内部

中国科学技术大学

工程硕士学位论文

基于云平台的人脸识别服务的设计与实现

作者姓名：何堃

学科专业：软件工程

校内导师：叶勇 副教授

企业导师：王超 高级工程师

完成时间：二〇一六年九月十九日

Limited

University of Science and Technology of China

A dissertation for master's degree of engineering

Design and Implementation of Face Identification Service Based on Cloud Platform

Author's Name：	Kun He
Speciality：	Software Engineering
Supervisor：	A.P. Yong Ye
Advisor：	SN ENGR. Chao Wang
Finished Time:	Sep. 19th, 2016

中国科学技术大学

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____　　签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开　□保密（____年）

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

摘　要

目前，人们对人脸识别领域研究的兴趣与日俱增，比如在公共安全中，需要通过身份验证技术获得物理和逻辑访问的相关权限；在多媒体数据管理和数字娱乐中也需要用到人脸分析和建模技术。由于人脸识别算法的实现需要使用大量的计算资源，这导致开发人员在本地实现算法时需要考虑硬件和软件的各种限制。

本文是以 Microsoft Azure 平台为基础，研究如何使用 Azure 平台提供的各种基础服务来设计与实现人脸识别云服务平台。在构建云服务平台的过程中，本文主要完成了以下工作：1）通过 Azure 平台提供的云存储服务来存储和管理与人脸识别相关的数据信息，分别使用 Azure 的 Blob 存储和 Table 存储来管理不同类型的数据；2）设计与实现了基于 Azure 平台的人脸识别 API，主要包括人脸识别 API、分组个人 API 和人脸分组 API；3）通过实验论证在 Azure 平台上进行大规模人脸识别的可行性，设计与实现了大规模人脸识别 API，从而可以快速高效进行大规模人脸识别；4）为开发人员提供了多种算法云服务接口，满足其不同的需求。

本文实现的人脸识别云服务平台，允许开发人员通过使用少量的网络资源换取复杂的人脸识别算法的实现。该云服务平台的实现，解决了以下问题：1）对于低配置的终端而言，解决了人脸识别算法运行的昂贵成本问题；2）消除了人脸识别算法在不同系统平台上实现的差异性；3）降低了开发人员在其本地应用上实现大规模人脸识别算法的难度。

关键词：人脸识别，云平台，云服务，大规模人脸识别

Abstract

There is now growing interest in the field of face identification, such as public safety, the need to obtain verification techniques of physical and logical access permissions by identity; as well as to use facial analysis and modeling techniques in data management and digital multimedia entertainment. Face identification algorithms need a lot of computing resources, while developers in the local usage need to take into account the limitations of hardware and software.

This paper studied base services of Microsoft Azure platform to design and implement a face identification cloud service platform. In order to build the platform, this paper mainly completed the following work: a) Through Azure cloud storage service store and manage data, use Azure Blob Storage and Table Storage to manage different types of data; b) Design and implement face identification API，include Face Identification API, Person API and Person Group API; c) Demonstrate the feasibility of large-scale face identification performed on Azure, design and implement the large-scale face identification API, so that people can quickly and efficiently carry out large-scale face recognition; d) Provide a variety of algorithms cloud service interfaces to meet developers' different needs.

This paper realized the face identification cloud services platform, allowed developers to exchange complex face identification algorithm by using a small amount of network resources. By providing the platform following issues can be addressed: a) For low profile terminal, the expensive cost of face identification algorithm running is solved; b) The differences of the algorithm implementation on different system platforms are eliminated; c) Reducing the difficulty developers to implement large-scale face identification algorithm in their local applications.

Keywords: Face Identification, Cloud Platform, Cloud Service, Large-scale Face Identification

目 录

第 1 章 绪论

1.1 研究背景

人脸识别发生在我们日常生活的许多场景中。相比其他生物体识别方式，人脸识别除了是自然的和非侵入式的，最重要的优势在于，人脸可以在一定距离外，并且以一种隐蔽的方式被捕获。在由 Hietmeyer 考虑的六大生物特征属性中，脸部特征在机读旅行证件系统中的得到最高的兼容性得分，这基于参与、更新、机器的要求以及公众的看法等因素。人脸识别，作为主要的生物识别技术之一，由于现如今图像捕获设备技术的加速进步和网络上存在的大量的脸部图像，已经变得越来越重要，同时在安全性上也有了更高的要求。

Takeo Kanade 于 1973 年首次开发出人脸识别系统。随后人脸识别技术的发展经历了一定的蛰伏期，直到 Sirovich 和 Kirby 在低维人脸表示领域的研究有了突破，人脸识别技术的研究再次振兴。对于人脸识别领域，其他重要的里程碑包括：Fisherface 方法，其中使用的线性判别分析（LDA）方法，是继 PCA 之后进一步实现更高精度的方法；使用本地的过滤器，比如 Gabor jets 可以提供更有效的面部特征；基于级联分类架构的 AdaBoost 方法等[1]。自从特征脸方法被提出后，人脸识别技术得到了飞速发展。在受限的情景下，比如在照明、造型、站姿等可控的情形下，人脸识别在性能上可超越人类识别，尤其在大规模识别领域[2]。

通常人脸识别的需求是只需找到最相似的脸即可，比如在监控录像中检查或识别人脸，往往是要求找到最相似的面孔。通过指定特定的置信水平阈值，选取所有相似度高于该阈值的脸。前面提到过，人脸识别系统的性能在很大程度上取决于多种因素，如光照、面部姿态、表情、年龄跨度、面部磨损和表情等[3]。人脸识别的应用可以分为两类：1）合作用户场景；2）非合作用户场景。合作用户场景主要是指用户愿意通过合适的合作方式展示自己的脸从而被授予访问或特权，比如“刷脸”。而非合作用户场景，则是在用户没有意识到的情况下进行人脸识别，以监控应用为代表。对于合作用户场景，比如访问控制而言，近距离人脸识别是最困难的问题，而对于非合作用户场景，比如监控识别而言，远距离人脸识别则是最具有挑战性的。

目前，云计算[4]已经成为了一个遍布世界各地的时髦词。参照美国国家标准与技术研究院（NIST）的定义，云计算是一种计算模式，可以按照用户需求方便的提供给用户网络访问，同时提供资源共享池并允许对其进行配置，池中的资源有网络、应用程序、服务等。这些资源在云计算模式中都可以被快速供应给用户，且只需付出少量的资源管理或者与其供应商少量的交互即可[5]。众所周知，面向服务的架构倡导的是“一切皆服务”，而云计算服务供应商则是根据模式的不同提供不同的服务。为此，NIST 定义了三种服务模式，具体为：1）基础设施即服务（IaaS）；2）平台即服务（PaaS）；3）软件即服务（SaaS）。同时，NIST 在对云计算的定义中也包括了了对云计算部署模型的定义，主要有公有云、私有云、社群云和混合云[6]。比如，公有云的服务是以现收现付的方式提供给广大用户，被出售的服务是效用计算；而当云服务被提供给公司内部或其他组织而并不是面向一般公众时，我们称之为私有云。云计算为我们提供了很多便利。比如，在做一些对动画、图形有高要求的工作时，我们可以使用云计算而不再需要高性能配置的个人计算机；我们可以任何时候在任何地方通过网络从“云”访问到我们的数据，而无须担心存储崩溃和数据丢失；如果云计算模型具有网格计算的后端，我们还可以用整个系统的超级计算机来研究或分析大型的复杂的数据。

Microsoft Azure（原名 Windows Azure）是微软云计算服务的总体品牌名称。2008 年微软在其专业开发者大会上对外首次发布，并于 2010 年被正式投入到商业应用中[7]。Azure 是微软提供的云计算基础平台，它提供给用户一系列云服务集成，包括云平台上的数据存储、网络、云计算和各种云上的应用程序等。同时，开发者可以选择在 Azure 上搭建各种云应用并将其与自己的工具集成。除此之外，开发者还可以利用微软开放的架构，开发自己的与计算机、内部服务器或其他工具相连的各种应用[8]。Microsoft Azure 使用了云计算中的服务模式，从而为用户在使用云计算在线服务时提供了必须的操作系统和数据存储与基础管理平台。它既能够作为 IaaS 服务方式，提供各种操作系统，各种大小的虚拟机，并可以快速实现虚拟机的备份和转移等；同时也可以作为 PaaS 服务方式，被应用程序所使用，从而可以帮助应用程序实现比如扩展资源、监控资源等服务操作[9]。Microsoft Azure 目前的基础服务组件如图 1-1 所示。

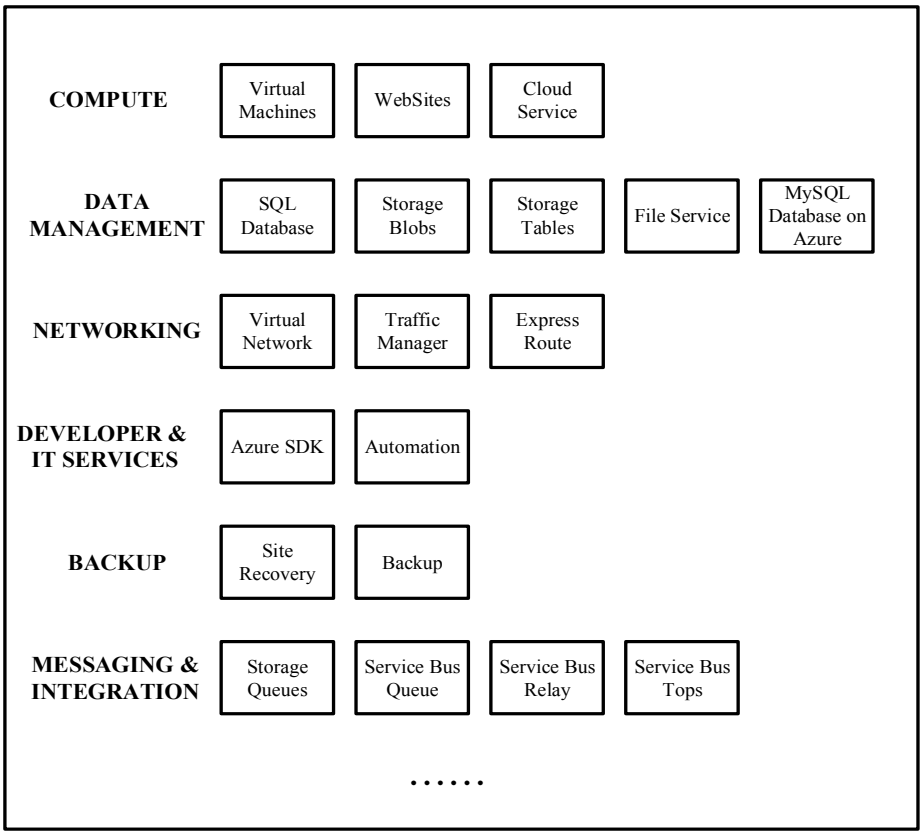

图 1-1 Azure 基础服务组件图

计算模型服务，支持应用程序的执行，是云平台能提供的最基本的功能。为此，Azure 提供了三种计算模型：1）Azure 虚拟机：主要作用是让用户能完全控制云中的虚拟机的实例，允许用户根据自身的需求来创建虚拟机，仅在其运行时按分钟进行收费。2）Web 应用：指的应用程序是在 Azure 平台上运行网站，且无需对 Web 服务器进行管理。用户可以将现有应用程序完整地迁移到 Azure Web 应用上，或选择直接在 Azure 上新建网站。3）Azure 云服务：Azure 云服务所提供的技术可明确用来支持用户的那些可靠的、可扩展的以及管理任务不多的应用程序。

数据管理服务，应用程序是离不开数据的，并且需要处理各种不同的数据类型。对于数据管理，Azure 提供了几种存储结构。1）SQL 数据库：Azure 平台的 SQL 数据库用来进行关系存储。它不仅提供 DBMS 的所有核心功能，同时在云平台上还是个 PasS 服务。2）Table 存储：提供了一种平面的 NoSQL 的方式来对数据进行存储管理，它采取的是键/值存储，不支持关系存储，可以存

储各种类型属性的数据。Azure 表对类型化的数据可以提供快速的访问，同时也可以高度缩放。3）Blob 存储：Blob 在 Azure 中主要的存储数据是非结构化的二进制数据，最大为 1TB。Blob 在应用程序使用 Azure 驱动器的情形下，可以为具体实例提供文件系统持久存储。

开发人员服务，目前 Azure 平台为开发人员提供了许多工具和服务，方便他们创建和维护 Azure 上的应用程序，主要包括 SDK 和自动化。微软为.NET、Java、PHP 等提供了对应的 SDK，并提供了一个面向任意语言的基本 SDK。Azure 自动化则指的是，可以在 Azure 上自动构建、管理和部署相关资源的方法。

除了以上介绍的 Azure 平台服务，Microsoft Azure 还提供了许多其他的服务，比如网络服务、备份服务、消息传送和集成服务等。这些服务极大地方便了用户在 Azure 上进行开发，具体可以参考 Azure 官方网站的详细介绍。

本文通过研究 Microsoft Azure 云平台的服务用于实现人脸识别的云服务平台的构建。在具体实现过程中主要用到了 Websites、Cloud Services、Cloud Compute、Storage Blobs、Storage Tables 等服务。

1.2 研究现状分析

现如今很多国家都在研究人脸识别技术，以美国、日本及部分欧洲国家为代表，其中不乏一些知名的研究机构，比如麻省理工大学媒体实验室、卡耐基梅陇大学人机交互研究所、微软研究院和剑桥大学工程学院等。上世纪 90 年代，高性能计算机的发明，促进了人脸识别技术研究领域的飞速发展，正式拉开了自动化人脸识别研究的帷幕[10]。在此期间，一些国外大学机构也取得了很多成果，他们的研究领域很广，有从心理学和感知学角度研究的，有从视觉机理角度进行研究的，而更多的则是研究如何对输入的图像进行计算机自动化人脸识别 [11]。目前，国际上在人脸识别技术研究领域主要形成了以下这些方法：1）几何特征识别法；2）模板匹配识别法；3）K-L 变换特征脸识别法；4）隐马尔可夫模型识别法；5）神经网络识别法等 [12]。

我国开始研究自动人脸识别技术要追溯到上世纪 80 年代，以清华大学、中国科学技术大学、中科院等研究机构为代表。目前，国内的人脸识别研究主要是集中在以下三个领域：1）几何特征下的人脸正面识别法；2）代数特征下的人脸正面识别法；3）连接机制下的人脸正面识别法[13]。这其中出现了一部分具

有代表性的人物：张辉，何振亚等采用对称主元分析神经网络，比较好地实现了大规模人脸数据的存储和快速识别功能；周激流的研究实现了具有反馈机制的正面人脸识别系统；彭辉、张长水等通过优化特征脸方法降低了算法的运算量[14]。

云计算是当今计算机领域的热点。它的出现，向世界宣告了低成本提供超级计算时代的到来，它将改变人们获取信息、分享内容和互相沟通的方式。许多公司很早就开始了云计算领域的研究，国际上比较著名的有 Amazon、Google、Microsoft 等，而国内比较有名的有百度、阿里、华为等，它们各自都研发了自己的云计算产品。其中，Microsoft 于 2008 年 10 月推出 Windows Azure 操作系统[15]。作为微软的又一次重大的转型，Azure 通过在互联网架构上打造新的云计算平台，其底层是微软全球基础服务，由遍布全球的第四代数据中心构成。

微软在其 2015 年 Build 大会上，发布了 Project Oxford（现更名为微软认知服务），主要是为 Microsoft Azure 的使用者提供了免费的视觉和语音识别等的 API 和 SDK。微软研究院主导开发了这一系列的服务，它所提供给开发者的模型都是通过机器学习和深度学习在微软现有的其他产品的基础上通过技术训练得到的。本论文所研究的人脸识别 API 就是其视觉识别 API 中的一种。微软的人脸识别 API 是以 Azure 平台为基础，使用了其最先进的人脸算法，为开发者在应用程序开发过程中提供了算法技术支持。通过 API 的使用，可以检测和识别出图像中的人脸。目前人脸 API 主要有以下使用场景，通过检测图像中的人脸，用方框标出被检测到的人脸并且用像素点明确标注人脸在图片中的位置，可以识别出人脸属性比如年龄、性别、脸部特征点等。使用者通过提供图片 URL 或上传图片方式即可进行人脸检测，在一张图片中最多可检测到 64 张人脸[16]。

本文所研究的基于 Microsoft Azure 云平台的人脸识别算法即云服务平台，其目的是将现有的人脸识别算法涉及的相关资源进行整合，包括计算资源、数据资源等，通过云服务平台的方式提供给用户使用[17]。本质上是通过云服务的方式将人脸识别算法对用户开放，通过网络提供给用户算法的接口调用。从而用户只需要根据 API 调用人脸识别的算法接口，就可以在自己的应用程序上实现云服务平台上的人脸识别算法，此外用户还可以根据自己的需求定制所需的算法应用。在此过程中，用户无需关心具体的算法实现、硬件性能、平台搭建等问题。通过将人脸识别算法与云平台结合提供服务，可以推广算法在产品中的应用效果和范畴，使其更具有实际意义。

1.3 论文主要工作

本文主要工作是基于 Microsoft Azure 平台设计与实现人脸识别的云服务平台。为实现这一目标，主要完成了以下工作：

1) 使用 Azure 平台存储和管理数据：任何平台服务都是离不开数据的。如何在数据不断增长的情况下保证数据一致性并实现有效的数据扩展，是本文在 Azure 平台数据存储管理中需要考虑的重要的问题。尤其是将人脸分组的量级从 1000 扩展到百万量级，更加增加了数据有效存储管理的难度。为了解决这些问题，我们在 Azure 平台上使用 Table 和 Blob 存储服务，分别用来负责存储和管理结构化的数据信息和二进制文件的数据信息。

2) 设计与实现基于 Azure 平台的人脸识别 API：主要是将人脸识别算法的实现以接口的方式提供给用户调用，用户可以通过 HTTP 协议来访问所需的接口。用户只需要付出少量的网络流量即可换取复杂的云平台人脸识别算法服务，而且在任何时间任何地点通过网络都可以使用云平台的服务。本论文中主要设计与实现了以下 API：

a) 人脸识别 API：该 API 主要用于从用户提供的一组人脸分组中，根据相似人脸置信度排序返回与查询人脸相匹配的人，用户在调用 API 时可以指定返回结果数量。

b) 分组个人 API：主要用来在用户指定的特定的人脸分组中管理其中的分组个人，并且管理隶属于这个人的所有的脸。

c) 人脸分组 API：由用户创建并管理指定的一组人，人脸分组对人脸识别 API 来说是重要的数据。识别所参考的人的脸来源于用户指定的分组。此外，人脸分组 API 还提供了训练分组接口，通常人脸分组需要在被训练然后才被应用于识别过程。

3) 人脸识别 API 的改进和优化：目前支持的人脸识别算法 API 对于人脸分组中分组个人数量上限制是 1000。本文通过实验论证在云服务平台上进行大规模人脸识别的可行性，在之前的 API 基础上进行改进和优化，实现了了大规模人脸识别 API。主要是基于 Azure 平台，通过建立人脸索引，在确保识别正确率与之前算法比不受太大的影响下，考虑响应时间和本地下载等问题，选择最合适的参数来实现百万量级的人脸识别算法。通过重新设计和实现之前算法

中的特征脸合成方法，省去在人脸分组 API 中提到的训练分组方法，从而简化了目前人脸识别 API 服务的步骤，优化了接口请求时间，提高了识别效率

4) 提供多种算法云服务接口：人脸识别算法的实现是需要大量消耗计算资源的，考虑到平台、硬件性能等因素的限制，用户想要在本地直接实现人脸识别算法需要投入大量的资源。然而通过云平台提供算法服务，用户只需要通过访问接口，仅用较少的网络流量即可获取多种复杂的算法服务，并且可以任何时间在互联网的任何地点都能实现所需算法，使它为自己的应用程序服务。

1.4 论文组织结构

本文分七章撰写，论文组织结构安排如下：

第一章为绪论，主要是对本文研究的基于 Azure 云平台的人脸识别云服务平台的研究背景、现状和本文主要完成的工作进行介绍。

第二章为相关原理与技术的介绍。首先，简要介绍了人脸识别技术。然后，针对 Microsoft Azure 云平台进行研究分析，主要介绍 Azure 上的相关架构和服务，并分析人脸识别算法服务在该平台上如何构建。此外，还介绍了 RESTful API 技术框架的使用。

第三章为云服务平台需求分析与概要设计。对本文实现的云服务平台分别从功能上和性能上进行需求分析，再依据需求分析的结果对整个云服务平台进行概要设计。

第四章为云服务平台详细设计与实现，将云服务平台的提供的服务分为三层：数据管理层、算法计算层和 API 接口设计层，分别介绍了各层服务的详细设计与实现。

第五章为人脸识别 API 的设计、实现与改进。详细介绍了云服务平台人脸识别 API 的设计与实现，并对之前的 API 进行改进和优化，实现了大规模人脸识别算法的云服务。

第六章为云服务平台的功能测试与性能分析。首先，介绍了平台的功能测试的方法并展示了在多平台应用中的实现效果。然后，分别分析了云服务平台的参数性能和服务性能。

第七章为总结和展望，总结本文完成的研究工作，并对将来进一步的研究工作提出展望。

第 2 章 相关原理与技术介绍

2.1 引言

本文项目目标是实现基于 Microsoft Azure 平台的人脸识别云服务平台，使用了 Azure 平台提供的云服务、云计算和云存储等功能。其中云计算技术的运用，向用户提供了一种随机化的网络存取服务，允许用户使用并自定义运算资源，包括服务、存储和网络等，并能通过使用少量的管理或与服务供应商的交互来得到资源的供应与释放[18]。除此之外，使用 Azure 平台提供的服务也带来了许多好处，提高了企业级应用的灵活度，优化了资源分配，并且使管理成本支出最优化。

