1引言 2

1.1编写目的 2

1.2背景 2

1.3定义 2

1.4参考资料 2

2总体设计 3

2.1需求规定 3

2.2运行环境 3

2.3基本设计概念和处理流程 4

2.4结构 5

2.5功能器求与程序的关系 6

2.6人工处理过程 6

2.7尚未问决的问题 7

3接口设计 7

3.1用户接口 7

3.2外部接口 8

3.3内部接口 8

4运行设计 8

4.1运行模块组合 8

4.2运行控制 8

4.3运行时间 9

5系统数据结构设计 9

5.1逻辑结构设计要点 9

5.2物理结构设计要点 9

5.3数据结构与程序的关系 10

6系统出错处理设计 10

6.1出错信息 10

6.2补救措施 10

6.3系统维护设计 11

**概要设计说明书**

1**引言**

1.1**编写目的**

本文档对车牌识别系统的概要设计，说明整个软件系统的结构，明确系统各个模块的处理流程和模块交互之间采用的数据结构及协议。文档面向软件的开发和测试人员，软件开发者根据本文档对软件进行进一步的细化设计，然后进行开发工作，软件测试人员依照本文档对系统的功能模块进行测试工作，保障系统能够按照设计要求正确工作。

1.2**背景**

* 软件名称：车牌识别系统
* 开发者：
* 用户：交通执法人员、交通保险现场处置人员以及希望在其开发的软件中集成车牌识别功能的软件开发人员。

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 职责 |
| 王鹏 |  |  |
| 王振宁 |  |  |
| 刘文楠 |  |  |
| 南梦瑶 |  |  |
| 谢锦源 |  |  |

* 运行环境：

服务端：计算机集群

客户端：具备图像采集和网络访问能力的任意终端设备

1.3**定义**

无

1.4**参考资料**

[1]张海藩.软件工程导论[M].2008

[2]JSON-RPC Working Group.JSON-RPC 2.0 Specification[S].2013

[3]（英）SAM NEWMAN 著；崔力强，张骏译.微服务设计[M].2016

2**总体设计**

2.1**需求规定**

系统包括任务接收、任务分发、任务管理、车牌识别和系统管理五个主要的功能模块，各个功能模块的结构功能如下：

* 任务接收

任务接受模块处理需要具备以下两个功能：

* + 当客户端请求新建一个任务时，向客户端返回图片上传路径以及任务编号。
  + 当客户端上传图片完毕，并向服务端发送确认信息后，向队列推送新任务。
* 任务分发

任务分发模块负责对消息队列进行监控，当有新的任务时，根据计算节点的负载情况，向适当的计算节点分发计算任务，或者选择新增计算节点。

* 任务管理

任务管理模块负责处理对系统中任务的查询、删除操作。

* 车牌识别

车牌识别模块是系统中的主要功能模块，负责对接收到的图片进行车牌的定位、识别，并输出识别结果。

* 系统管理

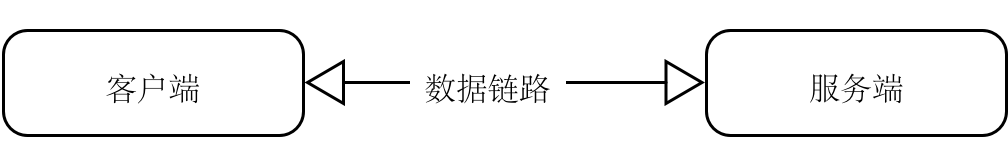
系统管理模块监控系统的运行状态以及对系统内节点进行新增、删除、重新部署等操作。

2.2**运行环境**

运行本系统的服务端程序的计算机需要有对外的网络服务能力，同时需要具备图像处理单元（GPU）。为使系统能够正常执行系统的所有功能模块，运行本系统的计算机中还需要安装以下支持软件：

* 操作系统：Ubuntu 16.04 LTS 64bit
* 外部依赖软件：
  + Java SDK —— Java 运行库
  + Redis ——内存数据库
  + OpenCV —— 计算机视觉开源库
  + Tensorflow —— 深度学习开源框架
  + Apache Kafka —— 消息队列开源开
  + Apache ZooKeeper —— 分布式协调服务

