",
    "address": "北京市朝阳区...",
    "id_number": "110101199001010123",
    "issue_authority": null,
    "valid_period": null
  }
}
```

```
}
```

单张身份证识别 (文件上传方式)

请求地址: `/api/v1/ocr/idcard/upload`

请求方法: POST

请求参数:

- `image`: 上传的身份证图片文件 (form-data格式)
- `side`: 身份证正反面, 可选值: front (正面)、back (背面)

请求示例:

```
curl -X POST "http://localhost:8000/api/v1/ocr/idcard/upload" \
-H "accept: application/json" \
-F "image=@/path/to/idcard.jpg" \
-F "side=front"
```

响应结果: 与Base64方式相同

批量身份证识别 (Base64方式)

请求地址: `/api/v1/ocr/idcard/batch`

请求方法: POST

请求参数:

```
{
  "images": [
    {
      "image": "base64编码的图片数据1",
      "side": "front"
    },
    {
      "image": "base64编码的图片数据2",
      "side": "back"
    }
  ]
}
```

批量身份证识别 (文件上传方式)

请求地址: `/api/v1/ocr/idcard/batch/upload`

请求方法: POST

请求参数:

- `front_image`: 上传的身份证正面图片文件 (可选)
- `back_image`: 上传的身份证背面图片文件 (可选)

3. 🔍 健康检查API

请求地址: `/api/v1/health`

请求方法: GET

响应结果:

```
{
  "code": 0,
  "message": "服务正常",
  "data": {
    "status": "healthy",
    "version": "0.1.2",
    "timestamp": 1626345678
  }
}
```

🧪 使用测试脚本

项目提供了一个测试脚本 `test_api.py`，可以用来测试API接口是否正常工作。

测试健康检查API

```
python test_api.py --health
```

测试单张身份证识别

测试身份证正面识别 (Base64方式)

```
python test_api.py --image 身份证正面图片路径.jpg --side front
```

测试身份证背面识别 (Base64方式)

```
python test_api.py --image 身份证背面图片路径.jpg --side back
```

测试身份证正面识别 (文件上传方式)

```
python test_api.py --image 身份证正面图片路径.jpg --side front --upload
```

测试身份证背面识别 (文件上传方式)

```
python test_api.py --image 身份证背面图片路径.jpg --side back --upload
```

测试批量身份证识别

测试批量身份证识别 (Base64方式)

```
python test_api.py --front 身份证正面图片路径.jpg --back 身份证背面图片路径.jpg
```

测试批量身份证识别 (文件上传方式)

```
python test_api.py --front 身份证正面图片路径.jpg --back 身份证背面图片路径.jpg --upload
```

自定义API地址


如果服务不是运行在默认地址，可以使用 `--url` 参数指定API地址：

```
python test_api.py --url http://服务器IP:端口/api/v1 --health
```

图片要求

为了获得最佳识别效果，上传的身份证图片应满足以下要求：

1. **图片清晰**：无严重模糊、反光
2. **完整显示**：身份证完整显示在图片中
3. **光线均匀**：无过度曝光或过暗现象
4. **主体突出**：身份证尽量占据图片的主要部分

 **处理能力**：服务会对图片进行预处理（包括旋转校正、裁剪等），但原始图片质量仍然是影响识别准确率的关键因素。

常见问题解答

1. 服务启动失败

问题：运行 `python run.py` 后，服务无法正常启动。

解决方法：

- 查看启动时的配置信息，确认内存是否足够
- 检查日志文件（logs目录下）查看具体错误信息
- 确认所有依赖包已正确安装：`pip list | grep paddle`
- 确认Python版本是3.7或更高：`python --version`
- 如果是内存不足，尝试使用内存受限配置：

```
export WORKERS=1
export OCR_PROCESS_POOL_SIZE=1
python run.py
```

2. 识别准确率不高

问题：身份证信息识别不准确或识别不全。

解决方法：

- 提高上传图片的质量和清晰度
- 确保身份证在图片中完整显示
- 避免图片中有复杂背景或其他干扰元素
- 调整光线，避免反光和阴影
- 图片尺寸建议不超过2000x2000像素

3. 服务响应慢

问题：API请求响应时间过长。

解决方法：

- 查看当前配置： `python app/config.py`
- 根据服务器性能增加OCR进程数：

```
export OCR_PROCESS_POOL_SIZE=4 # 根据内存调整
export MAX_CONCURRENT_REQUESTS=8
```

- 减小上传图片的尺寸
- 升级服务器硬件配置
- 使用高性能配置方案（参考配置优化指南）

4. 内存占用过高

问题：服务运行一段时间后内存占用过高。

解决方法：

- 启用内存优化配置：

```
export MEMORY_OPTIMIZATION=True
export ENABLE_GC_AFTER_REQUEST=True
```

- 减少进程数量：

```
export WORKERS=2
export OCR_PROCESS_POOL_SIZE=2
```

- 使用内存受限配置方案
- 定期重启服务或增加服务器内存

5. 422参数验证错误

问题：请求返回422状态码。

解决方法：

- 确保使用正确的请求格式（multipart/form-data用于文件上传）
- 检查 `side` 参数值是否为 `front`、`back` 或 `both`
- 确认文件参数名为 `image`
- 如果配置了API密钥，确认提供了 `X-API-KEY` 头部
- 查看详细错误信息（新版本会提供详细的验证失败信息）

6. 上传图片失败

问题：使用Base64方式上传图片时出现编码错误。

解决方法：

- 尝试使用文件上传API（`/api/v1/ocr/idcard/upload`）代替Base64编码方式
- 确保Base64编码正确，不要包含换行符
- 检查图片格式是否支持（支持JPG、PNG、BMP等常见格式）

性能监控

内存和CPU监控

```
# 查看进程内存使用
ps aux | grep python

# 实时系统监控
htop

# 查看系统内存
free -h
```




日志监控

```
# 实时查看日志
tail -f logs/sfzocr.log

# 查看错误统计
grep ERROR logs/sfzocr.log | wc -l

# 查看最近的错误
tail -n 100 logs/sfzocr.log | grep ERROR
```

文档资源

-  [配置优化指南](#) - 详细的性能调优说明
-  [配置模板](#) - 不同场景的配置示例
-  [API文档](#) - 交互式API文档（需先启动服务）

安全建议

生产环境部署

1. 配置API密钥：

```
export API_KEYS="your-strong-secret-key-1,your-strong-secret-key-2"
```

2. 关闭调试模式：

```
export DEBUG=False
```

3. 配置适当的日志级别：

```
export LOG_LEVEL=WARNING
```

4. 限制网络访问：

- 配置防火墙规则
- 使用反向代理（如Nginx）
- 配置HTTPS

技术支持

如果您在使用过程中遇到任何问题，可以通过以下方式获取帮助：

1. 查看配置和日志：

```
python app/config.py # 查看当前配置  
tail -f logs/sfzocr.log # 查看日志
```

2. 参考文档：

- [配置优化指南](#)
- [GitHub Issues](#)

3. 联系支持：

- GitHub项目页面提交Issue
- 发送邮件至技术支持邮箱

许可证

本项目采用MIT许可证，详情请参阅LICENSE文件。

 持续优化中，建议关注项目更新获取最新功能和性能改进。