2.2 人脸识别技术

本文实现的云服务平台是以人脸识别算法作为基础，目的在于解决该算法计算量大、数据存储需求高，从而导致算法被实际硬件性能限制影响的问题。在此基础上，提出通过将人脸识别算法以算法即服务的方式提供给用户使用。

人脸识别属于视觉模式识别的问题，一张脸可以表示为一个三维物体，当它在受到姿势、面部表情、外界光照变化等因素影响时，需要依据所获得的图像来进行人脸识别。而目前大多数应用使用到的是二维脸部图像，另外一些应用由于使用三维（深度或范围）的图像或超越视觉频谱的光学图像则需要更高的安全级别。人脸识别系统通常可以分为四个模块，如图 2-1 所示。

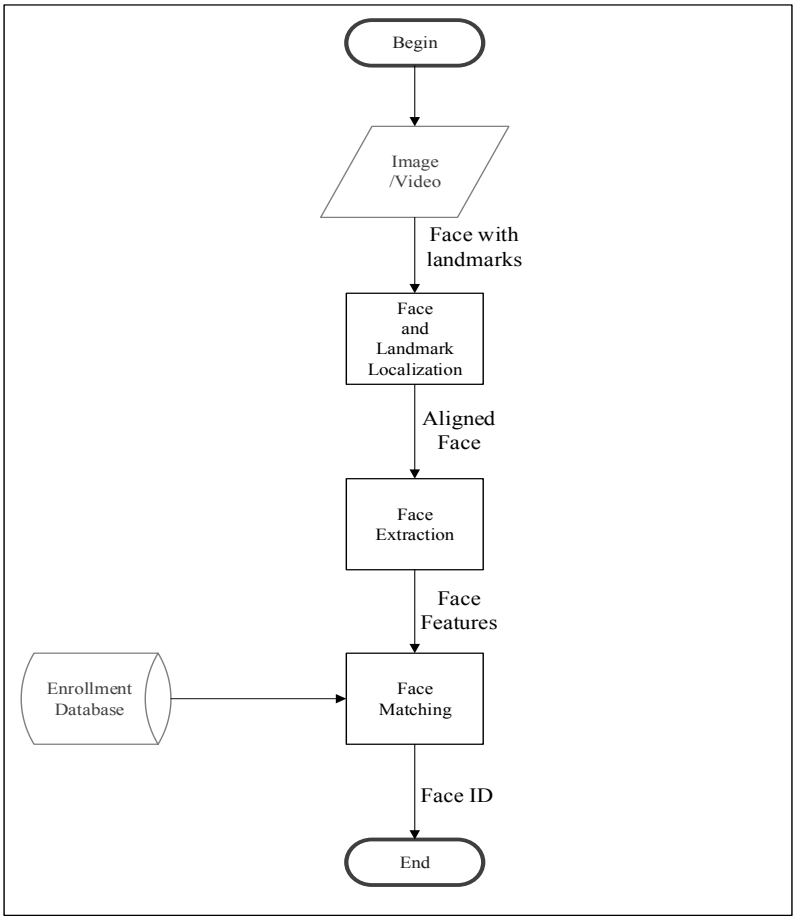

图 2-1 人脸识别处理流程图

1) 面部界标本地化。人脸的检测可以将人脸从背景中分离出来。在视频的情况下，被检测到的人脸需要使用脸部跟踪组件从而可以使其在跨越多个帧的时候被跟踪到。使用人脸检测，可以划分出人脸的区域和定位，实现面部界标本地化。

2) 面部标准化。通过脸部的几何级数和光度来实现标准化。其中的几何归一化处理可以通过脸部剪裁把脸转换成标准帧，扭曲和变形可以被用于更复杂的几何标准化。而测光归一处理，则可以在某些属性上标准化人脸，比如照明、灰度级等。

3) 面部特征提取。面部特征提取是在规范化的人脸上通过提取可以用来区分不同的人的不同的面孔的突出的特征信息，并且这些特征相对于几何和光度变化是具有鲁棒性的。这些被提取的面部特征是为了进行之后的人脸匹配工作。

4) 人脸匹配。用被检测出的人脸的特征与已存在的一张或多张脸进行匹配。对于一对一的情形而言，结果为“是”或者“否”即可；而对于一对多的情形，结果则是找到有足够置信度的最匹配的人脸，或是低于一定阈值未知的人脸。对于人脸识别系统而言，在人脸匹配阶段最主要的挑战是对于被用来比较的面部特征能否找到合适的相似指标。

2.3 Azure 的基础设施即服务

Microsoft Azure 是微软推出的公有云服务（Public Cloud Service）平台，属于微软线上服务的一部分，自 2008 年 2 月发布以来，到如今已有八年，而从 2010 年开始运营到现在（2016）也有六年时间了。从最开始的最基本的三个服务，已经扩展到如今的超过三十多种的服务。大致可以划分为计算、存储、应用程序服务、数据、备份、网络等。NIST 定义了基础设施即服务（IaaS）、PaaS 平台即服务（PaaS）和软件即服务（SaaS）三种模式，分别是以基础架构、平台和软件进行区分。在这三种服务的基础上还衍生发展出运算即服务、管理即服务、存储即服务等其他模式，将它们统称为 XaaS[19]。

IaaS 是云计算的最底层，它是由大量的服务器、存储、网络架构和其他硬件组成的，为云计算提供了硬件基础，提供了云平台计算的能力[20]。目前，Microsoft Azure 在全球拥有 22 个资料中心和 32 个内容传播网络 POP，2015 年被 Gartmer 列为云计算的领先者。IasS 基本上是提拱了不同的云平台的硬件和计算资源方案给使用者，Azure 则以各种操作系统的虚拟机的方案提供给用户选择。图 2-2 是 Azure 的基础设施即服务的技术架构图，描绘了 IaaS 服务的平台的配置、迁移和管理的流程。

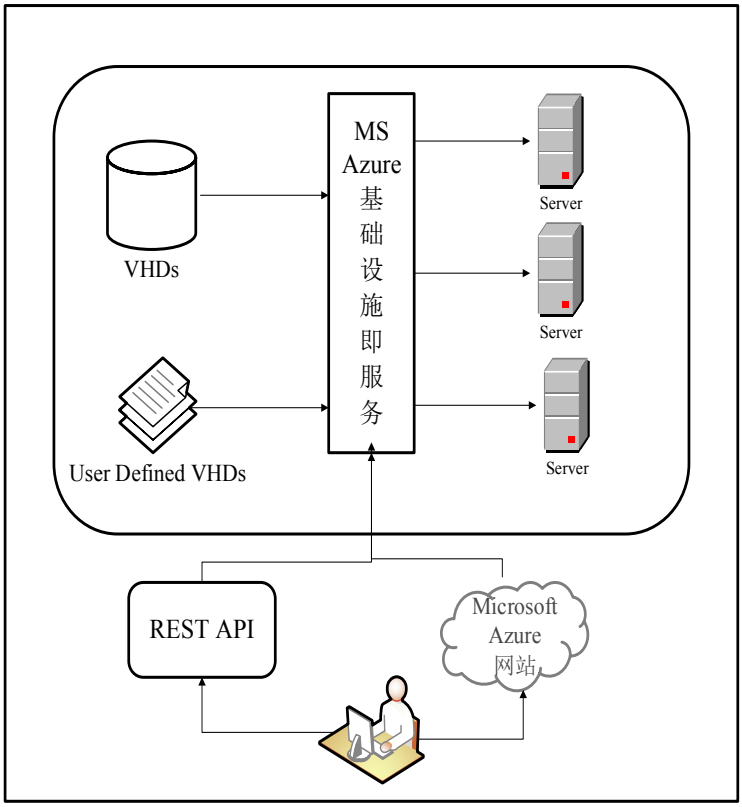

图 2-2 Azure 基础架构即服务模式图

1) 配置：Microsoft Azure 平台对于虚拟机的管理可以采用以下两种方式：

a) Azure 官方网站：使用者可以通过登陆 Azure 官方网站，使用网站提供的图形化界面直接对 VM 进行选择配置。

b) REST API：使用者也可以通过调用 REST API 运行 Microsoft Azure Shell 来实现基础配置。

2) 管理：用户通过 Microsoft Azure 官方网站来完成 Azure 平台上的基础设施的管理，根据网站提供的管理界面，可以方便快捷的完成对基础设施的订阅、配置和管理操作。

3) 迁移：Azure 平台可以通过快速捕获 VM 的方式生成相应的映像文件（VHD），从而可以实现在所有支持 VHD 类型的文件的云计算数据平台上的快速迁移。VHD 文件可以用在 IaaS 中生成一台与之前的 VM 功能相同的新的 VM 提供给用户使用。此外，通过 Azure 云平台提供的存储服务也能共实现虚拟机上数据的持久化存储。

上述的这些 Azure 平台在 IaaS 上提供的功能，能够帮助 Azure 根据用户具体使用情况的变化来调节各种计算资源，从而更高效的为用户服务，同时也可以通过 Azure 的内部监控服务按用户的实际使用量来收取费用。这体现了 IaaS 的一个重要功能，动态扩充。动态扩充主要是指当使用者的应用程序在某些时间段出现较大的流量使用时，IaaS 架构要能够自动的扩张，保证服务在高承载的情况下顺利运行，而当流量降低时，要能够自动恢复在平时正常的使用配置即可 [21]。由于以上的功能需求，Azure 允许用户将自定义的 Windows 或 Linux 操作系统的虚拟机部署到实际的生产环境中，也可以直接使用库中提供的预先配置好的映像。在允许扩展的同时，Azure 的收费是按需改变的，使用者可以根据自己的实际需求来改变在 Azure 上的资源的使用，同时根据 Azure 监控服务提供的结果，按实际的功能使用量和具体使用时间付费即可。

之前提到过硬件是云平台的运算资源，但是仅仅有硬件是不够的，还需要有足够的软件才可以运行。除了软硬件之外，数据也是平台中必不可少的组成部分，应用程序离不开数据。为此，Azure 平台为用户提供了 SQL Database、Blob 存储以及 Table 存储等服务，来方便用户构建满足其数据需要的合理的解决方案。在此基础上，Azure 平台也为用户实现了相应的数据服务的统一性、灵活性以及可扩展性等。Azure 平台上的数据存储方式都是以服务的方式面向用户的，用户可以在云平台上借助这些服务创建应用程序。Azure 的数据存储服务也保证了用户的数据安全性、一致性和快速访问[22]。

HPC（High-Performance Computing），即高性能计算，也就是我们过去常说的超级计算。微软通过扩展其 Azure 平台的突发计算能力和大群集的计算能力，使得用户能够通过云平台使用 HPC 级的计算[23]。Azure 平台对用户是按需提供计算资源的，从而允许用户可以在云平台上运行大量的并行作业或批处理作业。在需要更多的计算资源时，可以将用户本地的 HPC 群集扩展到云平台中，或者完全在云平台上进行运行计算[24]。Azure 平台提供 Azure Batch 服务给用户，即以平台服务的形式自动缩放计算资源并且提供给用户作业计划程序，从而方便用户在 Azure 中实现并行的大规模的应用程序。主要是通过在多个 VM 上并行作业的方式来实现 Azure 平台上的 HPC，需要考虑到合理的调度方式，Azure 为此提供了相应的计划程序来完成跨实例分发的工作。此外，Azure 平台通过扩展并利用高级网络技术如远程直接内存访问技术，同时使用行业标准信息传递接口来运行 HPC 应用程序，以方便用户在需要的时候快速得到结果。

Azure 平台提供的 HPC 功能，可以在为用户在云平台的各种规模的应用开发带来规模的优势和灵活性。

上述对 Azure 基础设施平台的详细介绍，证明了 Microsoft Azure 平台可以为用户提供其需要的硬件设施、软件应用和数据存储的管理。该平台有能力保障用户开发的应用程序能正常运行，并能够对这些应用程序进行轻松、高效、快速的管理，平台服务是动态扩展的，对资源和时间的按需收费也能有效降低运营成本。

2.4 Azure 的平台即服务

平台服务一直是 Azure 平台的基石，也属于它最早开发的服务，从最初的 Web Role 和 Worker Role，发展到如今的 Website Role\Mobile Service 等。平台服务除了需要操作系统和必要的环境作为基础之外，还需要有应有程序开发的界面以及提供给程序开发的资源。Azure 同之前的 MFC、.NET Framework 一样也提供给开发人员相应的开发平台和工具。

本文项目目标是实现基于云平台的，保证计算资源和数据资源是可扩展的并可以被灵活分配的云服务平台，用户可以通过网络调用平台提供的人脸识别算法的 API 接口。在前一部分中，我们介绍了 Azure 提供的 IaaS 服务，它提供了应用程序运行所需的各种硬件资源、软件基础等基本的计算单元，从而保证了程序的有效运行。但是，如果直接在开发中用 VM 搭建平台创建应用程序，需要实现相关的资源管理、按需动态扩张与收缩、计算资源群组等功能，从而导致了开发难度的提升。针对这种情况，Azure 提供了平台即服务的方式来解决相应的问题。使用 PaaS 服务不需要了解基础组建的技术细节，也无需关心硬件的属性。Azure 平台对使用者隐藏了这些具体的技术细节，取而代之提供给用户的是以合理的方式管理资源的 API。用户所需要关心的主要是如何创建存储，怎样使用平台提供的 EndPoint 去有效地管理资源等。本文项目的实现需要使用到大量的计算和存储资源，并最后为开发人员提供算法的 API 接口。Azure 平台的 PaaS 为用户提供了云服务和存储服务，分别用来完成计算资源群组和对数据进行存储管理。这也是本论文主要用到的服务平台，具体可参考图 2-3。

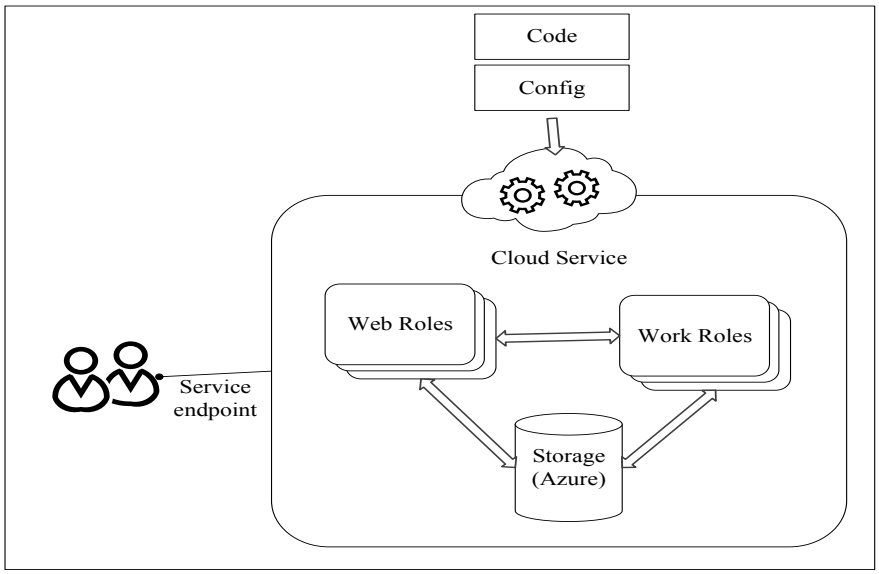

图 2-3 通过 Azure 云服务实现平台即服务图

2.4.1 运算服务

运算服务指的是在创建应用程序的虚拟机上执行的实例，它是极为重要的服务，Azure 平台上的云服务就是由它来提供的。Azure 上的云服务主要是指用户创建其应用程序并在云平台上运行该程序，与此程序相关的代码和配置。通过在 Azure 云平台上创建云服务，用户可以选择部署多层的基于 Web 的应用程序，从而可以实现应用角色的方便快捷的扩展和处理分发作业等。Azure 平台根据应用程序的不同，被分为两种基本的角色，分别是 Web Role 和 Worker Role。

Web Role 主要是用来管理以 HTTP 或 HTTPS 为主的服务，包括 Website、Web Service 或 Web API 等。它为托管的前端的应用程序提供了专门的 IIS Web Server，并将用户应用程序自动的部署到 IIS 上。

Worker Role 主要用来处理背景工作，同时它也可以利用 WCF 等使用 HTTP 的服务，由于没有 IIS，因此它可以使用的系统资源比 Web 角色要多。它主要负责的任务，是应用程序中可以独立于 IO 和交互的异步执行的任务和需要长时间运行或永久运行的任务。

Azure 云服务是由一个或多个 Web Role 和 Worker Role 组成的，每个角色都有自己管理的应用程序和配置文件。这两种角色的设置主要是为了实现云计算功能与相应的服务接口分离开来，采用松耦合的设计模式来更加有效的对应用程序进行管理。云服务在 Azure 平台上需要能够对应用程序完成日常的维护，

对操作系统的故障要能修补，对硬件的故障也要能修复。Azure 的云服务有两个环境，过渡环境和生产环境。两者的区别在于他们访问 Azure 云服务的时候使用了不同的虚拟 IP 地址，所以用户在升级的时候，可以通过直接交换访问两个部署的虚拟 IP 地址，来完成交换操作。两种环境的同时存在，能够让开发者在过渡环境中完成对应用程序的调试，稳定后再切换到生产环境中发布。这种更新迭代的过程不会导致平台服务的停止。

除了刚刚介绍的 Azure 平台云服务的两个角色，云服务还包括一些其他的内容。云服务提供监测功能，比如 Azure 的虚拟机会检测那些发生故障的物理服务器，并选择在新的服务器上重启原先的 VM。云服务中默认的配置为使用最少的监测从虚拟机的操作系统收集来详细的性能指数。同时，它提供的 Azure Diagnostic 允许用户通过 API 接口获取应用程序运行时的诊断数据。

2.4.2 存储服务

Azure 平台提供的存储服务包括关系型和非关系型两种。关系型的有 SQL Database；非关系型的主要是 Blob Storage、Table Storage、Queue Storage 等。根据开发需要，开发人员可以选择不同的存储服务来对数据进行存储。关系型的 SQL Database 是以服务的方式向用户提供的关系型数据库，它提供了不同级别的多种服务给用户，满足不同的工作负载，并且可以与 SQL Server 很好的实现兼容。而在本文项目中主要使用的是非关系型数据存储中的 Blob Storage 和 Table Storage。图 2-4 表示了 Azure 平台上非关系型存储资源的关系图。

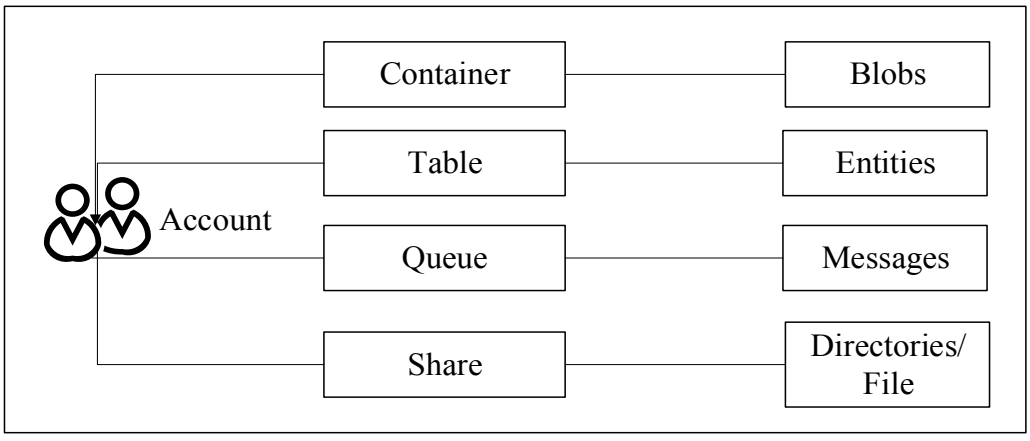

图 2-4 Azure 平台资源关系图

从上图可以看出每个账户都包括 Blob 存储、Table 存储和 Queue 存储三项服务。Blob 存储主要用于存储非结构化的文件数据，其中的 Blob 文件可以是任何类型的二进制文件或文本文件。Table 可用来存储结构化的数据，它存储的是 NoSQL 的键/属性。Queue 存储负责为云服务的各个组件之间的通信和工作流处理提供消息传送。各存储服务有功能上的差异，表 2-1 对它们进行了比较。

表 2-1 Azure 存储服务的功能表

类型	功能	界面	限制
Blob	虚拟机服务所需要的 VHD 就是保存在 Blob 内的	REST API Managed API	Block 最大为 4MB，Block Blob 最大为 200GB，Page Blob 最大为 1TB
Table	负责存储结构性的数据信息，比如表格或者索引	REST API Managed API	数据列数无限制，表格本身与数据列相关。
Queue	专门使用在需要 First-In-First-Out 的处理的应用程序中	REST API Managed API	通信数量无限制，每个信息最大为 8KB

本文实现的云平台采用了 Azure 云服务提供的应用程序集群。使用 Blob 存储非结构化的文件如图像、索引文件等；使用 Table 存储结构化的用户信息、分组个人实体信息和分组实体实体信息等；并使用消息队列在组件之间进行信息的传递。不管是上述提到的哪种类型的存储服务，都是通过调用 API 来实现的，真正的数据资源都被分散的存储在资料中心的某台存储服务器的某个磁盘中，并且为了保持服务等级协议（SLA）的服务水准，数据会被分散的存储在不同的磁盘中。每台存储服务负责管理一组磁盘阵列，通过调用 REST API 来对磁盘的存取进行管理。这种分散的存储方式可以保证当某个存储区发生故障或者磁盘被损坏时，可以由其他存储着相同数据的存储区提供数据给应用程序。同时，也可以分散存储的负载和流量，从而提高了存储的可扩展性和数据一致性[25]。