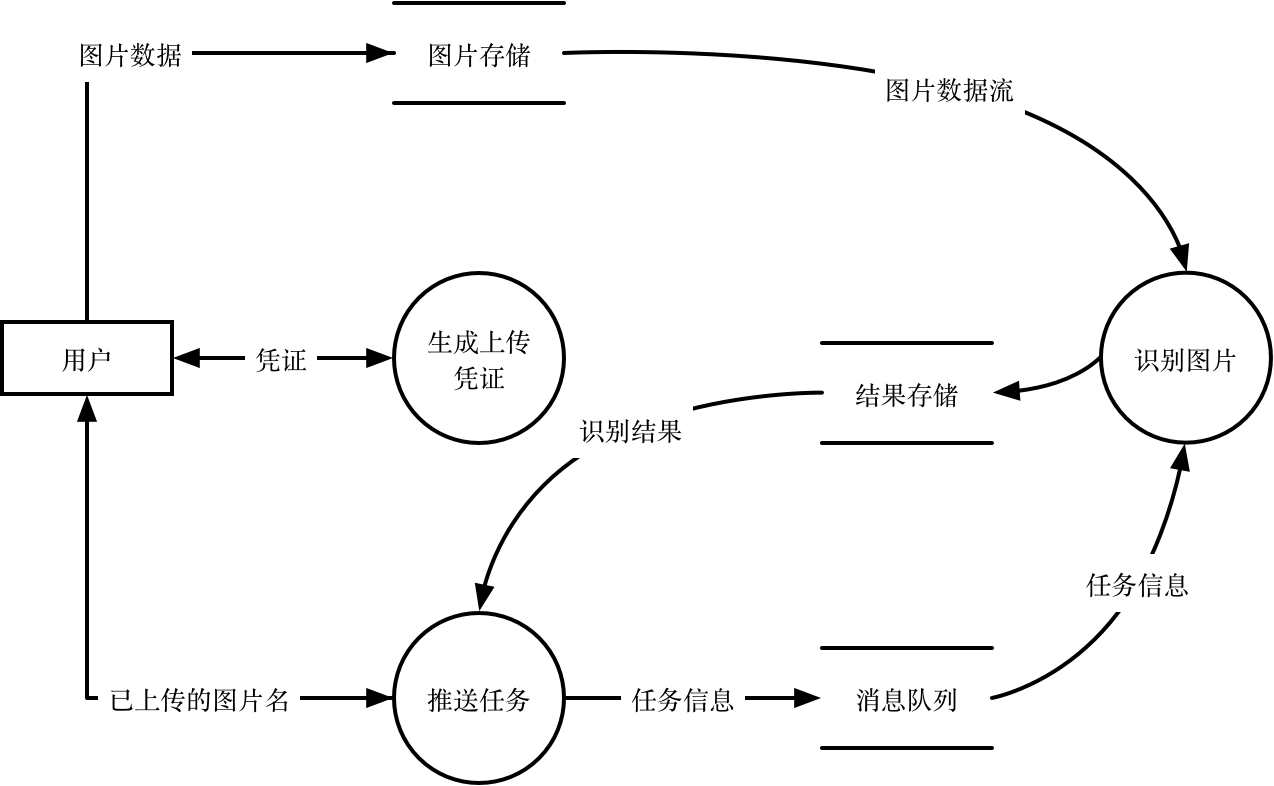
2.3**基本设计概念和处理流程**

系统对外暴露的结构为 C/S （客户端-服务端）结构，系统的外部结构图如下：

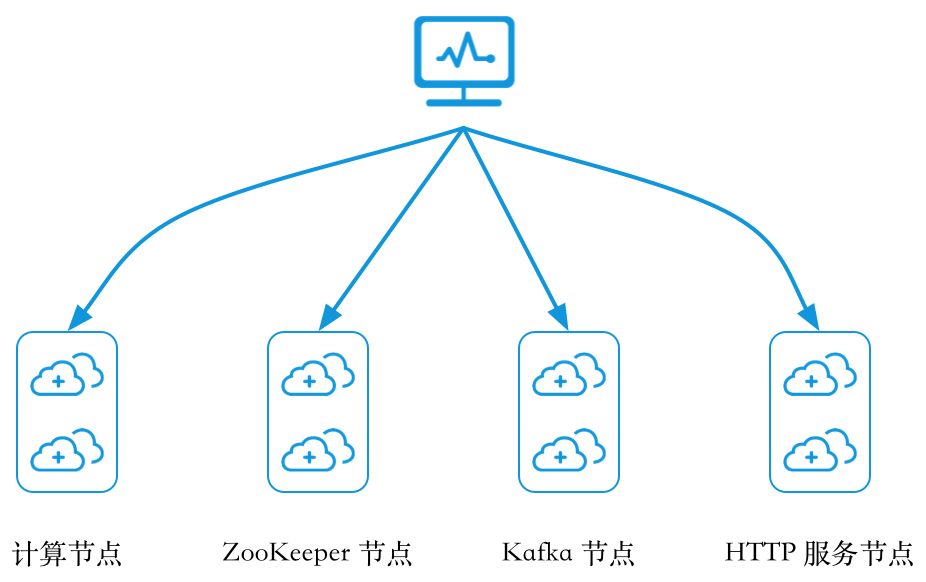
服务端部分为计算机集群，分为以下五类节点：

