目录

[Optical Character Recognition 2](#_Toc181997866)

[1. 分解结构 2](#_Toc181997867)

[2. 使用结构 2](#_Toc181997868)

[3. 层结构 2](#_Toc181997869)

[4. 类结构 2](#_Toc181997870)

[5. 数据模型 3](#_Toc181997871)

[服务器端计算 3](#_Toc181997872)

[并行计算 3](#_Toc181997873)

[服务器在局域网（LAN）和外部网络的差异 3](#_Toc181997874)

[软件体系结构设计 4](#_Toc181997875)

[代码模块结构 4](#_Toc181997876)

[1. 前端模块 4](#_Toc181997877)

[2. 后端模块 4](#_Toc181997878)

[3. 数据存储模块 5](#_Toc181997879)

[4. DevOps模块 5](#_Toc181997880)

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 邮箱 | 手机 |
| 石夏源（组长） | 202230091036 | [3162902145@qq.com](mailto:3162902145@qq.com) | 19513736825 |
| 黄超平 | 202230480465 |  |  |
| 陈金辉 | 202230480137 |  |  |
| 赖健康 | 202230481103 |  |  |
| 陈津乐 | 202230490013 |  |  |

## Optical Character Recognition

对于光学字符识别（OCR）应用程序，模块的设计和结构对于提高效率、可维护性和可扩展性至关重要。

### 1. 分解结构

计划将OCR应用程序，划分为以下功能模块：

1. **图像获取模块**：处理图像数据的输入，可能来自扫描仪、相机或图像文件。
2. **预处理模块**：负责改善图像质量以便识别。包括降噪、规范化、二值化和校正倾斜。
3. **字符分割模块**：分割图像以识别和隔离字符或单词。
4. **识别模块**：将分割后的图像转换为文本字符，该模块实现模式识别算法或机器学习模型。
5. **后处理模块**：使用上下文或基于词典的校正来纠正常见的识别错误，提高输出文本的准确性。
6. **输出模块**：将识别的文本以格式化的方式展示给用户或发送给其他应用程序，也可能处理各种格式的导出。

### 2. 使用结构

1. **图像获取模块** 使用 **预处理模块** 来在处理前清洁图像。
2. **预处理模块** 使用 **字符分割模块** 来传递预处理后的图像进行分割。
3. **字符分割模块** 使用 **识别模块** 将分割的字符图像发送进行识别。
4. **识别模块** 使用 **后处理模块** 优化其输出。
5. **输出模块** 使用来自 **后处理模块** 的数据生成最终输出。

### 3. 层结构

1. **接口层**：处理与外部系统或用户的交互，管理输入和输出。
2. **处理层**：包括预处理、字符分割、识别和后处理模块，按顺序处理数据。
3. **数据访问层**：管理所需数据的存储和检索，如模型参数、字典和用户设置。

### 4. 类结构

在面向对象设计中，每个模块实现为一个类或一个包（预期规划）：

1. **ImageReader**：读取各种来源图像。
2. **ImageProcessor**：用于图像调整和转换。
3. **Segmenter**：将图像分解成可识别字符或单词。
4. **Recognizer**：实现识别算法。
5. **Corrector**：用于错误校正和上下文感知改进。
6. **OutputHandler**：用于格式化和输出数据。

### 5. 数据模型

虽然对OCR任务来说不是主要关注点，但数据模型可能有助于处理用户设置或存储OCR模板和结果：

1. **用户设置**：存储用户首选项和配置。
2. **模板**：管理不同类型文件的预定义OCR模板。
3. **OCR结果**：存档OCR输出及其元数据，如时间戳和源信息。

### 服务器端计算

对于OCR系统，特别是处理高解析度或大量图像的场景，需要服务器端进行大量计算。以下模块会部署在计算能力更优的服务器上以利用服务端的计算资源：

1. **预处理模块**：服务器可以更快地处理图像的清洁、规范化等操作，尤其是在并行处理多个图像时。
2. **字符分割模块** 和 **识别模块**：这些模块在处理大数据量时尤其受益于服务器的高性能计算能力。利用GPU加速或其他高性能计算资源，可以显著提高处理速度。
3. **后处理模块**：在服务器上运行可以利用复杂的算法来校正识别错误，这些算法需要较大的计算资源。

### 并行计算

并行计算可以显著加快OCR应用程序的处理速度，在处理层和数据访问层中，考虑在多核服务器或通过分布式计算环境中运行，各个模块可以同时处理多个任务，从而减少总处理时间。

### 服务器在局域网（LAN）和外部网络的差异

1. **局域网服务器**：通常具有更低的延迟和更高的数据传输速率。适合部署在对实时处理和数据安全性要求较高的内部网络中。
2. **外部网络服务器**（如云服务器）：提供更好的可扩展性和灵活性，可以根据需求动态调整资源。但可能涉及更高的延迟和数据传输安全性问题。

### 软件体系结构设计

1. **微服务架构**：将OCR系统的各个模块（如图像获取、预处理、识别等）设计为独立的微服务，不同的服务器或容器（后续开展实践还是使用容器）中独立部署和扩展这些服务。
2. **容错和负载均衡**：设计时考虑包括负载均衡器和容错机制，确保系统在高负载或部分服务器故障时仍能保持高可用性和性能。

## 代码模块结构

### 1. 前端模块

**前端界面**：

1. **用户界面（UI）模块**：负责渲染用户界面，包括图像上传界面、处理状态显示、以及结果展示。
2. **前端逻辑模块**：处理用户输入，如上传的图像数据，以及与后端的API通信逻辑。

**技术栈**：

1. HTML/CSS/JavaScript
2. 前端框架使用Vue.js，用于构建用户界面和处理前端逻辑。

### 2. 后端模块

**核心OCR处理服务**：

1. **图像接收与验证模块**：接收前端发送的图像文件，验证文件类型和大小。
2. **图像预处理服务**：进行图像清洁、规范化和二值化等操作。
3. **图像分割服务**：将预处理后的图像分割成单独的字符或文字区块。
4. **字符识别服务**：将分割的图像转换为文本数据。
5. **后处理服务**：对识别出的文本进行校对和优化，提高识别准确率。
6. **结果输出服务**：将识别结果格式化后返回给前端。

**API层**：

1. **RESTful API**：提供与前端通信的接口，用于图像上传和结果回传。
2. **API网关**：作为单一入口点管理所有后端服务的API调用，提供路由、负载均衡和安全控制。

**技术栈**：

1. Python/Flask 或 Django，用于编写服务和API。
2. TensorFlow, PyTorch 或其他机器学习库，用于执行图像处理和识别任务。

### 3. 数据存储模块

1. **数据库服务**：存储用户数据、应用配置和历史处理结果。
2. **文件存储服务**：存储上传的原始图像和处理过程中生成的数据。

**技术栈**：

1. 关系数据库MySQL\NoSQL数据库MongoDB。
2. 文件存储解决方案Amazon S3更优,但本项目使用本地文件系统。

### 4. DevOps模块

1. **容器化**：使用Docker容器化所有服务，确保环境一致性和可移植性。
2. **自动化部署**：利用Kubernetes进行服务的部署、扩展和管理。
3. **持续集成/持续部署（CI/CD）**：配置CI/CD流水线自动化测试和部署过程。