2.4.3 基于 Azure 的人脸识别服务

本项目中使用的人脸识别算法主要是通过计算和数据资源来实现的。我们将对算法的直接使用，以服务的方式面向用户，在多用户并发请求并且需要大规模的计算资源来实现算法运算的情形下，云平台会按需扩大计算资源，并在请求数量降低时按需减少计算资源，将其恢复到正常水平。大规模用户的访问也会造成大规模数据的堆积问题，云平台需要能及时响应数据访问或数据存储的请求，并且保证在数据量不断增大的同时，能有效地做到数据库和数据存储容量的扩张、备份并保证安全性[26]。

同时本平台也要实现对用户的计算资源和数据存储资源的开放功能，要确保能够提供足够的计算资源访问并使用的云服务平台。在 Azure 平台上可以将算法接口调用和计算资源分隔开，通过云服务的方式用户可以方便快捷地实现算法的运算并对算法资源进行有效的管理操作。Azure 提供了虚拟机群组，使用多台分散的服务器来保障人脸识别算法在被大规模并发调用时，能顺利执行。当然，在使用虚拟机群组的时候，云平台需要能做到保障数据的一致性，即平台某处某节点数据发生变化能快速及时的在其他相关联的节点中得到相应的变化处理[27]。Azure 中提供的数据云存储服务，就是使用了分布式存储，好处是可以实现数据一致性，并且保证数据变化时能够被及时更新处理。

根据之前的介绍和分析，我们可以得出这样的结论，即 Microsoft Azure 平台提供的基础设施即服务（IaaS）和平台即服务（PaaS）都能够满足人脸识别算法在平台上实现的要求。云服务提供的虚拟机群组，可以将本项目实现需要的各部分的组件和资源整合，同时可以通过云服务来配置、管理和实现组件相关的各项功能。Azure 平台提供的非关系型的数据存储服务，可以满足本项目对于大规模人脸数据图像、人脸特征索引信息的存储和管理，也可以管理信息在组件中的流通等[28]。综上所述，本文需要实现的基于云平台的人脸识别云服务平台是可以在 Microsoft Azure 平台上进行部署、开发和实现的。

2.5 RESTful API

REST 的概念是由 Roy Fielding 博士在 2000 年首次提出的。REST 是 Representational State Transfer 的代表，它是一种软件设计的风格。RESTful API

在目前是一套比较成熟的互联网应用程序的 API 设计的理论[29]。REST 架构对如今复杂的网络服务情况非常适用，特别是在需要高性能计算的云计算和分布式系统中更是有一定的优势。

REST 设计框架包括了若干原则和条件，满足了这些原则的设计才可以被称为 RESTful。部分原则如下所示：

1) 使用的是基于客户端/服务器的设计架构。
2) 服务器与客户端的通信是无状态的，客户端发起的请求访问需携带必要的信息，而无需关心请求时的状态。无状态性提高了灵活性、稳定性等。
3) 使用了统一的操作接口，这在 REST 中是非常重要的设计点，通过统一的接口降低了层级之间的耦合性并且提高了功能的独立性。
4) 采用唯一的标识资源的方法，在 HTTP 协议中就是使用 URL。并通过指定方法来操作资源，如 HTTP 中的 GET、POST、DELETE 等方法。

API 需使用通信协议，本文平台中使用的是 HTTP 协议。通常应该尽量设计将 API 部署在其特有的域名之下，并将现在使用的版本号放入到 URL 当中。API 中使用的 EndPoint（路径或终点），是用来指明 API 的具体的 URL。URL 在 RESTful API 架构中是一种资源，在 URL 格式中一般是使用名词，通常设计中是对应于数据库中的表的名称的，由于表是实体的集合，所以在 URL 中应该使用名词的复数。对于具体资源的操作，常常选择使用 HTTP 的动词来表示[30]。

在设计 RESTful API 时，除了我们之前提到的对名词和动词的概念之外，还需要关注内容格式，即 REST 中的 R 所指代的 Representation（表现），它将数据实体用外在的形式表示。比如 URI 只是表示资源实体的位置，并不是它的形式，所以在 HTTP 的 Request 的 Header 中使用 Content-type 和 Accept 对其进行指定。本系统中主要是通过 MIME Types 来定义内容格式。

超文本驱动（HATEOAS）在 RESTful 框架设计中是非常重要的概念[31]。Web 上的应用程序可被看成有限状态机，它由大量的应用程序的状态组成。通常，资源之间的关联是通过超链接实现的，所以超链接既可以表示资源间的关系，也可以表示状态的迁移。超文本中包含了数据和表示状态迁移的各种语义。通过超文本，可以将服务器提供的资源暴露出来，这些资源在执行的时候被解析发现，而并不是之前定义好的。实际上，超文本定义了服务器端提供的服务的协议。相应的，客户端可以根据超文本的状态迁移语言得到响应。超文本的状态迁移语义是稳定的、可以保持不变的。

2.6 本章小结

本章主要对云服务平台所采用的相关原理和技术进行介绍。首先对人脸识别技术进行了简单的介绍，然后介绍了 Azure 云平台提供的基础设施即服务，在此基础上又讨论了 Azure 云平台提供的平台即服务，详细介绍了平台相关的计算、存储和与人脸识别算法相关服务的实现。这些服务保障了本文项目在 Microsoft Azure 上可以被部署实施，且 Azure 提供了方便快捷的配置方法和界面化的用户管理，使得用户对资源的配置和管理可以更加轻松地实现。最后，本章又对 RESTful API 设计进行了阐述，本文项目提供的服务在面向用户时表现为 API 接口调用的形式。RESTful API 的设计风格在高性能计算的云计算和分布式系统中具有优势，这正好符合本文系统中提供的人脸识别云服务的需求。

第 3 章 云服务平台需求分析与概要设计

3.1 引言

本章主要是对本文项目中实现的云服务平台进行需求分析和概要设计。本文的目标是设计一个基于 Microsoft Azure 平台的人脸识别云服务平台，该平台的主要面向用户是开发人员，为此本章分别从功能和性能上对云服务平台进行需求分析。同时，在明确需求分析的基础上，对云服务平台进行概要设计。

3.2 云服务平台需求分析

本文平台主要的用户角色是开发人员，云服务平台的设计目标是让开发人员通过使用人脸识别云服务平台可以方便的实现其本地人脸识别应用开发。本节主要是分析开发人员在实际使用云服务平台时，对平台在功能上和性能上的具体需求。

3.2.1 云服务平台功能需求

考虑开发人员在实际开发过程中对人脸识别云服务平台的需求，本平台主要的功能需求如下：

1) 云存储功能：通过使用云平台提供的云存储服务，保障用户有独立安全的存储空间。基于 Azure 平台对人脸识别算法服务实现过程中使用的数据（包括人脸图像和脸部特征点等）进行存储和管理。开发人员可以通过调用 API 将数据上传到云平台数据存储中，从而既可以省去多次同样的数据传输操作，也可以节省网络流量。用户只需通过数据索引的方式引用数据，这也节省了相应的运营成本。本文平台 API 提供的人脸识别、新建人脸分组、新增人脸等接口都是对相关数据的操作。这些数据也可以为以后的算法研究改进提供数据样本。

2) 云平台计算功能：云服务平台实现了人脸识别云服务所需的相关算法接口，主要涉及的算法有：人脸识别、特征点提取、大规模索引等。云平台计算服务主要负责算法计算、算法资源分配以及计算节点的管理等的实现，使得用

户可以选择脚本或文件等方式将本平台提供的算法接口应用到任意的联网终端上，具有平台无关性，保证算法可以高效的为用户提供服务。

3) 人脸识别 API 功能：采用 RESTFul Web API 服务框架[32]，使用 HTTP 协议进行通信，提供给开发人员多平台下的人脸识别算法服务 API 兼容接口，从而开发人员本地的应用程序可以管理和使用云平台上的数据存储。本项目提供的接口服务必须通过认证才能被调用访问。用户在调用 API 接口时需要提供合法的身份信息，本项目使用用户产品订阅标识进行认证。项目 API 有以下几种：人脸识别 API、分组个人 API 、人脸分组 API 和大规模人脸识别 API。

总结上述功能需求，可以设计出云服务平台用例图，如图 3-1 所示。

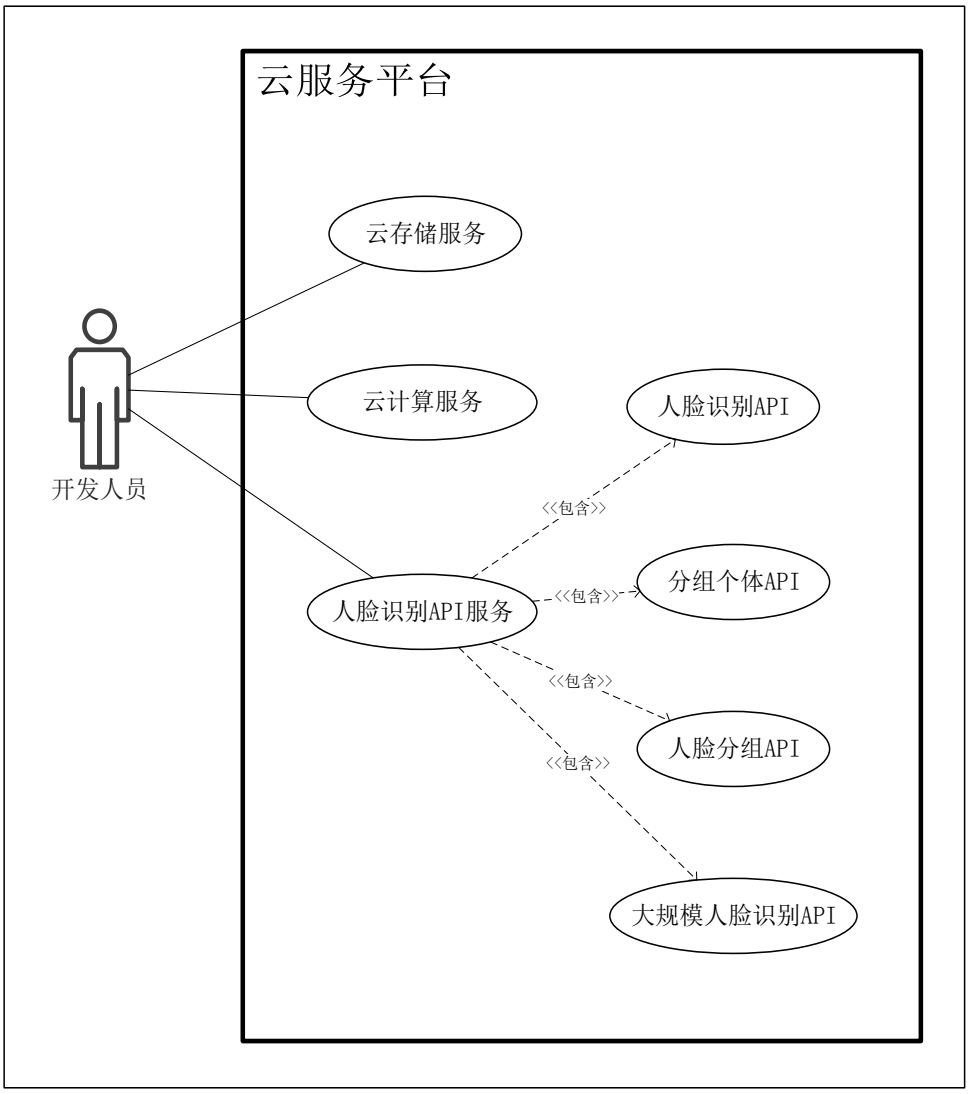

图 3-1 人脸识别云服务平台用例图

3.2.2 云服务平台性能需求

上一小节对云服务平台的功能需求做了简要分析，本小节是对云服务平台的性能需求进行分析。云服务平台的性能需求与功能需求、平台服务质量等密切相关，它贯穿在整个项目开发周期中，是我们需要关注和考虑的问题。就本文项目而言，表 3-1 详细列出了开发人员对于云服务平台的性能需求。

表 3-1 云服务平台性能需求表

需求名称	需求描述
准确性	保证人脸识别云平台提供的各类服务如存储、算法和接口服务等的准确性。
易用性	提供面向开发人员的API服务，在具体界面展示上满足简单易用。
健壮性	对于不合乎常规的输入要能够有对应并且正确输出。
消息处理速度	任务分发器结合计算节点状态和任务分配算法分发任务来提高消息处理速度。
可维护性	对于云服务平台出现的突发情况下可以快速维护并恢复API服务。
可扩充性	遇到使用增加的情况时，通过增加Azure平台的计算节点来提升平台的吞吐量。
数据安全性	通过使用云服务平台提供的数据存储服务，保证安全性。
大规模性	在不折损太多精度的前提下，满足快速大规模人脸识别的需求。

3.3 云服务平台概要设计

根据之前对本文云服务平台功能需求和性能需求的分析，本节将从软件工程的视角，分别从逻辑架构和物理架构上对云服务平台进行概要设计。

3.3.1 逻辑架构设计

本文项目是在是在 Microsoft Azure 平台上搭建，根据本文研究的云服务平台提供的服务和最终的实现功能，将该平台的逻辑架构设计为三个部分，详细如图 3-2 所示。

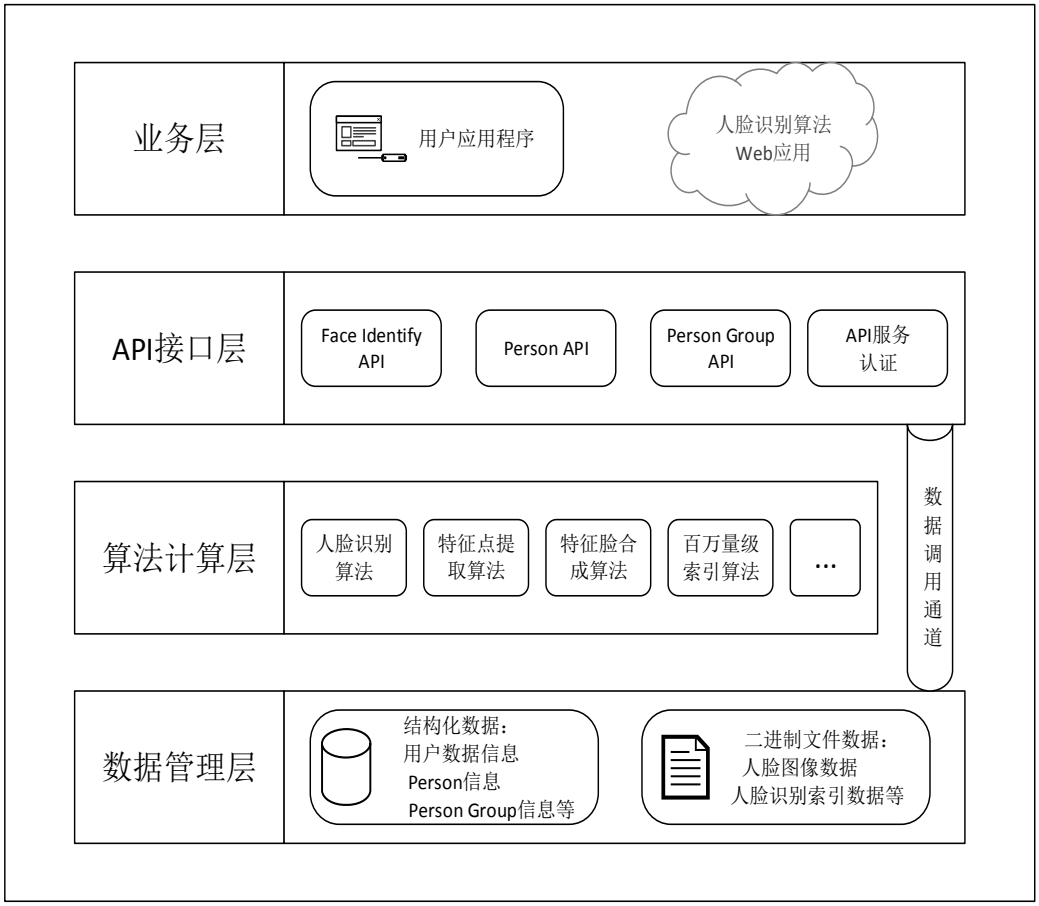

图 3-2 云服务平台逻辑架构设计图

1) 数据管理层：本文实现的平台所关注的主要有两种数据形式：结构化数据和二进制文件数据。作为数据存储层，通过使用 Azure 提供的 Storage 服务，分布式地对人脸图像数据、人脸识别索引数据、用户信息等数据信息进行存储和管理。这种分布式的云存储管理，在保证了数据一致性的同时，也保证了数据传输速度。

2) 算法计算层：本层在数据管理层提供的数据基础上，负责保证实现人脸识别云服务所需的相关算法，包括：人脸识别、特征点提取、大规模索引等。开发人员若在应用程序开发时本地实现这些算法，需要投入大量计算资源以保证算法的实现。而通过云服务平台的算法计算层，可以在多用户并发的情况下，保证用户对算法的需求[33]。Azure 平台有效的处理了负载均衡的问题，并且通过 CPU 对资源的自动缩放策略，确保了算法和资源的时效性。

3) API 接口层：该层负责提供算法服务的 API 接口给开发人员调用，本项目所设计与实现的 API 包括：人脸识别 API、分组个人 API、人脸分组 API 以及大规模人脸识别 API。该层面向图中的业务层，开发人员可以通过调用这些 API 完成对平台算法计算层和数据存储层的调用，从而实现其具体的业务应用。通过在算法层调用之前对 API 接口调用的权限认证、身份认证等操作，确保算法和资源能够被合法地调用。

上述服务层最终是为了实现业务层的服务，由平台的逻辑架构图可以看出其中业务层还可以通过一定的数据调用通道直接使用数据存储层的相关数据，这点可以使用户在业务层更方便地使用相关数据[34]。但是业务层不能直接使用算法计算层的资源，它必须调用 API 接口来使用算法。通过架构图中展示的所有服务层，开发人员可以使用该层级服务来实现本地的应用程序开发或是相关的移动 WEB 应用开发等。

3.3.2 物理架构设计

根据之前的图 3-2 展示的本文云服务平台的逻辑架构设计，我们可以对应设计出该平台的物理架构，两者的映射关系可以通过图 3-3 来详细描述。可以看出对应平台逻辑架构的层级设计，在数据存储部分，为了保证实际使用数据的一致性，采用了 Azure Table Storage 结合 Azure Blob Storage 对数据进行云存储。而使用 Azure 提供的云服务也可以做到计算资源合理分配以及负载均衡管理，从而使算法可以更有效地为用户服务。因此，在物理架构上，我们通过将其组件分离对应逻辑架构上的层级服务，成功构建了可以满足开发人员的关于人脸识别的云服务平台。

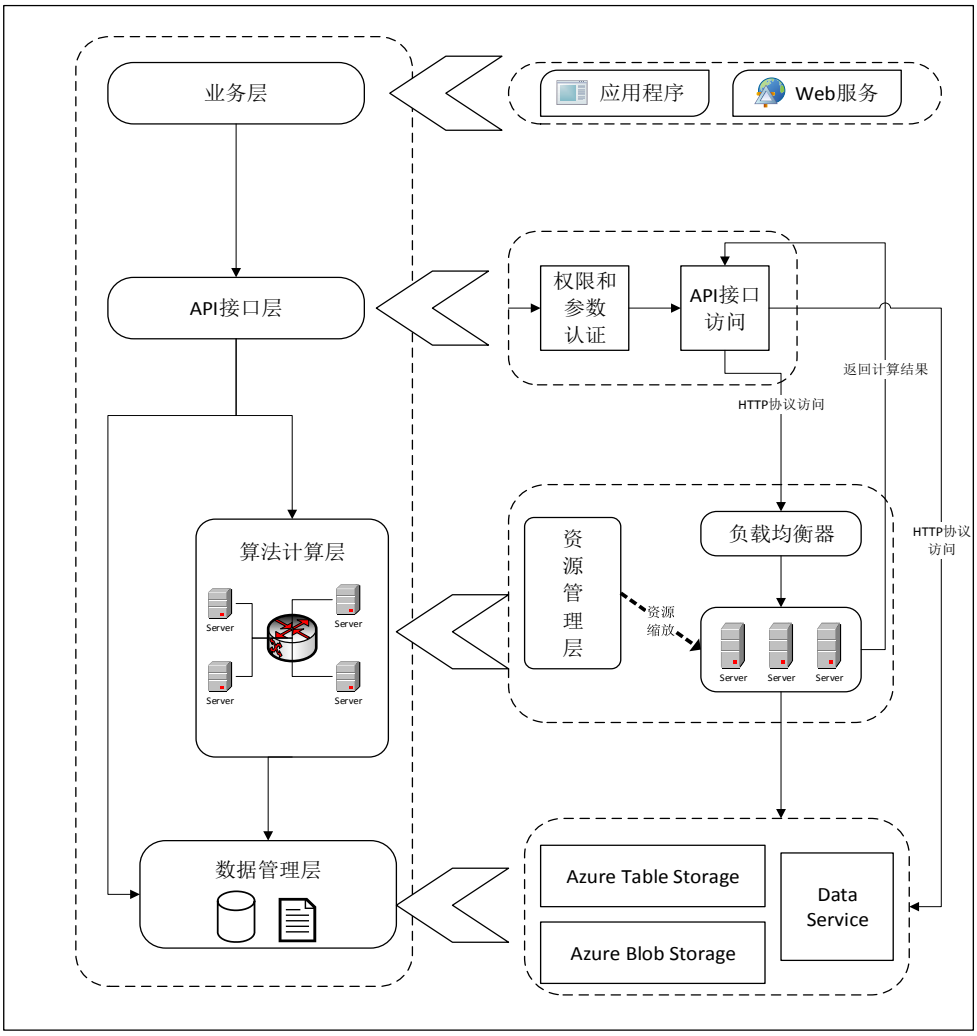

图 3-3 云服务平台物理架构设计图

1) 数据管理层：图像信息、索引信息、人脸分组信息和分组个人信息等数据是本云平台使用的主要数据。在对这些数据的实际管理中，使用 Azure 的 Table 存储来存储管理结构化的数据信息，而非结构化的二进制文件数据信息则用 Blob 存储来进行存储管理。为了保证数据在存储中的唯一性，对不同类型的数据信息，我们都指定了一定的命名规则，从而可以避免在对数据的操作中遇到数据冲突的情形。Azure 平台为开发人员对数据的调用操作提供了具体的 SDK，开发人员通过网络使用该 SDK 即可。数据管理层负责平台的数据管理，并为其上层提供数据服务调用。