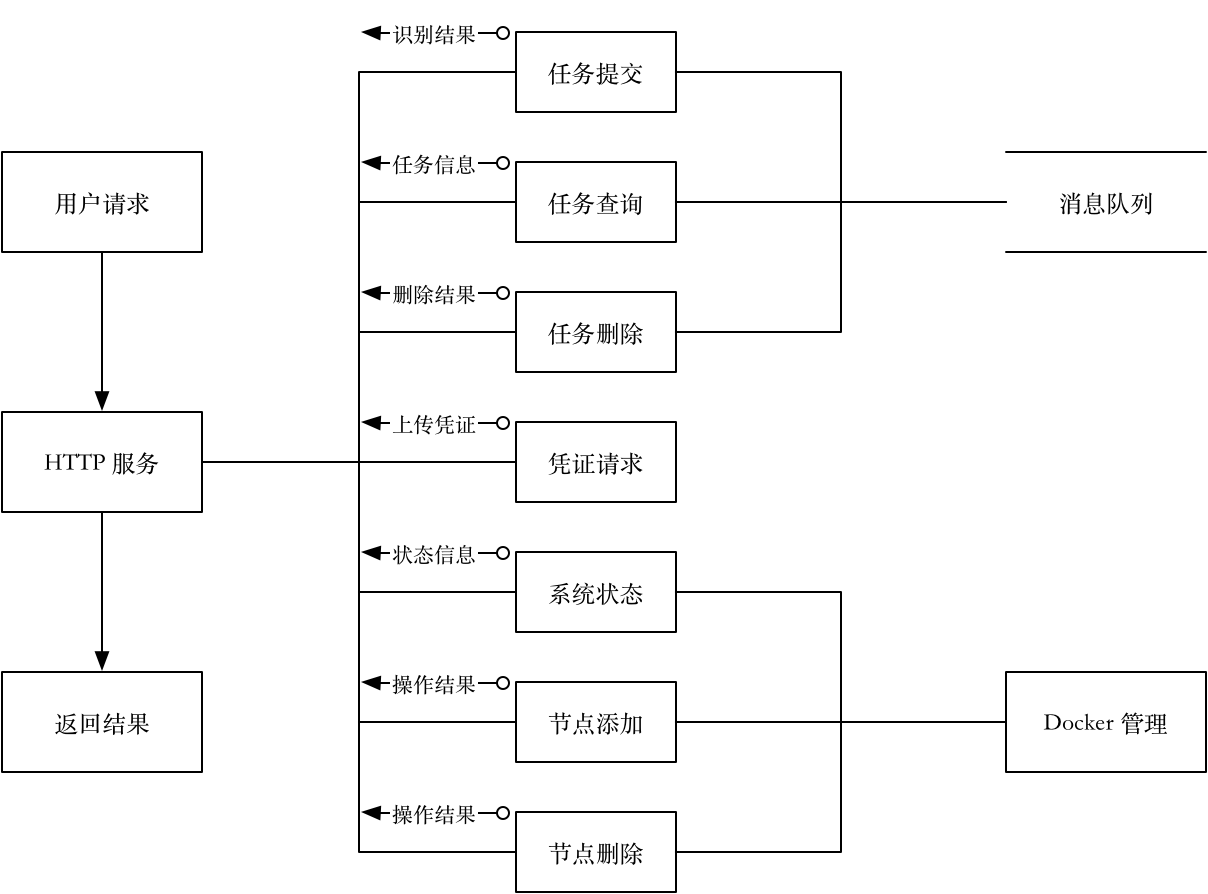
* 管理节点
* 计算节点
* ZooKeeper 服务节点
* Kafka 服务节点
* HTTP 服务节点

管理节点使用 Docker 集群管理平台对其他四类节点进行监控，其他四类节点都为一个Docker 容器实例。管理节点监控其他节点的运行情况，并对异常节点进行重启或下线操作。