2) 算法计算层：该层主要负责实现与云服务平台相关所有算法的计算，通过使用从下层的数据管理层获得计算所需的各种数据，服务于上层的服务层对

其 HTTP 协议访问的调用，并将算法的计算结果返回给上层。它是由多个服务器组成的服务器集群，通过 Azure 提供的负载均衡来进行服务管理，将上层的访问请求分发到服务器集群中去处理，实现了集群中各个计算资源的合理调用。并通过资源缩放管理器，按需缩放资源。算法计算层是整个云服务平台的算法计算核心，所有对于本平台人脸识别算法的调用都要通过调用该层提供的算法接口来实现。

3) API 接口层：API 接口层在云服务平台上的设计是直接服务于上层业务层的。从业务层发起的各种请求，到达 API 接口层，该层负责对请求进行权限认证、参数合法性认证等，只有认证通过，才能继续进行之后的接口调用。API 接口层的重要作用是保证了下层的所有服务面向上层接口的唯一性，从而确保了业务层调用的合法性和唯一性；同时，它也保护了下层算法计算和数据管理层资源的安全性，使得下层的资源能够被合理有效地使用。所以，API 接口层可以被理解成隔离层，它有效地将上下业务层隔离开，既过滤了不合法的接口调用，也避免绕过接口调用直接操作资源等可能导致的危害，是整个架构中起着非常重要作用的中间层。

3.4 本章小结

本章主要是对人脸识别云服务平台进行需求分析与概要设计。首先从平台主要用户开发人员的角度出发，分别从功能上和性能上对云服务平台的需求进行分析。然后，根据需求分析结果，对云平台的提出概要设计，分别从逻辑架构和物理架构角度设计，将平台划分为数据管理层、算法计算层和 API 接口层。本章节为后续章节对云服务平台的详细设计与实现做了充分的准备。

第 4 章 云服务平台详细设计与实现

4.1 引言

本章主要研究如何详细设计与实现基于云平台的人脸识别云服务平台，目的是向开发人员提供人脸识别算法的服务接口，使开发人员可以通过调用所需接口轻松实现本地应用程序开发。具体是以 Azure 平台为基础，通过使用该平台提供的平台即服务功能，实现项目的平台搭建、部署和扩展等操作。上一章对云服务平台进行了需求分析和概要设计，将平台架构主要按层级服务进行划分，主要为三层：数据管理层、算法计算层和 API 接口层。本章对这些不同功能的层级服务进行详细设计与实现。

4.2 数据管理层的设计与实现

数据管理层是基于 Azure 平台使用其 Table Storage 和 Blob Storage 两种存储方式对平台中的数据信息进行存储和管理。对于结构化数据，可以通过对象关系映射（ORM）技术进行管理；而对于二进制文件则可以利用 Auzre 提供的 SDK 对 Blob 文件进行各种相关操作。

数据管理处理的数据包括人脸图像文件、索引文件、人脸特征点信息、用户信息等。通过对这些数据的分析，可以总结出它们具有以下这些特性：

1) 人脸图像文件，属于二进制数据信息。对其约束性要求不强。
2) 人脸识别索引文件，它需要将索引的人脸特征集合与人脸分组中的个人标识联系起来，从而在人脸识别范围数量级很大的时候能够返回有效信息，是结构化强的数据信息。
3) 人脸特征点集合信息。它从具体的某个人的某张脸提取出来的特征点信息的集合，从属于某个人，也是结构化的数据信息。
4) 用户数据信息，用户可以创建并管理自己的人脸分组，以及分组中个人的信息，本文利用 Azure 平台为用户提供的产品订阅标识，来区分用户，为用户维护特有的数据空间，保证数据的完整性。

5) 人脸图像文件及相应的被提取的人脸特征点集合会随着用户的使用而不断增多，为此需要考虑如何有效地分配空间，对数据进行存储管理。
6) 由于本云服务平台所使用算法对人脸图像数据有极大的依赖性，数据信息在使用时需要确保其自身的一致性。为了确保算法的准确性，若是对数据进行了信息变动操作，则需要考虑如何操作既可以有效地反应在数据上，又不会影响到算法的运行效率。

综上所述，我们可以对之前提到的数据信息进行分类存储，将人脸图像文件、人脸识别索引文件等数据信息使用 Azure 提供的 Blob 存储，而与云平台相关的用户信息、人脸特征点集合信息等数据信息则使用 Azure 提供的 Table 存储来进行管理。

4.2.1 Azure Blob 存储

使用 Azure Blob 存储提供的、服务，通过调用其 REST API 实现对文本和二进制数据的上传、管理、组织和删除的操作。云服务平台 Blob 服务在 REST API 中支持以下的资源，如图 4-1 所示。

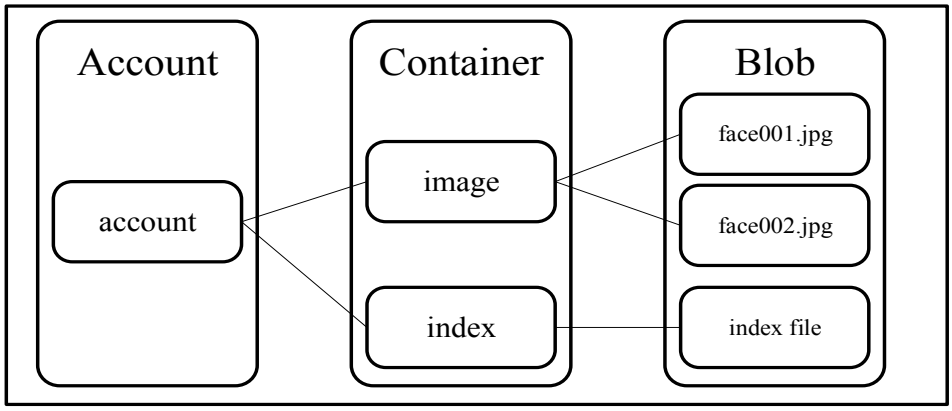

图 4-1 云服务平台 Blob 服务组件图

由上图可以看出，Blob 文件在 Azure 中是用容器（Container）来创建的，我们为使用同一个产品订阅标识（Subcription Id）的用户分配同一个容器。所以容器实质上就是数据存储空间，功能相当于一般的文件系统中的文件目录，可以存储不同类别的数据信息。通过使用不同的容器可以将不同种类的数据存储在不同的数据空间中。由于 Azure 上 Container 或 Blob 资源都对应一个基本的 URI, 将使用的数据放在不同的空间中也能提高用户对文件的查询速度。对

于每个使用人脸识别云服务平台算法接口的应用程序，根据它们的 Subscription Id 的不同，平台为它们创建不同的 Container 来放置数据从而保证了数据的独立性。本文云服务平台主要使用 Blob 来存储人脸图片数据、二进制索引文件等。

4.2.2 Azure Table 存储

通过 Azure 平台提供的 Table 服务对结构化的数据存储创建表，以及插入、更新、删除数据等，本文云服务平台 Table 服务组件图可参考图 4-2。云服务平台使用 Table 对人脸分组（Person Group）信息、分组个人（Person）信息和用户信息等进行存储。

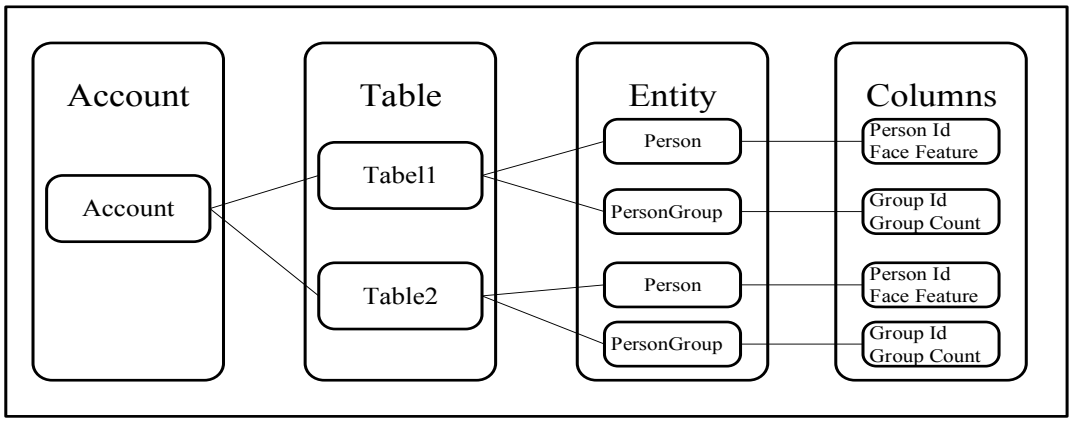

图 4-2 云服务平台 Table 服务组件图

Azure Table 存储中每行对应一个实体，每行中的列则对应着实体属性。在表中每个实体必须拥有其特有的 Partition Key 和 Row Key，它们可以与账户名称及表名一起定位表服务中实体的具体位置。在本文研究的人脸识别云服务平台中，我们为每个使用不同账号的应用分配相应的表空间，在表中用不同的实体对平台中的不同种类的信息进行存储。以下列举本平台中相关数据信息的实体结构设计，表 4-1 和表 4-2 显示了账户信息在平台中被存储的两种实体结构。

表 4-1 账户实体结构 A 表

属性名称	数据类型	数据信息
PartitionKey	String	Subscription Information
RowKey	String	PersonGroupCount
Timestamp	DateTime	Record Time
PersonGroupCount	Int32	Record Person Group Count

表 4-2 账户实体结构 B 表

属性名称	数据类型	数据信息
PartitionKey	String	Subscription Information
RowKey	String	PersonCount
Timestamp	DateTime	Record Time
PersonCount	Int32	Record Person Count in a Group

关于人脸分组（Person Group）数据信息组成的实体，如表 4-3 所示：

表 4-3 人脸分组实体结构表

属性名称	数据类型	数据信息
PartitionKey	String	PersonGroup
RowKey	String	PersonGroup ID
Timestamp	DateTime	Record Time
Guid	Guid	Guid for Person Group
Name	String	PersonGroup Name
PersonCount	Int32	Numbers of Person in Group
TrainingStatus	String	Record Training Status
UserData	Binary	User’s Payload Data

关于分组个人（Person）数据信息组成的实体，则如表 4-4 所示：

表 4-4 分组个人实体结构表

属性名称	数据类型	数据信息
PartitionKey	String	PersonGroup Guid
RowKey	String	Person ID
Timestamp	DateTime	Record Time
Name	String	Person Name
UserData	Binary	User’s Payload Data
FaceID_1	Binary	Face Feature of Face 1
FaceID_2	Binary	Face Feature of Face 2
.....		
FaceID_248	Binary	Face Feature of Face 248

以上详细介绍了本平台数据管理层在 Azure 上数据存储管理的实现，通过 Azure 提供的 Blob 和 Table 服务，实现了对结构化和非结构化数据的管理。在对数据信息的具体操作上，选择了使用 LINQ 语言集成查询技术。该技术可以在 Visual Studio 上，通过编写 C#代码，调用相关 API 和查询数据库相同的方式来操作存储在 Azure 平台中的数据。

4.3 算法计算层的设计与实现

本论文实现的云服务平台实现了人脸识别云服务所需的相关算法接口，主要涉及的算法有：人脸识别、特征点提取、大规模索引等。所以算法计算层主要负责算法计算、算法资源分配以及计算节点的管理等的实现。

对于算法计算层而言，当上层 API 接口层的服务请求到达时，首先将这些任务放入队列中，使用任务分发器来处理任务请求队列。然后，任务分发器根据计算节点访问路径和节点状态的分析结果，将不同的任务请求分发到它认为的最有效的节点完成计算任务，最后可以将计算结果返回给 API 请求，从而在多任务请求的情况下，合理并高效地完成对各任务的响应。图 4-3 是对算法计算层计算节点模块和算法组件对任务请求处理细节的展示。从图中我们可以看出在本文云服务平台中，实质上使用一组服务器去完成各算法请求的分布式计算，每个独立的计算节点都可以完成相应的算法计算并返回计算后的结果。

为了完成这样的架构，在设计和实现算法计算层时，需考虑以下问题：

- 任务分发器的功能设计：一方面，对于收到来自上层的服务请求，任务分发器会计算节点访问路径，并分析个节点的状态，如节点的 CPU 使用率等，来明确将任务分配给最合适的节点去处理。另一方面，在当前节点都被使用的情况下，可以使用任务队列来放置未处理的任务，等到有节点空闲的时候，再去执行相应的计算任务。
- 算法计算层的封装性：算法计算层中的每个节点都是由一台虚拟机组成，对节点需按照统一的 HTTP 协议来访问。如此才能保证请求的任务被顺利完成，也确保了计算节点能够以一定的方式被任务分发器统一地分配、调用和管理。
- 计算节点的数据无关性：在本层中的计算节点是被要求与数据无关的，只有在具有数据无关性的前提下，才能确保任意一个被任务分发器调用的计算节点，对同一调用请求运算的结果是一致的。从而保证了计算的正确性，也可以确保每个计算节点都可以合理地被调用，并可以对其进行扩展。
- 计算节点的可扩展性：在计算节点数据无关性的讨论中已经提到过可扩展性，对于人脸识别算法这种对计算资源有很大需求的算法而言，需通过扩展计算节点的方式来提升云平台的整个吞吐量，从而可以应对用户量增大的情况。

➢ 统一的访问方式：对于访问方式的设计是，只提供平台内部组件的访问接口，对外部请求调用是不提供接口的，也就是不允许来自平台外部的调用。

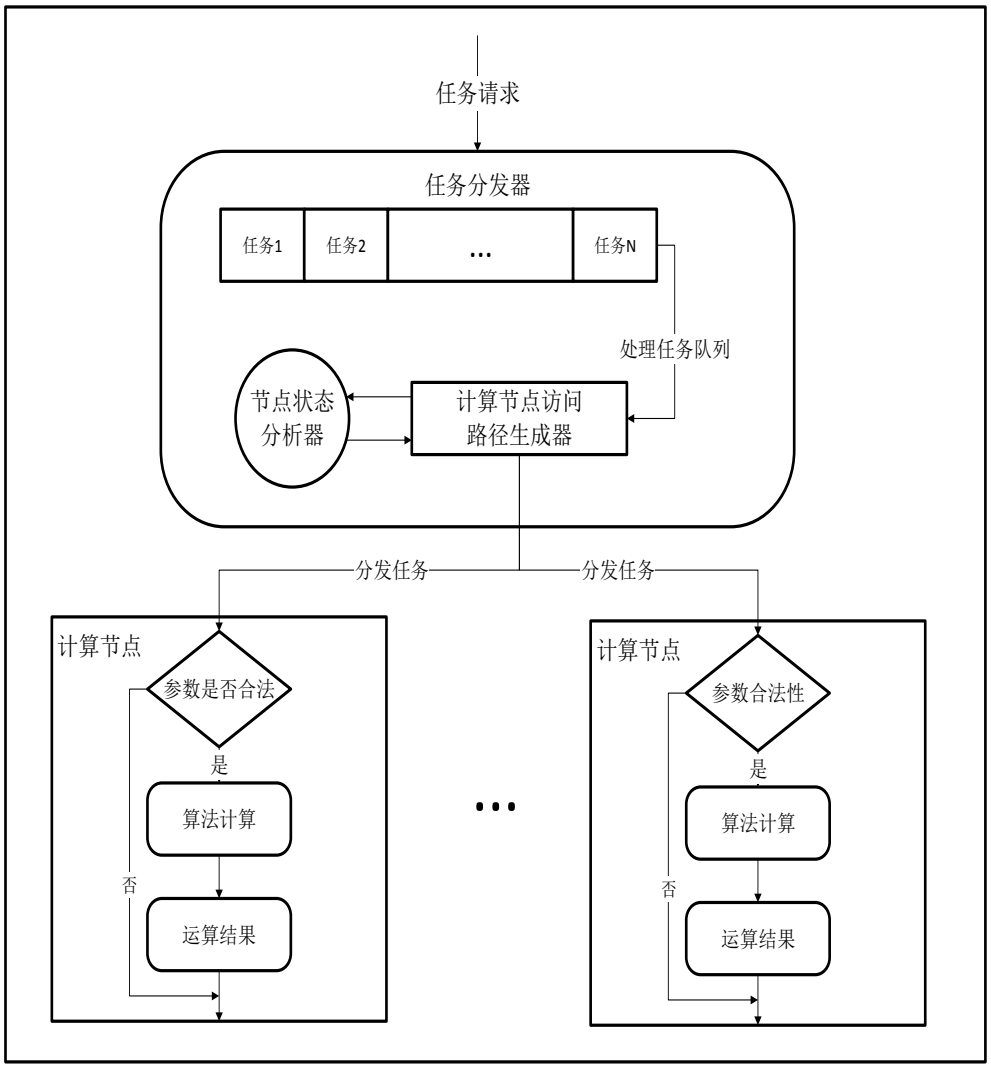

图 4-3 算法计算层计算节点模块图

计算节点是算法计算层中非常重要的组件，在本平台中采用 Windows Communication Foundation（WCF）框架来设计和实现计算节点。WCF 服务框架模型可以参考下图 4-4，主要由三部分构成：

1) 服务类（Service）：一般是采用 C#或 VB.NET 等语言进行编写，实现一个或多个方法。本平台是使用 C#语言编写，实现了多个相应的方法。
2) 宿主（Host）：是指应用程序域、进程，服务将在该环境下运行。ASP.NET、WPF 和 COM+都可被当作宿主。本平台采用.NET 作宿主。

3) 终结点（EndPoint）：主要负责提供对外的接口地址，允许客户端通过其实现对服务的访问。本平台使用的是 HTTP 通信协议进行通信。

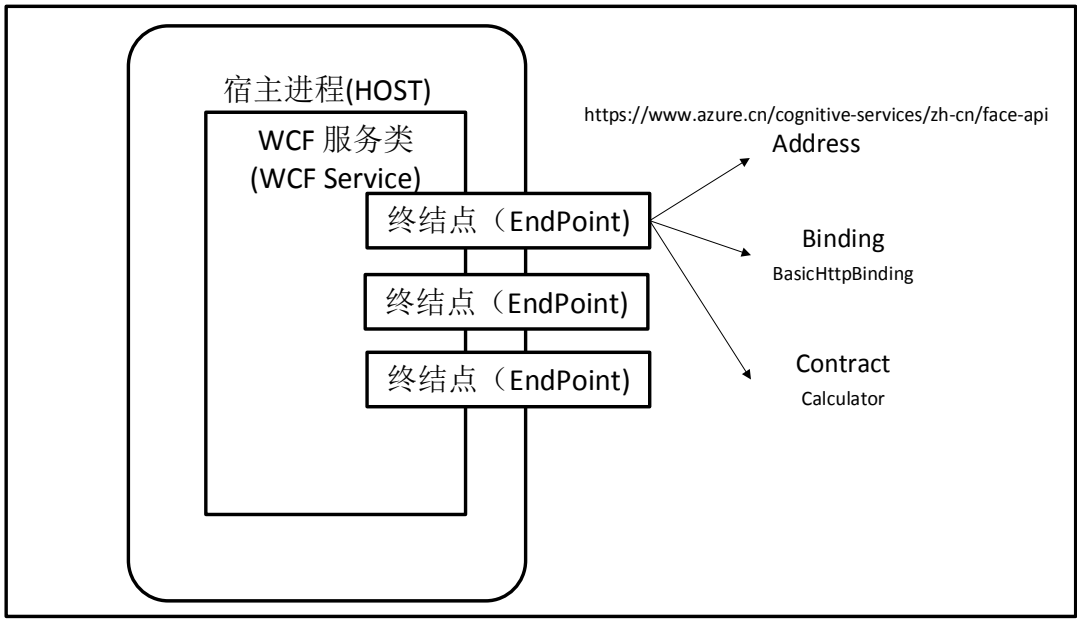

图 4-4 WCF 框架模型图

4.4 API 接口层的设计与实现

API 接口层是面向业务层提供 API 接口，通过该接口，实现对平台人脸识别算法的各种调用。其中来自业务层的调用，可能是来自外部应用程序的接口调用，也可能是直接来自平台内部的接口调用，因此需要在调用接口之前对用户身份、参数是否合法等前置条件作确认操作。本文项目使用 Web API 结合 ASP.NET MVC 进行开发，Web API 框架是面向 HTTP 协议的，与 WCF 相比而言，Web API 没有那么多配置，只是面向 HTTP 协议的设计。它与 ASP.NET MVC 结合，返回 Json 对象，并且封装了数据的序列化、反序列化，使得接口实现和调用都更加简单[35]。图 4-5 对本文项目中实现的 API 框架流程进行了描绘。

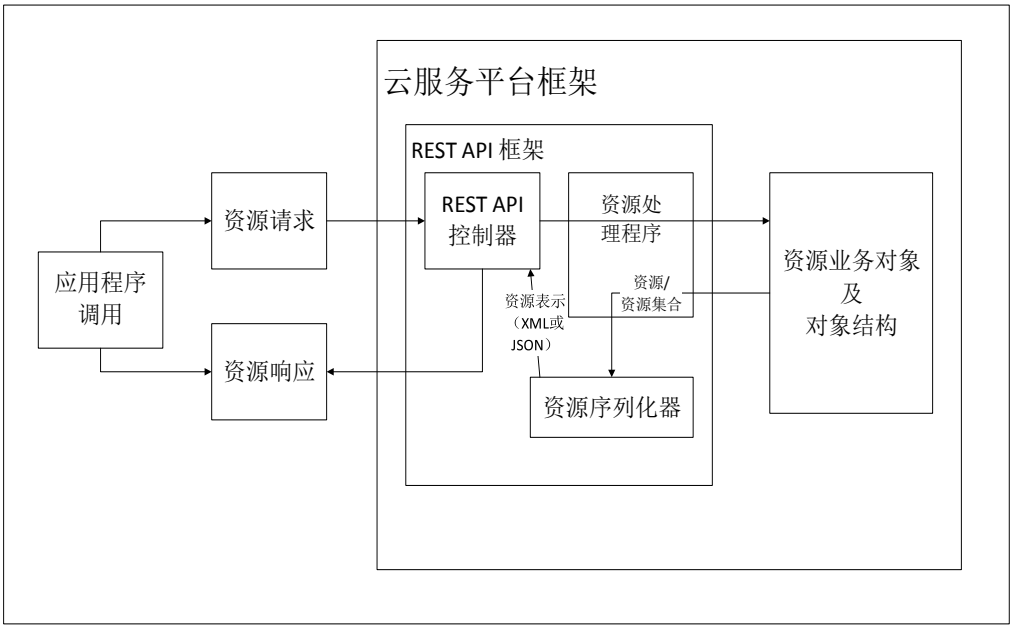

图 4-5 云服务平台 API 框架图

API 接口层在实现的过程中，以下几方面是我们关注的：

➢ 数据序列化：Web API 目前支持两种数据格式的可序列化，即 Json 和 xml。通常在 HTTP 请求中，Web API 会对数据自动使用 xml 序列化，同时使用了 Newtonsoft.Json 框架对 Json 类型的数据进行序列化。在 Web API 的 Json 序列化同时支持了对匿名类型的序列化，这一特性在本文云服务平台的使用方便了开发人员自己组装需要的数据并返回结果[36]。

➢ RESTful 架构模式：在本平台 API 接口调用的实现上，使用了 RESTful 架构模式，从而使得算法计算层提供的每个算法的接口都有其对应的 URL（统一资源定位符）。这使得业务层的应用程序可以遵照 HTTP 协议，通过对算法接口的 URL 发起地址访问请求，顺利调用算法接口。Web API 框架会根据具体 HTTP 请求的方法（Get、Post、Delete 等）去控制器找到匹配的 Action，从而实现跨平台的服务调用。

➢ 参数的绑定：在 Web API 框架中使用了参数绑定类。它做出了如下规定：只允许 Action 的参数列表中的一个参数，从 Http Post Body 中反序列化出来。当 Action 的参数列表中只有一个参数，Web API 会尝试直接将 Http Post Body 反序列化为对应的对象。

设计云服务平台 API 调用流程如图 4-6。跟据此流程图，可以将 API 调用过程总结为以下的几个步骤。

1) 用户上传运算中需要使用的图像数据到云存储中，云服务平台返回存储的名称，可以通过这一名称作为图像的唯一的标识来请求获取其内容。
2) 算法被调用之前需要经过 API 接口用户身份认证和参数的合法性验证，用户才可以顺利对算法进行调用。
3) 算法执行过后，需要将运算结果进行数据格式转化，将用户需要的数据信息通过 API 接口提供给用户。

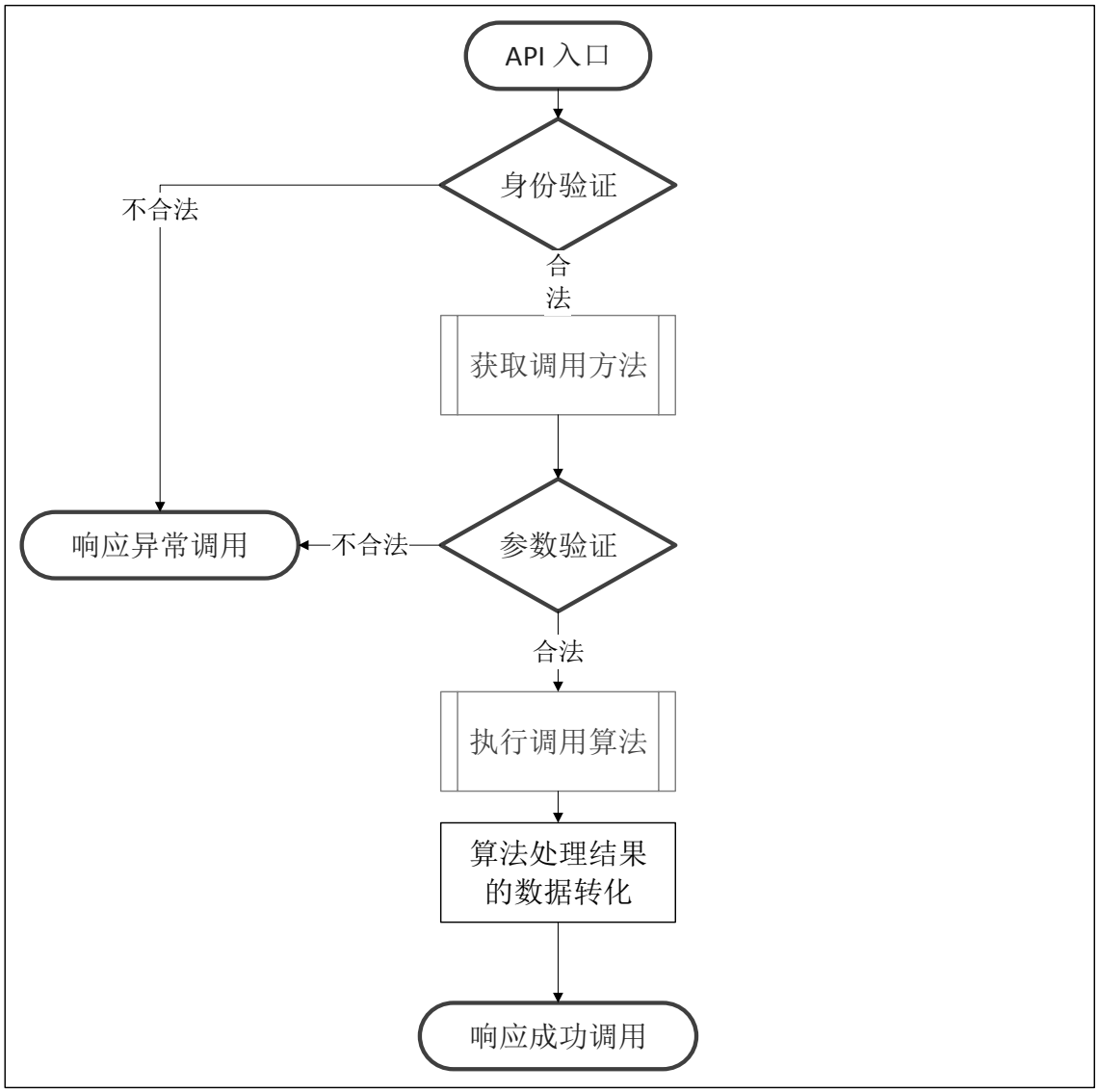

图 4-6 API 请求响应过程图

本文云服务平台具体实现的人脸识别 API 接口与地址参数如表 4-5 所示：

表 4-5 云服务平台部分 API 的访问地址与参数表

API 名称	地址	参数
Face Identify	POST/identify	faceIds 、 personGroupId 、 maxNumOfCandidateReturned（可选）、confidenceThreshold（可选）
Add a Person Face	POST/persongroups/{personGroupId}/persons/{personId}/persistedFaces[?userData][&targetFace]	personGroupId、personId、userData（可选）、targetFace（可选）
Delete a Person Face	DELETE/persongroups/{personGroupId}/persons/{personId}/persistedFaces/{persistedFaceId}	personGroupId、personId、persistedFaceId
Get a Person Face	GET/persongroups/{personGroupId}/persons/{personId}/persistedFaces/{persistedFaceId}	personGroupId、personId、persistedFaceId
Create a Person	POST/persongroups/{personGroupId}/persons	personGroupId、name、userData（可选）
Delete a Person	DELETE/persongroups/{personGroupId}/persons/{personId}	personGroupId、personId
Train Person Group	POST/persongroups/{personGroupId}/train	personGroupId
Get Person Group Training Statuse	GET/persongroups/{personGroupId}/training	personGroupId

综上所述，可以看出本层实现的接口服务主要是两类，分别是面向业务层的接口和面向数据管理层的接口。面向业务层，提供了算法服务接口，业务层的应用程序可以通过访问算法接口对应的 URL 地址来实现对算法的调用；而面向数据管理层，则是实现了用户数据的管理，主要是指对用户的人脸图像数据进行上传和下载等。具体接口的设计、实现和改进过程将在本文的第五章进行详细介绍。

4.5 本章小结

本章主要介绍了云服务平台各层级服务的详细设计与实现，分别有数据管理层、算法计算层和 API 接口层。

数据管理层为云服务平台的其他层级管理数据信息。由于在架构上采用的是松耦合的方式，所以选择了基于 Azure 的云存储的方式来实现数据管理、备份、迁移、扩张等，并且可以保证数据的统一性。通过使用 Azure Blob 存储 和 Table 存储的服务，更加为平台的数据存储提高了效率和安全，方便了对数据信息的各种操作。

算法计算层主要保证云服务平台的人脸识别相关算法运行，采用 Azure 的云服务来实现该层多线程情形下任务分发到多节点进行计算的过程，有效实现了负载均衡。使云平台功能的实现与计算节点实现了一定程度的解耦，方便了在平台上对计算节点进行管理。同时，也优化了对计算资源的合理分配。

API 接口层则是服务于业务层和算法计算层的接口调用的。结合 Web API 框架与 ASP.NET MVC 框架实现了 RESTful 架构风格的 API 接口，并在此基础上对访问接口的用户身份的合法性和参数的合法性进行了校验操作。本层规定开发人员在遵循 HTTP 通信协议的前提下，对所需的算法服务接口进行调用，并得到接口返回的计算结果。

第 5 章 人脸识别 API 的设计、实现与改进

5.1 引言

第四章对云服务平台进行了详细设计并实现了其各层级服务。本章主要介绍与本文云服务平台相关的具体的人脸识别 API 的设计与实现。并在此基础上，通过实验论证在 Azure 平台上进行大规模人脸识别的可行性，改进与优化之前的人脸识别 API，最终实现大规模人脸识别 API。

5.2 人脸识别 API 设计与实现

本文实现的云服务平台 API 是在第四章讨论过的 WEP API 框架结合 ASP.NET MVC 框架的基础上实现的，在 IIS 上实现借宿，通信采用的是 HTTP 协议。除了考虑云服务平台 API 在技术上的选择，在功能上也需要对其进行划分。前面的章节中提到过，本文平台主要是提供人脸识别 API、分组个人 API、人脸分组 API 和大规模人脸识别 API。其中前三种是本平台提供的基础功能 API，而大规模人脸识别 API 的实现则是对前三种的进一步优化和改进，具体实现方法我们会在本章后续小节中详细介绍。人脸识别 API 主要是根据人脸相似置信度的不同，返回与输入人脸最接近的数据。分组个人 API 主要是对分组中的个人提供管理接口，提供包括新增人脸、删除人脸和修改人脸等功能。人脸分组 API 则是对用户定义的整个分组提供管理接口，包括新增分组、列表分组和训练分组等功能。

5.2.1 人脸识别 API

5.2.1.1 类结构设计与实现

人脸识别 API 负责对用户输入的人脸图像在人脸分组中进行识别，按置信度从高到低排序，返回用户在调用接口时指定数量的人脸。在这一过程中主要涉及到人脸识别、获取人脸分组进行比较等操作。该部分具体由三个类构成，其类图如图 5-1 所示。

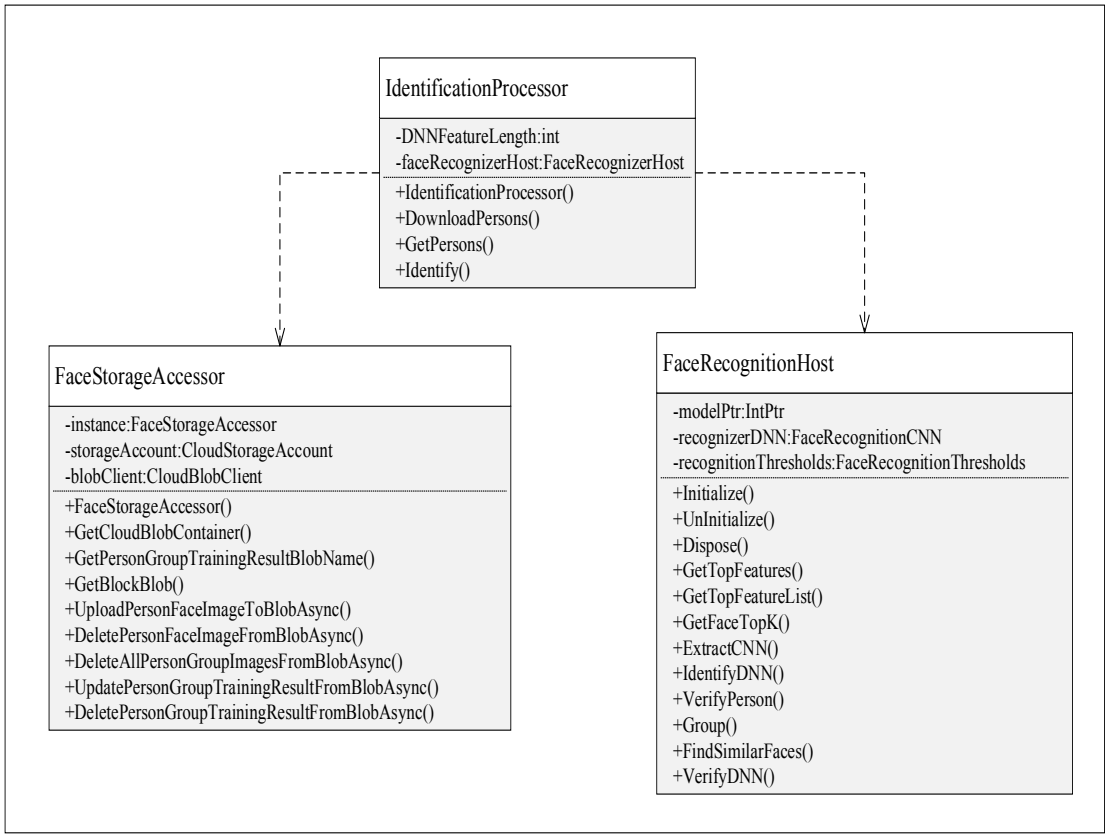

图 5-1 人脸识别 API 模块类图

从上图关系可以看出，在人脸识别 API 模块中 IdentificationProcessor 类依赖于 FaceStorageAccessor 类和 FaceRecognitionHost 类而实现。IdentificationProcessor 类主要是负责管理人脸识别操作的函数，开发在调用人脸识别接口时，首先会调用 IdentificationProcessor 类中的 Identify()方法。FaceStorageAccessor 类则是主要实现了操作 Auzre 上的相关 Blob 数据，比如在人脸识别时需要与 Azure 上的人脸分组数据相比较，FaceStorageAccessor 类提供了 GetCloudBlobContainer()、GetBlockBlob()方法满足了下载 Azure 平台中训练完成的人脸分组数据的需求。在此，训练完成的人脸分组数据是用来进行人脸识别的主要数据集合。FaceRecognitionHost 类负责的是具体的人脸识别算法部分，它提供了封装好的 IdentifyDNN()接口，可以按置信度返回与用户输入人脸最相似的人脸分组中的数据，并返回用户指定数量的结果。

5.2.1.2 流程设计与实现

人脸识别 API 模块的功能是从一个已经存在的人脸分组 （Person Group）中识别出与输入人脸最相似的人脸。API 调用入口为 IdentificationProcessor 类中的 Identify()， 首先它通过调用 FaceStorageAccessor 类下载被比较的人脸分组中的 Persons 集合，然后再调用 FaceRecognitionHost 类中的 IdentifyDNN()方法得到按置信度排序的识别结果。图 5-2 描绘了人脸识别 API 模块时序图。

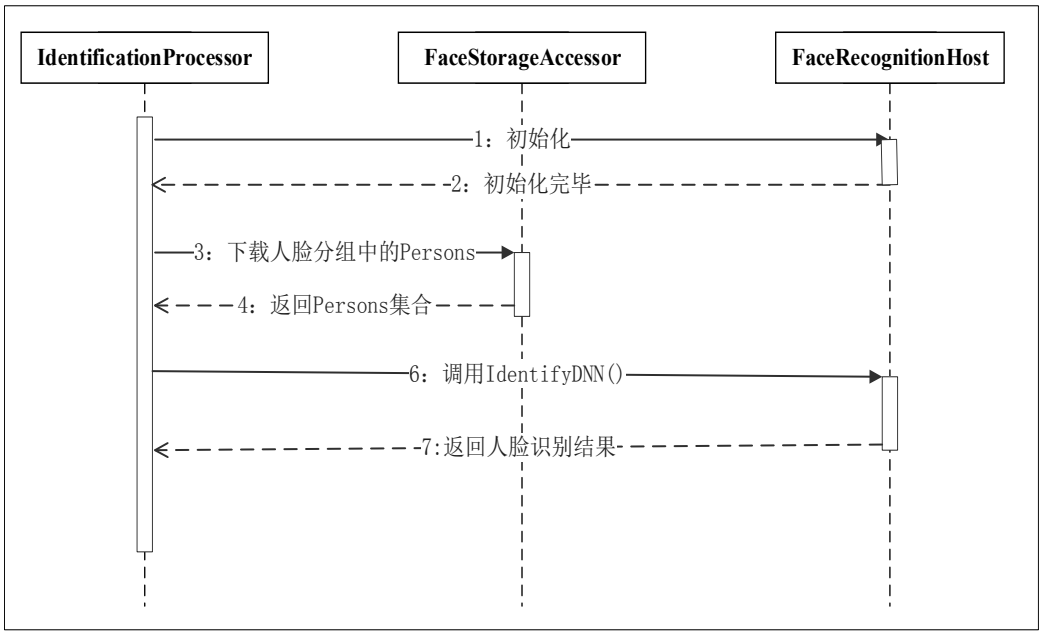

图 5-2 人脸识别 API 模块时序图

当人脸识别 API 模块被调用时，首先需要对用户的产品订阅标识(SubscriptionId)进行校验，从而判断调用是否合法。在合法的情况下，需要获取被识别的对象及其参考对象集数据，实际上就是要获取用户指定的被识别的人脸的特征集合和用户指定的人脸分组中所有人的特征脸数据集合（该集合是在 Azure 上已经训练完成的人脸分组数据）。通过调用人脸识别算法，将这两类数据进行比较，即可返回人脸识别结果集。模块具体活动图如图 5-3 所示。

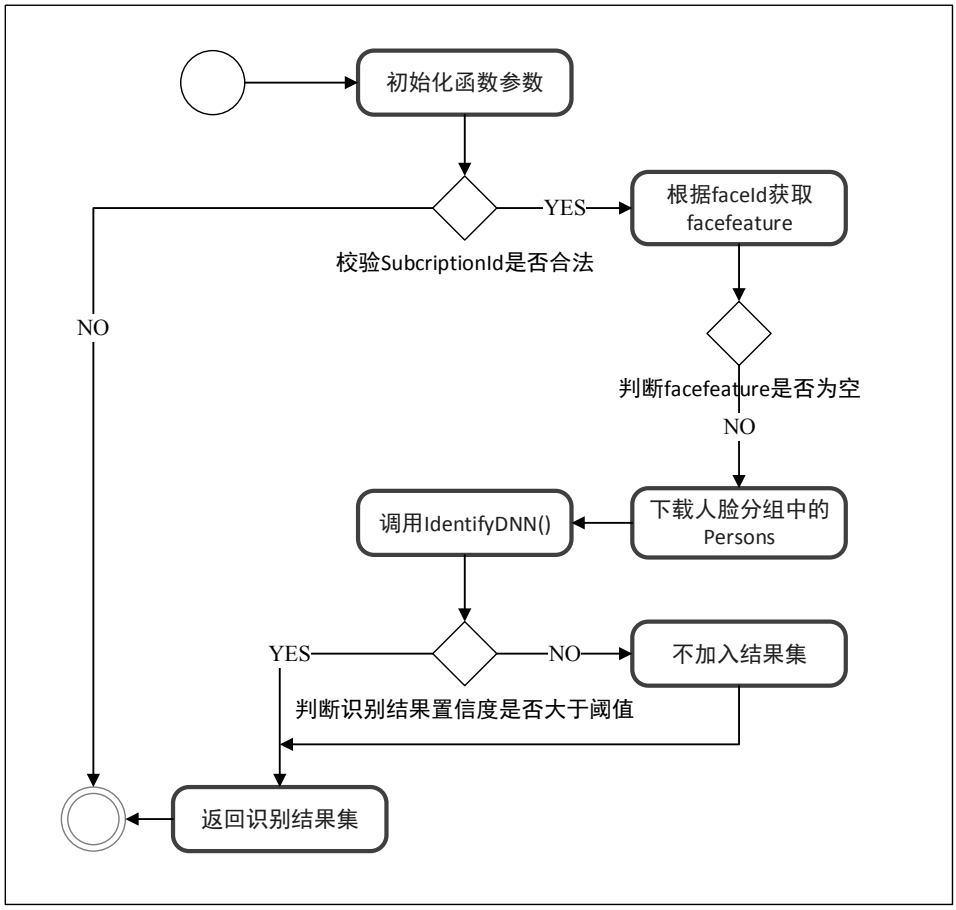

图 5-3 人脸识别 API 模块活动图

5.2.1.3 接口设计与实现

在设计人脸识别 API 接口时，有以下几个问题是我们需要注意的：

1) 本算法接口允许用户在同一个请求操作中，使用超过一张以上的人脸去做独立的识别操作，但是不能超过一定的数量，不然会影响接口的响应效果，目前规定为一次不能超过 10 张人脸。