系统中的顶层数据流如下：

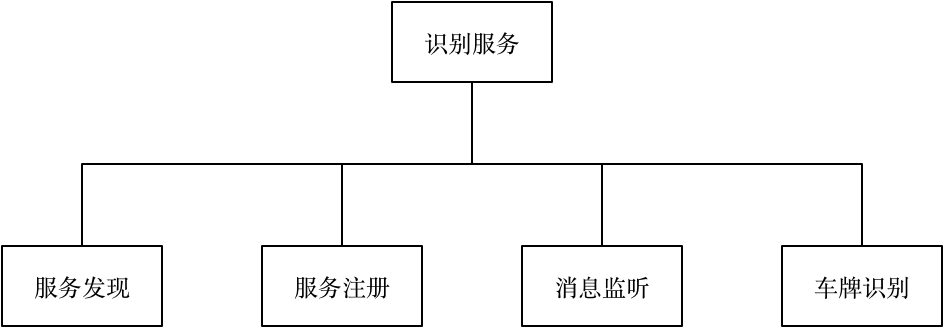
2.4**结构**

系统的总体结构如下：

其中 HTTP 负责提供整个系统的对外服务，其结构如下：

ZooKeeper 节点运行 Apache ZooKeeper 分布式协调程序，在本系统中，该节点负责服务的注册与发现功能，其保存着系统中所有运行中的节点信息、识别中的任务信息，以协调整个系统的工作。

Kafka 节点运行 Apache Kafka 消息队列程序，负责存放和分发由 HTTP 服务节点添加到队列中的任务信息。

识别节点负责从 Kafka 消息队列中取出任务进行识别，其结构如下：

2.5**功能器求与程序的关系**

系统各服务模块与需求的关系如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | HTTP 服务 | ZooKeeper 服务 | Kafka 服务 | 识别服务 |
| 识别车牌 | √ |  | √ | √ |
| 任务添加 | √ |  | √ |  |
| 任务删除 | √ | √ | √ |  |
| 任务查询 | √ |  | √ |  |
| 节点添加 | √ | √ |  |  |
| 节点删除 | √ | √ |  |  |
| 系统状态查询 | √ | √ |  |  |

2.6**人工处理过程**

系统中包含的各个节点服务器，在出现无法自行恢复的错误时，需要运维人员进行人工干预，干预过程包括：重新启动节点服务器以及服务程序、重新配置系统的节点并上线。

2.7**尚未问决的问题**

无

3**接口设计**

3.1**用户接口**

* 请求上传凭证

用户通过构造带有图片仓库信息和文件后缀参数的 HTTP URI，并向服务器发起 HTTP GET 请求来获取上传凭证，系统通过计算并返回带有上传文件的存储路径和凭证的应答信息。

请求样例如下：<http://localhost/token?bucket=plates&folder=dir_test&suffix=jpg>

返回信息样例如下：

{

"filename" "UNG0kg84cbDK0SL0fpx0.jpg",

"uploadToken":"token"

}

上传 Token 的计算规则遵循 七牛云[[1]](#footnote-1) 的开发文档。

* 提交识别任务

用户通过构造带有文件路径参数的 HTTP URI，向服务器发起 HTTP POST 请求，系统将分配任务的 UUID 并把其投递到消息队列中，然后等待处理完成后从内存数据库中获取识别结果并返回给用户。

识别结果的示例如下：

{

"task\_id": "25cd431c-89ab-4c83-9688-dddd7c735675",

"plates": [

{

"locate": [

611.269,

427.686,

609.582,

373.312,

847.931,

365.914,

849.618,

420.288

],

"chars": "蓝牌:粤BF29V1"

}

]

}

* 删除识别任务

用户利用带有任务编号的参数的 HTTP URI，向服务器发送 HTTP DELETE 请求来删除识别任务，通过返回的 HTTP 状态码标识操作的结果。

* 查询任务信息

用户利用带有任务编号的参数的 HTTP URI，向服务器发送 HTTP GET 请求来查询识别任务。

* 添加系统节点

用户通过 Docker 容器管理界面，向系统中添加指定类型的节点。

* 删除系统节点

用户通过 Docker 容器管理界面，将系统中指定类型的节点删除。

* 查询系统信息

用户通过 Docker 容器管理界面，查询运行中的系统各节点信息。

3.2**外部接口**

系统系统使用了 七牛云 提供的数据存储服务，需要向七牛云服务器请求图片数据。

3.3**内部接口**

系统各个模块为独立的服务，通过使用消息队列、内存数据库 Redis 完成接口的解耦，系统各个模块遵循预先约定的数据格式，并通过 Kafka、Redis 的数据访问协议，来完成各个模块的耦合。