2) 用户指定分组个人实体在隶属于某个人脸分组中，每个分组个人允许有一张以上的脸，但由于受到 Azure Table 存储中对实体大小的限制，其上限为 1MB，再结合算法中对每张人脸特征集合的存储大小，从而将分组个人实体的脸的数量限制在 248 以内。

3) 返回的结果中的候选人的数量，受限于用户指定要求返回的最多候选人数和相似置信度的阈值。如果没有人被识别出来，候选返回值将是一个空数组。

4) 除了应用在分组个人实体上，也可以将此 API 应用在脸这个概念上，用户可以从一个列的列表中识别出与输入的脸最相似的人脸，而不仅仅是从一个人脸分组中找到拥有相似的脸的人。

人脸识别 API 的接口的 URL 允许用户通过 POST 方法调用，当将 Header 信息中的 Content-Type 设置为 application/octet-stream 时，POST 操作的数据可以直接是 Binary 流。而当设置为 application/json 时，POST 数据可如下所示：

```
{
    "personGroupId" ： "group1",
    "faceIds" ： [
          "c5c24a82-6845-4031-9d5d-978df9175426",
          "65d083d4-9447-47d1-af30-b626144bf0fb"
     ],
    "maxNumOfCandidatesReturned"： 5,
    "confidenceThreshold"： 0.5
  }
```

表 5-1 和表 5-2 分别是对人脸识别 API 发起 Request 请求时和成功调用接口后返回的 Response 中的 JSON 的数据类型和内容进行设计和分析。

表 5-1 人脸识别接口 Request JSON 值表

数据	类型	描述
faceIds	Array	人脸可以通过 faceId 来独立标志，faceIds 表示分组中的所有 faceId。
personGroupId	String	personGroupId 标识了目标人脸群组
maxNumOfCandidatesReturned（optional）	Number	默认值为 1，目前 API 接口提供的值的设置范围为 1 到 10。
confidenceThreshold（optional）	Number	置信度阈值的设置，用来评判某张脸是不是属于某个人，范围为[0,1]。

表 5-2 人脸识别接口 Response JSON 值表

数据	类型	描述
faceId	String	被查询的人脸的 Id。
candidates	Array	返回的被识别的候选人，根据置信度排名。返回的结果大小不能比输入的 maxNumOfCandidatesReturned。如果没有人被识别出来，就返回一个空的数组。
personId	String	返回的被识别的人的 Id。
confidence	Number	人脸比较后的相似度，是一个浮点数，具体范围在 0.0 到 1.0 之间。

5.2.2 分组个人 API

5.2.2.1 类结构设计与实现

在前一小节的人脸识别 API 中我们提到了分组个人（Person）实体，本节将对与 Person 实体相关的 API 模块的实现展开描述。分组个人 API 模块主要功能是使得开发人员能够通过调用该 API 实现对 Person 实体和隶属于 Person 实体的人脸的管理。

以分组个人 API 中的新增人脸模块为例，它主要是由 PersonController、FaceStorgageAccessor、TableAccessor 和 DetectionAccessor 四个类构成，类图如图 5-4 所示。PersonController 类是分组个人 API 模块的 Person 对象的控制管理类，它为用户提供了对 Person 实体及其人脸进行新增、删除、修改、查看等具体操作的方法接口。而 PersonController 类的实现依赖于其他三个类。之前提到过，FaceStorgageAccessor 类主要实现了对 Auzre 上 Blob 数据的操作，与之相对应，TableAccessor 类则是主要实现了对 Azure 上 Table 中数据的操作。DetectionProcessor 类负责解析用户输入图像数据，比如图像 URL、图像流数据、目标探测人脸大小等。

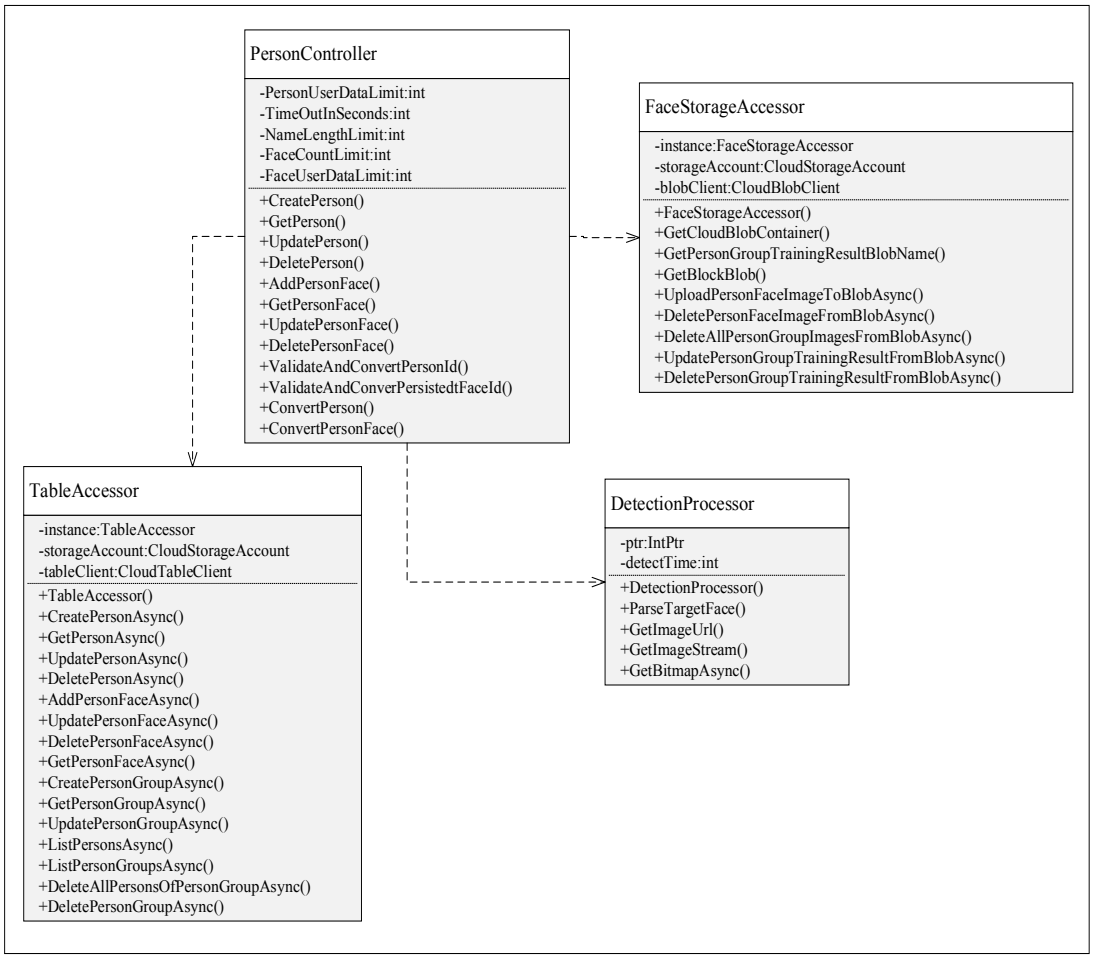

图 5-4 分组个人 API 中新增人脸模块类图

5.2.2.2 流程设计与实现

继续以分组个人 API 中的新增人脸模块为例，用户调用该模块 API 的目的是新增一张人脸到指定的 Person 实体中。该 API 调用入口为 PersonController 类，通过 DetectionProcessor 类提供的方法可以对用户调用 API 的信息进行处理，获得所需的目标人脸、图片 URL、图片流等数据信息。TableAccessor 类提供了方法确保程序获得 PersonGroup 集合，以及验证 PersonGroup 的状态值是否合法。FaceStorgageAccessor 类将用户指定新增的人脸图片流上传到 Azure 上的 Blob 中。当然，TableAccessor 类还负责将该人脸信息新增到 Azure 上的 Table 中。图 5-5 详细描绘了分组个人 API 中新增人脸模块的时序图。

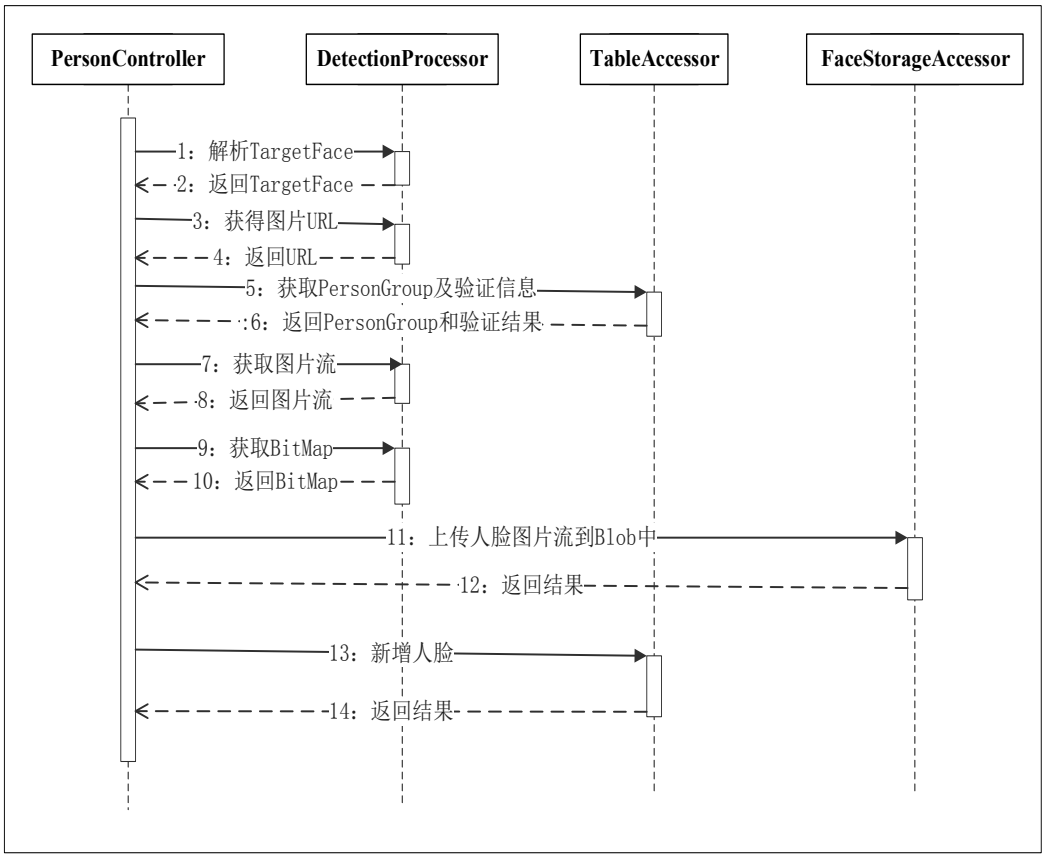

图 5-5 分组个人 API 中新增人脸模块时序图

当用户调用分组个人 API 中的新增人脸模块时，首先需要对用户的输入参数进行合法性校验。在合法的前提下，对用户输入的各种有效数据进行解析。由于是新增人脸，需要获取用户指定增加的人脸分组。将新增的人脸图片流数据上传到 Azure 上相关人脸分组的 Blob 中，再将新增人脸信息添加到 Auzre 的 Table 中。如果操作成功，返回成功响应信息；如果操作失败，则需要将上传到 Blob 中的数据删除，并返回失败响应信息。具体活动图如图 5-6 所示。

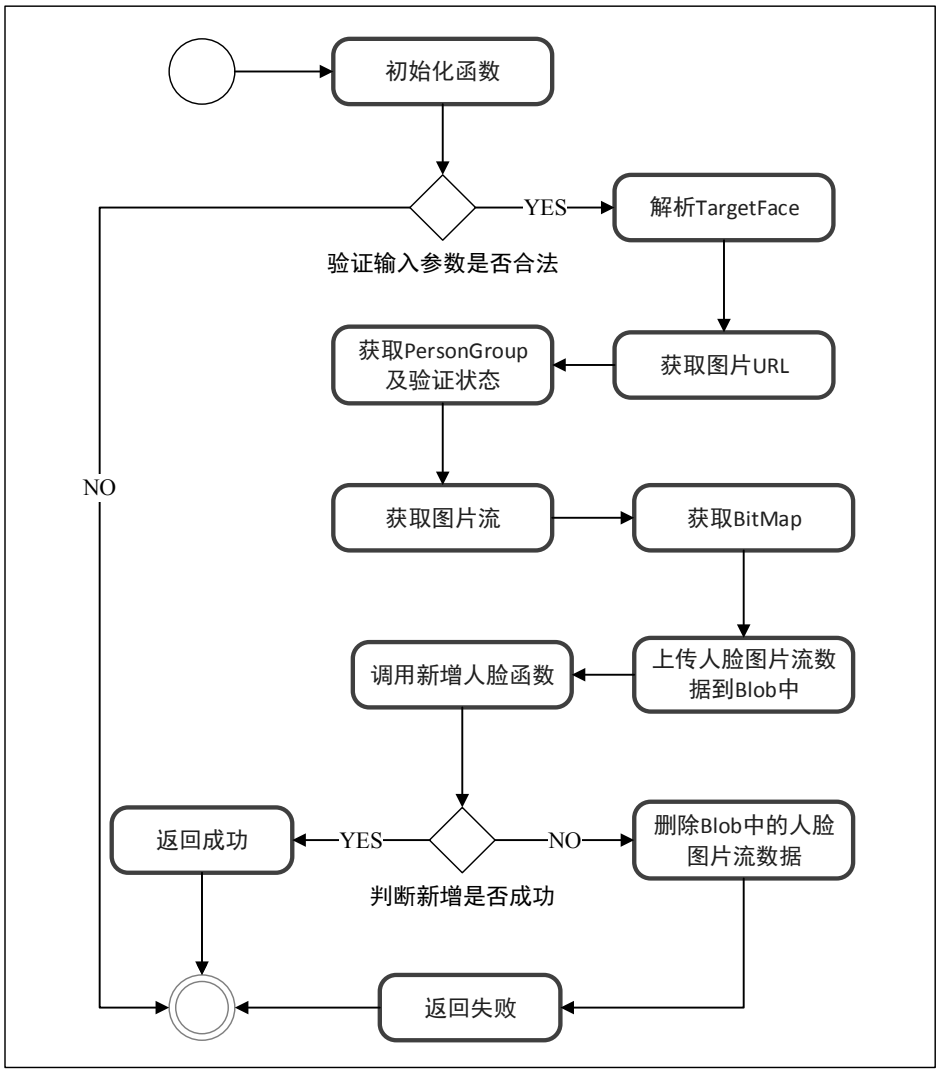

图 5-6 分组个人 API 中新增人脸模块活动图

5.2.2.3 接口设计与实现

可以根据 HTTP 协议通信方式的不同将分组个人 API 划分为四类：

1) HTTP/POST：主要有新增人脸的接口和创建人的接口。
2) HTTP/DELETE：与 POST 相对应，有删除人脸的接口和删除人的接口。
3) HTTP/GET：指的是获取人的信息的接口、获取人脸信息的接口以及显示分组中人的信息的接口。
4) HTTP/PATCH：是由更新人的信息的接口和更新人脸信息的接口组成。

以新增人脸为例，分析与设计相关的 API 的参数。对新增人脸接口的调用是为了实现将一张具体有一定代表性的脸添加到它隶属的 Person 实体中用于识

别功能。输入的人脸可以通过一个矩形方框标识出来作为目标人脸。调用此接口会返回一个 persisitedFaceId，它用来代表新增的这张脸，并且它是不会过期的。persistedFaceId 和 faceId 在本云服务平台中是不同的概念，persistedFaceId 代表的是添加到 Person 实体中的人脸的 ID，而 faceId 则表示的是图像中被探测到的人脸的 ID，前者不会过期，而后者会过期。

表 5-3 和表 5-4 分别是对新增人脸接口发起 Request 请求时和成功调用接口后返回的 Response 中 JSON 的数据类型和内容进行设计和分析。

表 5-3 新增人脸接口 Request JSON 值表

数据	类型	描述
personGroupId	String	personGroupId 标识了目标人脸分组
personId	String	personId 标识了人脸要被添加到的目标人
userData(optional)	String	表示用户自定义的关于新增人脸的信息，最大长度为 1KB
targetFace(optional)	String	用矩形框来标识被新增的人脸

表 5-4 新增人脸接口 Response JSON 值表

数据	类型	描述
persistedFaceId	String	persisitedFaceId 表示新增的脸，它是永久性数据，不会过期。不同与在 Face 实体中的 faceId 会过期，它会在检测和检测调用后的 24 小时之后过期。

5.2.3 人脸分组 API

5.2.3.1 类结构设计与实现

之前一直提到人脸分组（Person Group）的概念，下面将对与人脸分组（Person Group）实体相关的 API 模块的设计与实现进行介绍。人脸分组 API 主要设计目的是使开发人员能够通过调用该 API 实现对人脸分组实体相关信息的管理。

由于人脸分组中的新增、删除、修改等操作与人组个体是类似的，我们在此不再复述。下面，以人脸分组 API 中的训练分组模块为例来具体说明。训练分组模块主要是由 PersonGroupController、FaceStorgageAccessor、TableAccessor 三类构成，其类图如图 5-7 所示。PersonGroupController 类是人脸 API 模块的

PersonGroup 对象控制管理类，它为用户提供了对 Person Group 实体进行新增、删除、修改、训练等具体的方法接口。而 PersonGroupController 类的实现依赖于其他两个类。与分组个人 API 中一样，FaceStorgageAccessor 类和 TableAccessor 类分别实现了对 Auzre 上 Blob 中数据和 Table 中数据的操作。

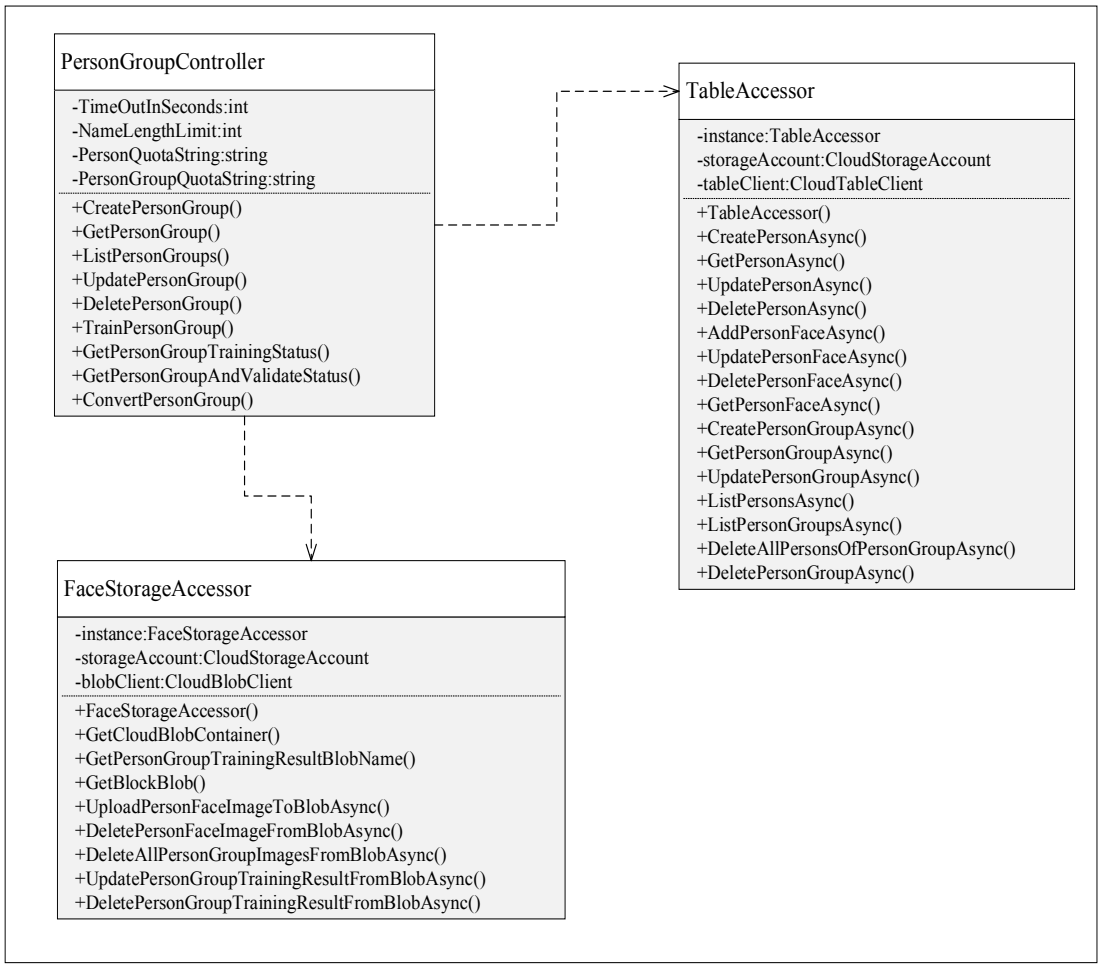

图 5-7 人脸分组 API 中的训练分组模块类图

5.2.3.2 流程设计与实现

继续以人脸分组 API 中的训练分组模块为例来说明处理流程。该模块的目的是实现分组数据的训练，流程相对比较简单明了，首先是通过 TableAccessor 类获取 PersonGroup 并验证其状态，然后再对训练方法调用并响应更新 PersonGroup 的训练状态。该模块的时序图可以参考图 5-8，活动图则如图 5-9 所示。

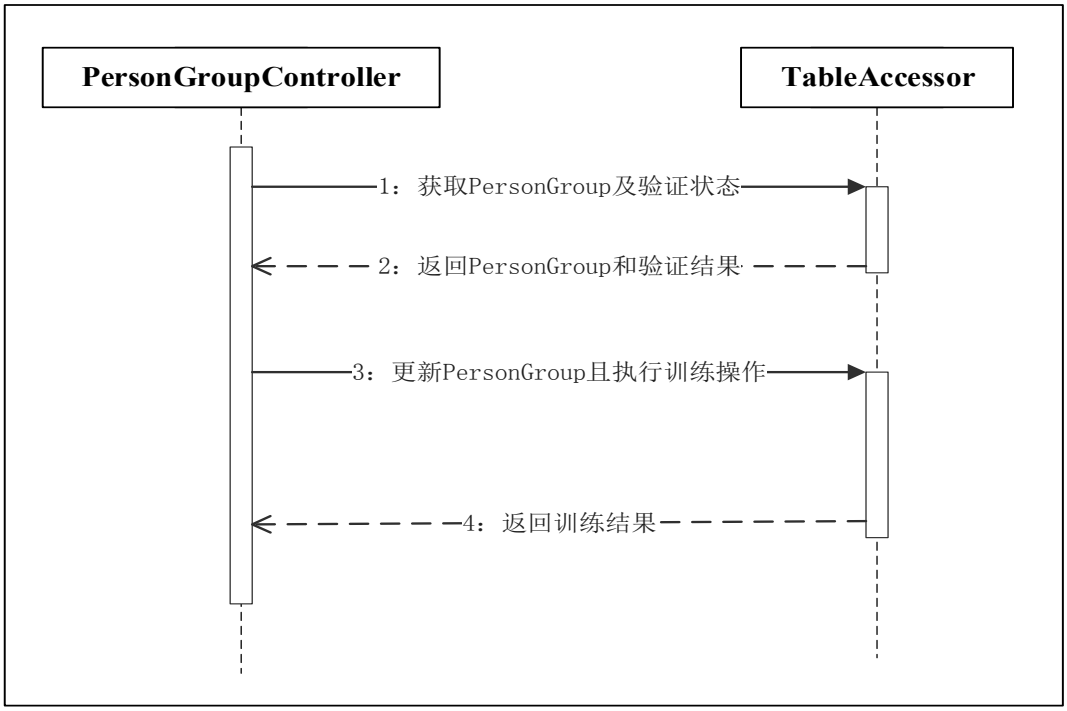

图 5-8 人脸分组 API 训练分组模块时序图

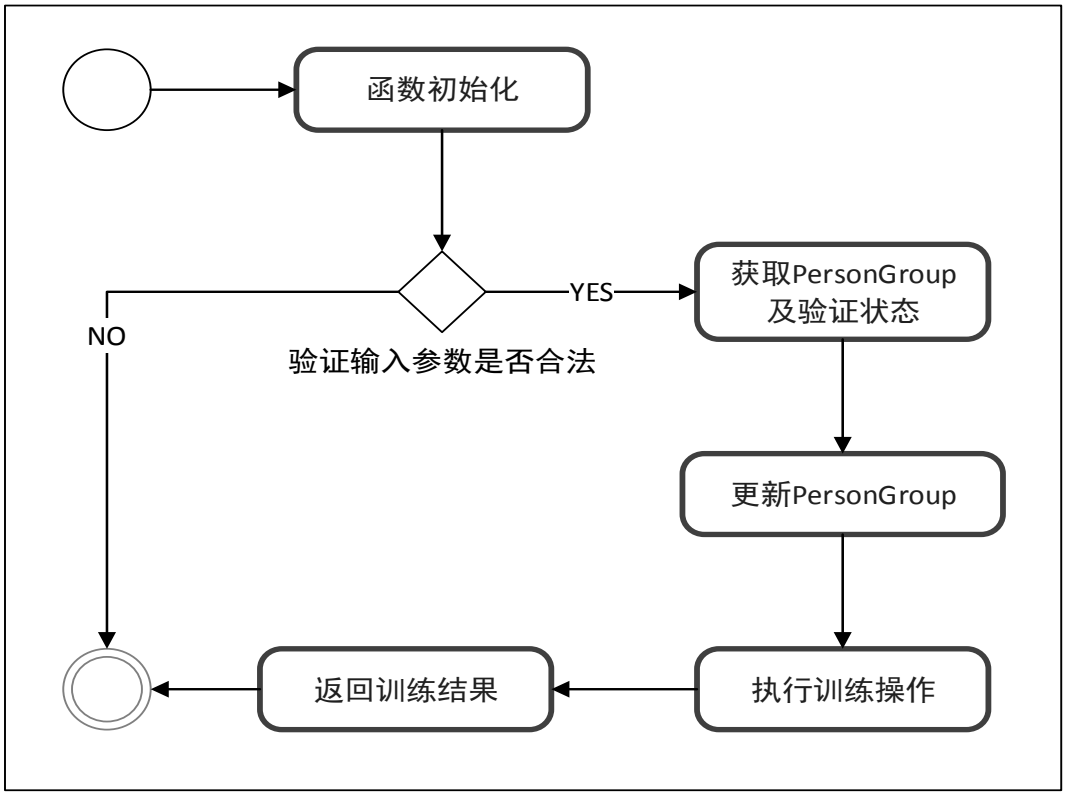

图 5-9 人脸分组 API 训练分组模块活动图

5.2.3.3 接口设计与实现

根据 HTTP 协议通信方式的不同可以将人脸分组 API 归纳为五类：

1) HTTP/PUT：主要是创建人脸分组接口。
2) HTTP/DELETE：与建人脸分组接口相对应，使用删除人脸分组接口。
3) HTTP/GET：包括获取分组信息接口、获取分组训练状态接口以及列表所有分组信息的接口。
4) HTTP/POST：在人脸分组中我们使用了训练分组数据的 POST 接口。
5) HTTP/PATCH：由更新群组信息接口组成。

这里提到的新增、删除、更新和列表接口与之前 Person 实体中的接口设计类似。仍以训练分组接口为例，它主要是提供给用户检索人脸分组完成或正在进行的训练状态。当用户调用训练分组接口时，训练这个操作会被触发，但是它不会被立即执行，平台会安排它被触发后在服务器端执行。目前平台中任何对更新分组接口的调用不会立即生效在具体的人脸数据上，只有当分组被训练后，才能看到更新变化。

表 5-5 是对训练分组接口发起 Request 请求时 JSON 的数据类型和内容进行的设计和分析。

表 5-5 训练分组接口 Request JSON 值表

数据	类型	描述
personGroupId	String	personGroupId 表示包括目标训练的人脸分组的 ID

用户对训练分组接口发起 Request 调用，若训练算法调用成功，则结果会返回一个空的响应集合；若执行失败，则会根据具体情形在 Response JSON 中返回对应的错误代号和错误信息。用户可以根据获取训练分组状态接口返回的信息来获取训练的状态，分为以下四种：

1) 未开始状态：训练过程等待执行，所以处于未执行的状态。
2) 执行中状态：训练动作在服务器上执行中。
3) 成功状态：成功，表示人脸分组已准备好可用于人脸识别操作中。
4) 失败状态：训练失败，通常是由于没有 person 存在或没有 persisted face 存在于人脸分组中。

5.3 人脸识别 API 的改进和优化

5.3.1 目前 API 服务的限制

前一节中介绍了本文云服务系统中与人脸识别算法相关的具体 API 的实现。在之前的设计中，将人脸分组中 Person 的上限设置为 1000，即表示在一个人脸分组中最多允许有 1000 个 Person 实体存在。这个设计无法满足现实生活中许多大规模的人脸库的识别需求，所以针对这一需求我们对原有的云服务进行了改进和优化。

大规模人脸识别是面向海量数据的人脸识别比对与搜索，它的实现需要保证能够快速检索，并且有高效的查询效率。如果将大规模的人脸识别的数据装入内存，再进行算法调用，这将要占用大量的内存并耗费巨大的计算资源。本项目的目标是对 API 进行改进，实现百万量级的人脸识别 API，但需能满足以下几个要求：

1) 百万量级的人脸识别速度需尽量控制在 2 秒以内。
2) 改进后在人脸库中的命中率与之前相比不能有太大的精度损失。
3) 人脸识别过程中本地处理的下载量也需要控制在一定的范围内。
4) 在实现过程中需考虑哪些操作可用并发实现，如图像的上传、下载等。
5) 选取性能最佳的参数
6) 实现百万级人脸识别的数据索引。

5.3.2 大规模人脸识别可行性论证

在实现大规模人脸识别算法之前，要通过实验论证其可行性。在有了前面内容铺垫的基础上，关注的重点是如何实现大规模情形下进行快速的图像索引。在低维空间，有许多高性能的所以结构可以拿来使用，比如 hash 索引、k-d 索引[37]和倒排索引等。但是人脸特征向量是高维向量，它包含了许多的脸部特征，并且各个特征是任意分布的，没有关联的。由于维数过高，可能会导致索引的性能变差，从而引发维数灾难[38]。针对大规模人脸识别的问题，本文借鉴 K 近邻搜索分类的原理，通过建立分类索引的方式对大规模人脸识别比对过程进行改进优化，并提出了相关参数的选择方法。具体实现过程从以下几个方面着手。

5.3.2.1 实验数据准备

准备实验所需的数据，实验要实现百万量级的人脸识别，目前平台并没有可用的百万量级的数据。为了论证实验的可行性，我们从已有的 8,000,000 张人脸开始着手，并从这些实验数据得出的结果去推导百万量级的实验可行性。

将准备好的实验人脸图像数据，利用 152 层的卷积神经网络（CNN）提取出 128 维的人脸特征向量[39]，用于标识每张人脸。通过对不同的人脸特征向量的比对，就可以计算出人脸的相似度，从而可以识别出不同的人脸归属。

选取特征脸，即人的代表性脸（Representive Face），也就是说在某个人有多张脸存在的情况下，通过计算脸与脸之间的距离矩阵，选出到其他脸平均距离最短的那个，作为代表性脸[40]。若平台中的人只拥有一张脸，则该脸直接被作为代表性脸，而无需计算。由于之前提到过，平台中的每张脸是使用 128 维的特征向量表示的，在计算脸与脸之间的距离中，我们使用到了相似度计算公式来度量人脸的相似度，其中也包括余弦相似度（Cosine Similarity）公式，如公式 5-1 和公式 5-2：

两个特征向量$a\ (x_{11},x_{12},\cdots,x_{1n})$，$b\ (x_{21},x_{22},\cdots,x_{2n})$ 之间的夹角余弦

$$\cos\ (\theta)\ =\frac{a\cdot b}{|a|\cdot|b|} \tag{5-1}$$

即

$$\cos\ (\theta)\ =\frac{\sum_{k=1}^{n}x_{1k}x_{2k}}{\sqrt{\sum_{k=1}^{n}x_{1k}^{2}}\sqrt{\sum_{k=1}^{n}x_{2k}^{2}}} \tag{5-2}$$

经过计算，最终提取出 256,000 个代表性脸可用于之后的实验中。在这 25.6 万的代表性脸中，约有 11 万是来源于只拥有一张脸的人，其余的 14 万的人拥有的脸数大于一张。

5.3.2.2 人脸距离分布计算

通过计算一些人脸的距离分布，来证明通过建立索引的方式大规模检索数据是可行的[41]。比如计算同一个人拥有的不同的脸之间的距离分布，用于之后的实验分析。图 5-10 和图 5-11 分别以折线图和饼图的形式表示了同一个人的不同脸之间距离的分布。其中图 5-10 将距离的每个区间的长度是 0.001，总范围为[0.0,1.0]，从而统计出距离落在 1000 个区间中的分布。图 5-10 表明，同一

个人的不同的脸是非常相似的。而图 5-11 表明 92%的同一个人的不同脸之间的距离是分布在[0.8,1.0]区间中的，这足以说明同一个人的不同的脸是相似的，也就间接说明，同一个人的不同的脸是可以被识别出来的。

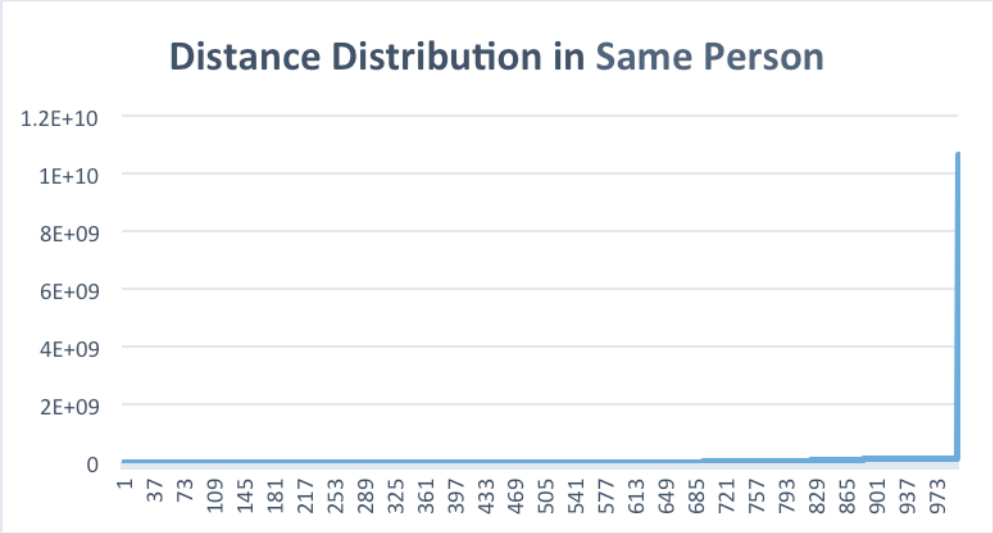

图 5-10 同一个人不同脸之间的距离分布图

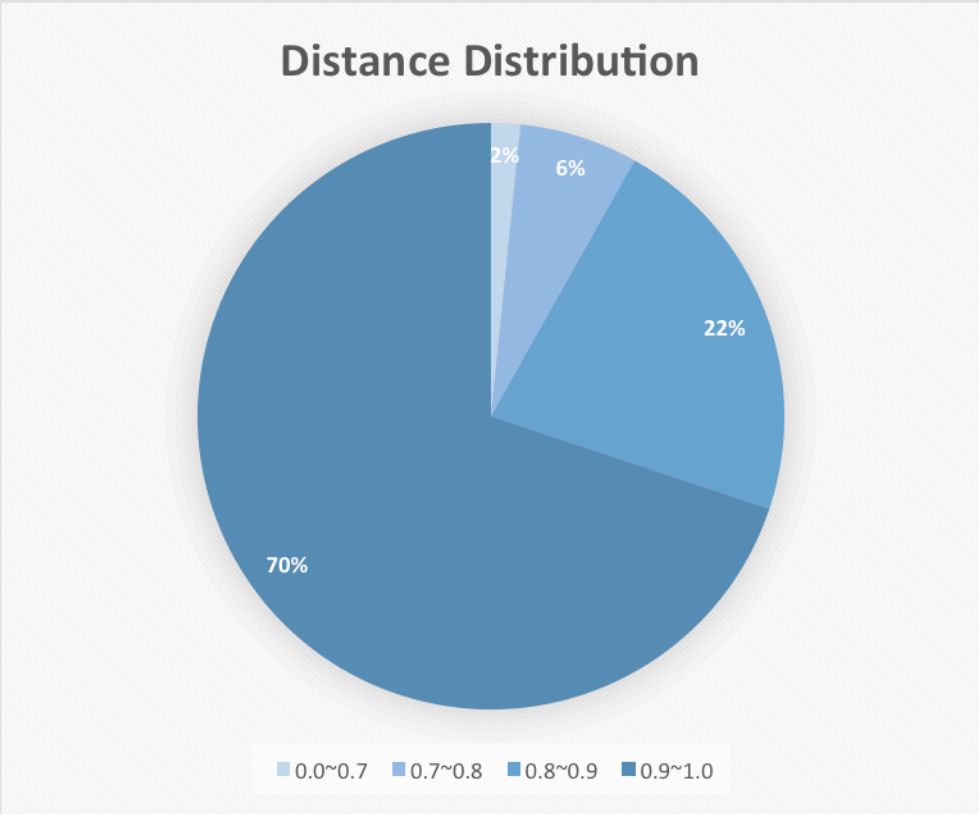

图 5-11 同一个人不同脸之间的距离分布图

图 5-12 则表示了不同人代表性脸之间的距离分布图，可以明显看出不同的人之间的脸的距离大多数分布在[0.3, 0.4] 的区间上，分布在两端的数据比较少，这是比较合理的情形，说明不同的人的脸是有区别的。

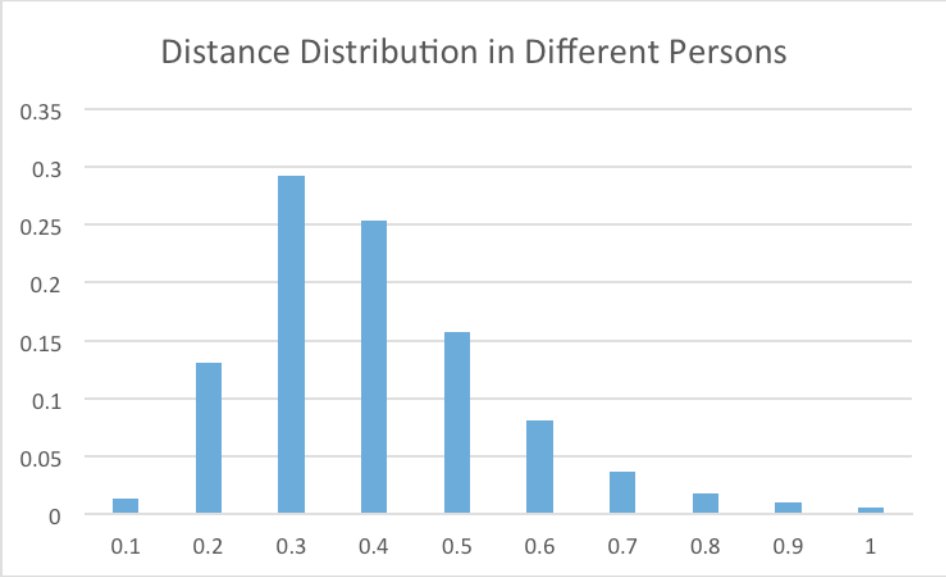

图 5-12 不同人的代表性脸的距离分布图

5.3.2.3 选取最佳性能参数

K 近邻检索(KNNQ)方式，是图像内容检索中常用的检索方式之一，它是根据用户给出的检索的条件结合相似度计算去图像库中查找最满足要求的数据，并按照计算出的相似程度的高低将结果返回给用户[42]。K 近邻检索方式返回的结果是与检索图像最相似的前 K 个数据，如公式 5-3：

$$KNNQ(q,K)=\left\{s_1,\cdots,s_k\subset DB\middle|\begin{matrix}\forall p\subset DB\backslash\{s_1,\cdots,s_k\}\\ ||s_i-q||\leq||p-q||,1\leq i\leq k\end{matrix}\right\} \quad (5\text{-}3)$$

用户可以根据实际情况指定需要返回的 K 的值。在本项目中，我们用到两个参数都是与 K 近检索相关的：Index K 和 Search K，分别用来表示建立索引的前 K 个值和某张脸进行人脸识别所得到的前 K 个搜索返回值。我们通过反复的实验比较，来确定平台能接受的最优的 K 值。

通过多次实验和对实验结果进行分析，选取出实验中性能最佳的实验参数，如表 5-6 所示：

表 5-6 实验参数表

参数	描述	参数值
Index Top K	被用来建立索引的匹配要求的前 K 个数据	5
Search Top K	人脸识别是用户需要搜索比对的前 K 个数据	20
Index Dataset	被选作建立索引参考的数据集合	10000
Threshold	人脸相似度阈值	0.5

具体的实验做法是：

1) 从 25 万多的代表性脸中，只拥有一张人脸的 11 万的人的集合中选取一定数量的数据集和作为建立索引参考样本集。
2) 使用拥有多于一张脸的 14 万人的代表性脸去建立索引，此处我们需要明确建立索引匹配的前 K 个数据的 K 值，即 Index Top K。
3) 由于 14 万张人脸来源于拥有多张脸的人的集合，选取 14 万人的非代表性脸来做识别搜索，比对返回的前 K 个值，即 Search Top K。
4) 实验比对在各种阈值（Threshold）限制情况下的人脸识别正确率。

在实验过程中，我们要在命中率精度损失较小的情形下，尽量使得 Index Top K 的值小，从而内存计算时的存储也会变小。同时 Search Top K 的值也不能特别大，如果它的值特别大，则在识别时需要下载计算的索引也会变大，使得响应时间变长，影响识别速度[43]。合适的参考数据集合能帮我们更好地实现算法，之前已经说明如何获取索引参考样本集，其大小也是我们关注的问题。经过反复实验和结果性能比较，最终筛选出最适合本云服务平台的参数的值。

5.3.3 大规模人脸识别 API

为了实现大规模的人脸识别 API，需要在之前实现的 API 基础上进行修改，主要涉及到索引的部分。本文项目使用 Azure Blob 存储中的 Blob 来存储和管理索引数据，用户主要是在调用新增人脸和删除人脸接口时，都需要对索引的 Blob 文件进行相关的操作。比如新增人脸的时候，需要对这张脸和索引参照数据集进行距离计算，然后返回 Index Top K 个数据，建立数据索引，将这张脸的 Person Id 放在返回的前 K 个索引的 Blob 文件中，从而完成索引的建立。删除人脸时，则会做相反的操作，将索引文件中与这张脸相关的 Person Id 删除。对于调用人