4**运行设计**

4.1**运行模块组合**

本程序以数据的处理阶段划分模块，一个模块完成整个处理流程中的特定功能，面向用户的 HTTP 服务接口根据用户输入，调用后台中的不同模块来完成整个处理过程，各个模块之间通过传递 JSON 格式的消息来实现模块之间的合作和数据共享。

4.2**运行控制**

程序的控制方式为通过 URI 访问对应的 HTTP 服务接口，只要符合用户接口的通信协议和数据格式规范，不对用户的操作方式作出任何限制。

4.3**运行时间**

程序的运行耗时为两类耗时的总和：网络延时 + 处理耗时，每个任务的总运行时间的上限设置为 1 秒，处理超时系统将直接返回超时出错信息。

5**系统数据结构设计**

5.1**逻辑结构设计要点**

* 任务结果（任务编号，车牌信息数组）
* 车牌信息（顶点坐标，车牌字符）
* 出错信息（错误代码，错误信息）

5.2**物理结构设计要点**

* 任务结果

|  |  |
| --- | --- |
| 属性名 | 数据类型 |
| 任务编号 | UUID |
| 车牌信息数组 | VECTOR |

任务结果存储在 Redis 内存数据库中，并定时写到数据库。

* 车牌信息

|  |  |
| --- | --- |
| 属性名 | 数据类型 |
| 顶点坐标 | ARRAY |
| 车牌字符 | STRING |

* 出错信息

|  |  |
| --- | --- |
| 属性名 | 数据类型 |
| 错误代码 | INTEGER |
| 错误信息 | STRING |

* 任务结果表

|  |  |  |
| --- | --- | --- |
| 属性名 | 数据类型 | 备注 |
| 任务编号 | VARCHAR(36) | UUID 字符串 |
| 识别结果 | TEXT | JSON 文本串 |

5.3**数据结构与程序的关系**

1. 任务消息

任务消息以 JSON 序列化字符串的形式存放在 Kafka 消息队列中，通过 Kafka 接口进行访问。

1. 任务结果

任务结果以 JSON 序列化字符串的形式存放在 Redis 内存数据库中，通过 Redis 访问接口进行访问。

1. 出错信息

出错信息分为两类：任务出错信息与任务结果一同存放在 Redis 内存数据库中，通过 Redis 访问接口进行访问；其它出错信息直接返回给用户，不作存储。

6**系统出错处理设计**

6.1**出错信息**

|  |  |  |
| --- | --- | --- |
| 错误类型 | 错误含义 | 错误处理 |
| 识别错误 | 无法识别到车牌 | 返回错误结果 |
| 任务超时 | 任务处理时间过长 | 返回超时信息 |
| 节点异常 | 节点应用故障或网络故障 | 节点重启 |
| 输入无效 | 用户的输入非法 | 返回错误信息 |
| 程序运行出错 | 程序运行时发生异常 | 尝试重新运行 |

6.2**补救措施**

本系统的数据均存在多个副本，当数据存储节点发生异常时，可以从其他数据副本中进行数据复制和恢复。

当系统服务节点出现异常时，系统的其他节点将接管系统的处理任务，不会对业务处理产生影响，当系统的处理节点不足，可能会发生大规模的任务超时，此时应该尽快向系统中新增处理节点。系统的处理节点发生时，Docker 集群管理程序会尝试重启应用。

当以上的机制都失效时，需要人工干预来对系统进行恢复，人工干预的措施为排查系统的服务节点网络连接、重新配置 Docker 容器服务使应用重新上线。

6.3**系统维护设计**

为了便于系统的维护，系统的设计过程中使用了 Docker 搭建微服务集群的设计，并使用 Docker 集群管理的方式进行维护和管理。对系统中任何模块作出的变更，只要符合模块之间的通信协议，就不会影响到整个系统的运行。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 维护项目 | HTTP 服务 | Kafka 服务 | ZooKeeper 服务 | Docker 管理器 |
| 添加节点 | √ |  | √ |  |
| 删除节点 | √ |  | √ |  |
| 系统状态查询 | √ |  | √ | √ |
| 数据备份 |  | √ |  | √ |
| 系统更新 |  |  | √ | √ |

1. https://www.qiniu.com [↑](#footnote-ref-1)