脸识别接口而言，并不会直接操作 Blob 数据，而是返回识别到的前 K 个数据，搜索着 K 个索引当中有没有存储我们需要识别的人，并按相似度返回。

5.3.3.1 类结构设计与实现

为了实现大规模人脸识别，主要使用到七个类，在原来的人脸识别基础上新增了五个类，分别是 MillionScaleSearch 类、TopKCompute 类、BlobAccessor 类、BlobItem 类和 AwaitQueue 类。其中，MillionScaleSerch 类的实现依赖于其他类。TopKCompute 类主要负责计算于索引相关的前 K 个值，它依赖于 FaceItem 类，并修改了 FaceItem 类，在其中新增了 TopKList 成员。而 BlobAccessor 类则关注索引信息在 Azure Blob 中的操作，它的实现依赖于 BlobItem 类和 AwaitQueue 类，分别提供了索引在 Blob 上的数据结构和对索引异步操作的队列管理。当然，还有 FaceRecognitionHost 类，它在之前的人脸识别 API 中就提到过，负责提供封装好的人脸识别算法接口。本文中与大规模识别相关的类图可参考图 5-13。

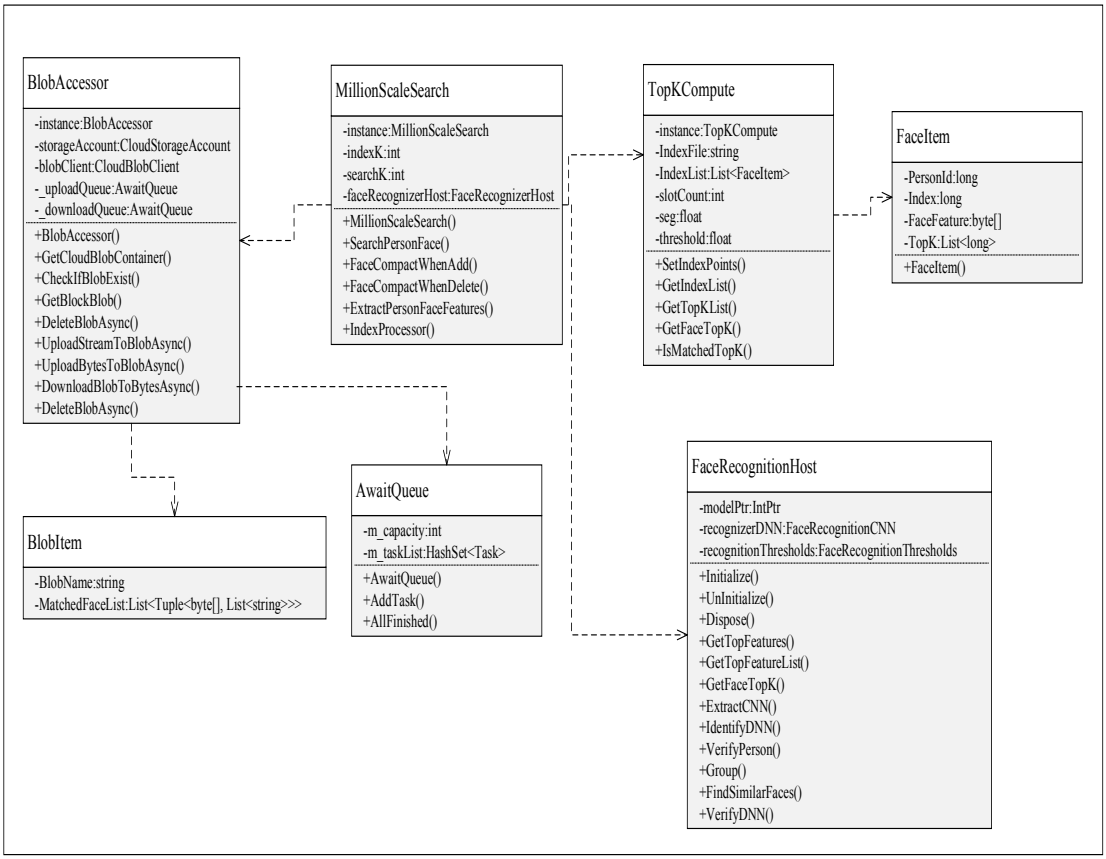

图 5-13 大规模人脸识别相关类图

5.3.3.2 流程设计与实现

大规模人脸识别 API 中新增了对索引文件的处理，流程设计也需相应调整。图 5-14 和图 5-15 分别是大规模情况下人脸识别 API 模块的时序图和活动图。

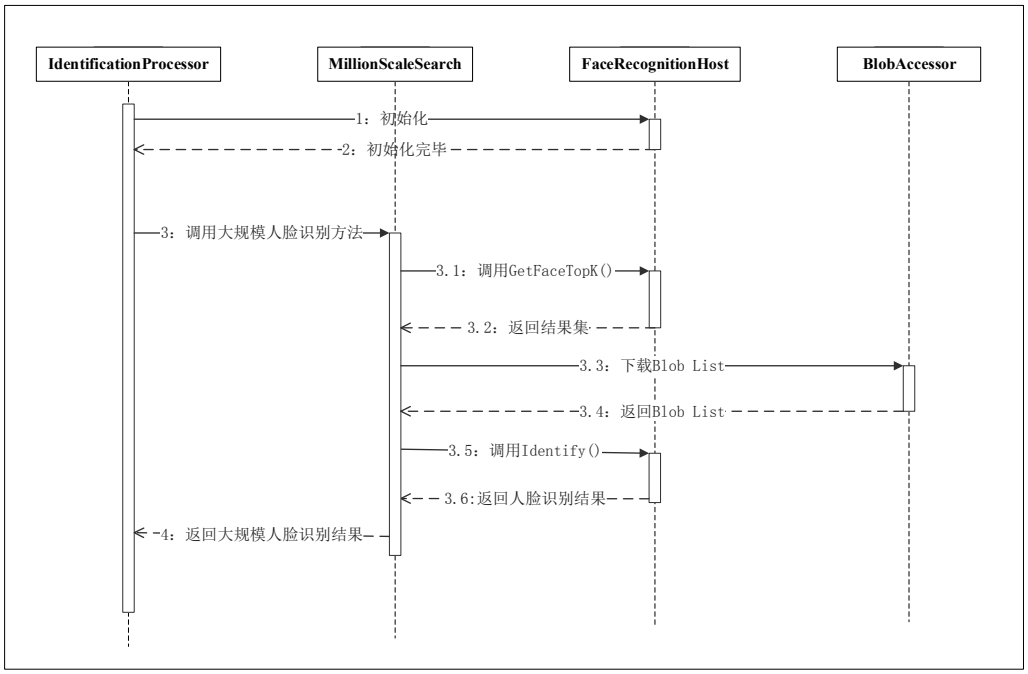

图 5-14 大规模人情况下脸识别 API 模块时序图

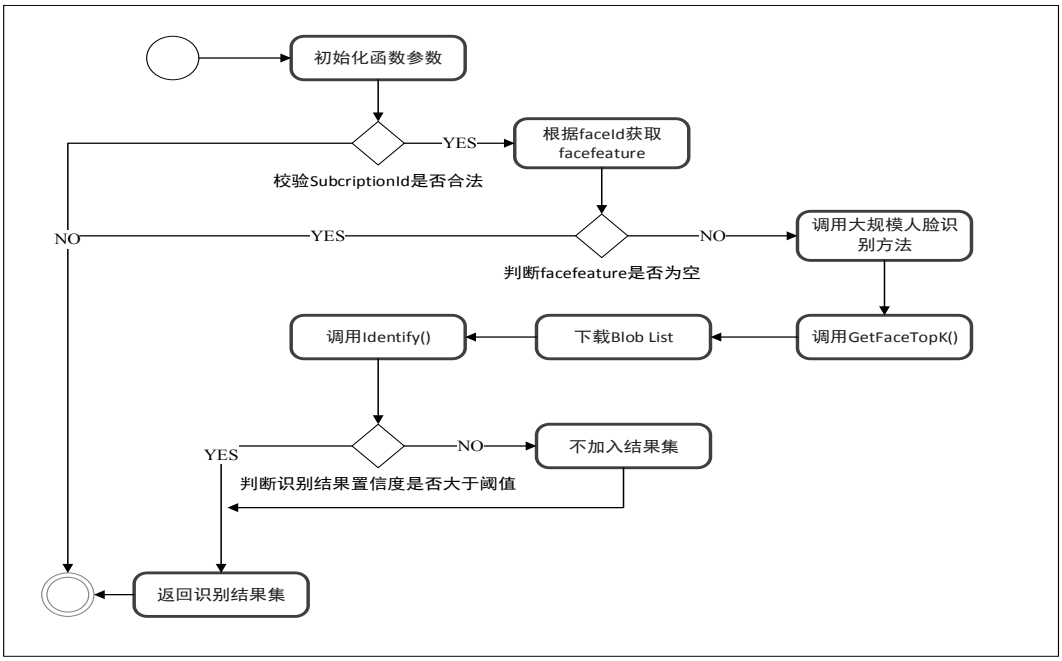

图 5-15 大规模情况下人脸识别 API 模块活动图

从上面两图中可以看出在大规模人脸识别流程中采用的是与 Azure Blob 存储中的索引文件比较的方法，来识别出与用户输入人脸最相似的结果集，其具体实现过程是比较复杂的。除了在识别的情况下需要操作索引 Blob 文件，在新增人脸和删除人脸时，也是需要对索引文件进行相应处理的。通过重新设计和实现接口中的特征脸合成方法，维护代表性特征脸集合，仅对发生变化的特征脸集合的数据进行索引相关操作。索引数据的及时更新可以确保人脸识别的正确性也省去了训练分组操作这一过程。具体可以参考图 5-16 和图 5-17，它们分别是大规模情况下新增人脸模块的时序图和活动图。

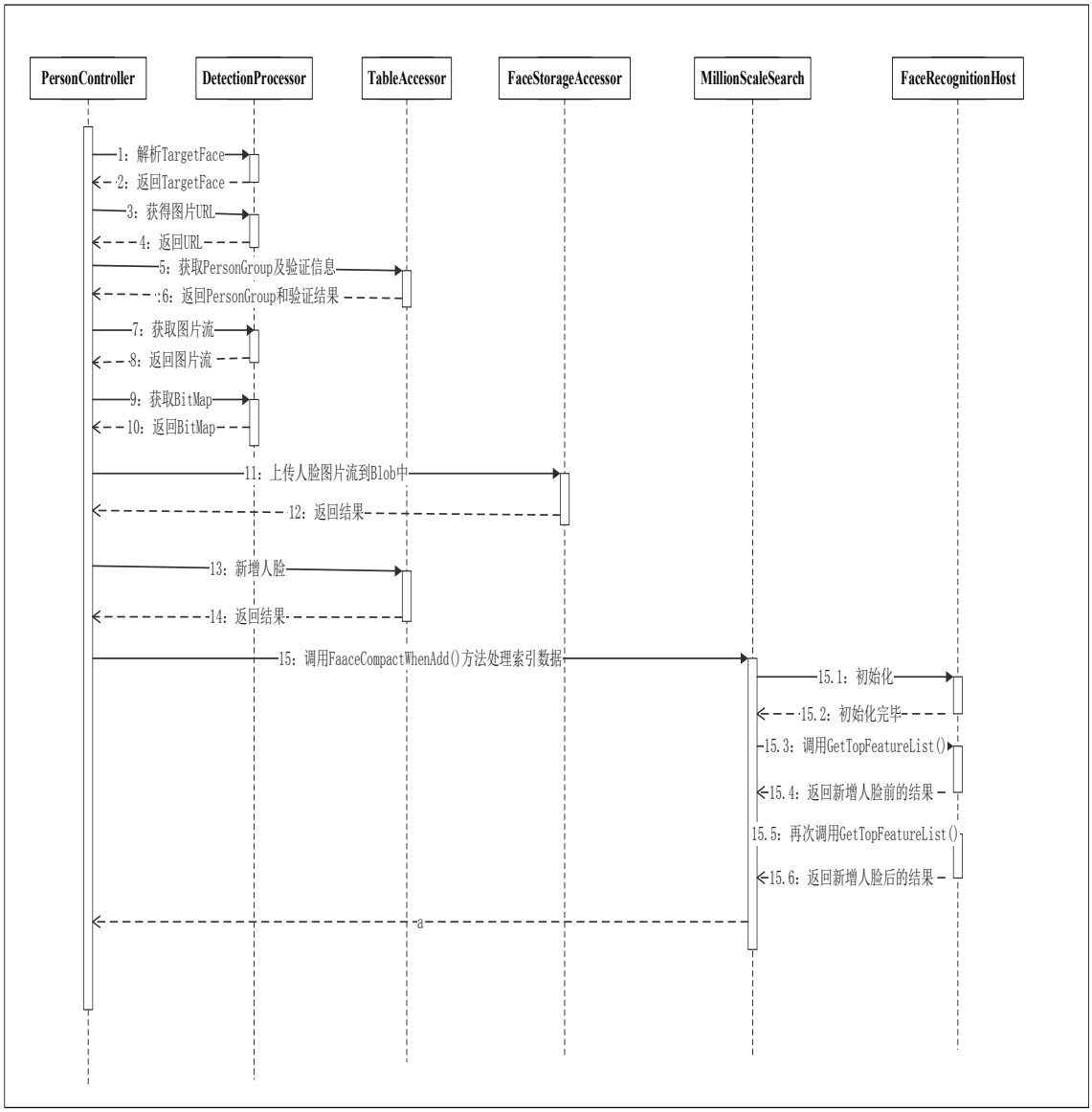

图 5-16 大规模情况下新增人脸模块时序图

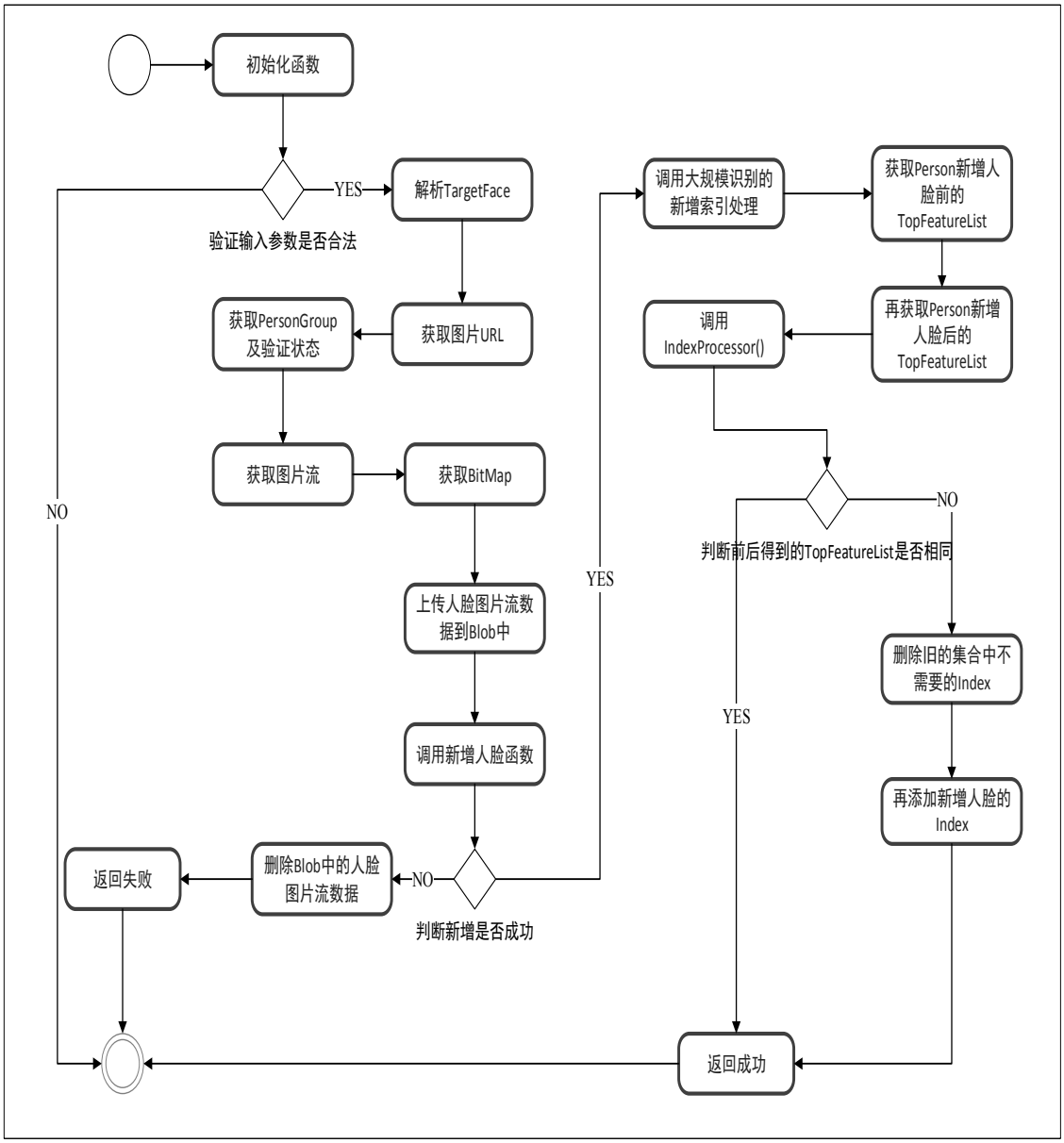

图 5-17 大规模条件下新增人脸模块活动图

5.3.3.3 接口设计与实现

大规模的人脸识别 API 需主要涉及到新增索引处理的部分与相关类和方法的变动。在面向用户的接口设计与实现细节上与前一节相比没有太大的变化，可以参考之前章节的介绍，在此不再对接口的设计与实现作重复的介绍。

5.4 本章小结

本章主要是介绍了人脸识别 API 的设计、实现与改进。首先，对本文项目中与人脸识别相关的 API 做了介绍，主要有人脸识别 API、分组个人 API 和人脸分组 API 这三种，分别从类结构、流程和接口三个角度介绍具体设计与实现。然后，分析目前云服务平台 API 服务的限制，主要关注的是大规模人脸识别服务，通过实验论证大规模人脸识别在基于 Auzre 的本云服务平台上的可行性。最后阐述了大规模人脸识别 API 的设计与实现，它是对之前介绍的 API 的改进与优化。大规模人脸库快速搜索的实现，在现实生活中可以应用于许多场景，比如公安系统、银行在线识别系统等。

第 6 章 平台的功能测试与性能分析

6.1 引言

本文项目基于 Azure 平台，完成了人脸识别云服务平台的设计、实现和优化。通过云服务平台的实现，将计算量大的人脸识别算法以服务的形式提供给开发人员。使开发人员能够将算法用在自己的应用程序的开发中，从而实现了算法即服务的功能。

6.2 平台的功能测试

6.2.1 API 功能测试

Azure 平台支持各种语言的 SDK，本文项目中采用的是.NET 环境下的 SDK，以 Azure 平台为基础架构整个项目。本文云服务平台部署所采用的软件环境如表 6-1 所示。

表 6-1 云服务平台部署软件环境表

类型	内容
操作系统	Windows 10 Enterprise
运行环境	Visual Studio 2015
平台基础	Microsoft Azure
开发语言	C#
Web 开发框架	ASP.NET MVC
建模工具	Windows Office Visio 2013

用户在使用云服务平台 API 时，首先需提供身份信息，在本文项目中唯一标识用户身份有效的信息指的是用户订阅产品标识。在通过平台认证用户身份信息合法后，用户可以开始调用需要的算法方法，在调用之前平台会对用户调用方法的参数进行验证。当参数也满足要求合法的情况后，才会顺利进入到算法实现的过程中。算法运算完成之后，并不会直接把运算结果返回给用户，平

台还会对算法运算得到的结果数据进行转化。平台会依据从请求通信中得到的数据信息，反馈给应用程序满足其反馈数据信息要求的格式，并同时序列化反馈的数据格式。只有将响应数据成功反馈给用户，才表明接口被成功响应调用。若调用失败，API 接口则会反馈给用户平台定义的错误码和错误信息，帮助用户找到错误的原因，方便用户的重新调用。

本文平台测试的主要对象是接口，目的是检测云服务平台与外部系统以及云服务平台内部之间的交互点，重点测试检查数据交互、控制管理过程以及测试平台间的依赖关系。为此，本文主要采用以下三种接口测试方案

- API 逻辑测试：依据 API 业务逻辑来设计测试用例，目标是测试在正常输入的情况下是否能得到正确的响应结果，设计测试用例的方法跟黑盒测试类似，主要用到两种方法，分别是等价类方法和边界值方法。
- API 出错处理：逻辑测试的对象是正常逻辑，即保证云服务平台对外部提供的接口服务可以正常工作，但这种测试方法不保证数据的安全，以及程序在非正常情况下的逻辑正确性。从而需要测试出错处理，主要包括以下几个方面：输入空值测试，比如在传入对象参数时，需要进行 NULL 值的测试；参数属性测试，测试输入一个未赋值参数场景；异常测试，制造异常使用场景，对异常描述是否清晰等。
- 路径测试：以上两种方法使得单个的接口服务正常提供得到了保证，但还需测试能否在业务流中满足各种业务的需求。路径测试方法就是通过设计尽可能少的测试用例，来保证数据在云服务平台的各种业务场景下都是安全可靠的。

通过使用以上三种测试方法，对本文云服务平台提供的接口服务进行了比较完整的测试，再确保单个接口正常提供服务的前提下，也保证了多个接口合作场景下可以正常工作。表 6-2 列出了部分接口功能测试用例表。

表 6-2 部分接口功能测试用例表

用例序号	用例描述	预期测试结果	实际测试结果
1	调用人脸识别接口	在接口调用输入正确的情况下，返回用户指定数目的人脸识别结果	正常
2	调用新建人脸接口	在接口调用输入正确的情况下，返回新增人脸成功响应结果	正常
3	调用删除人脸接口	在接口调用输入正确的情况下，返回删除人脸成功响应结果	正常
4	调用列表显示人脸接口	在接口调用输入正确的情况下，返回指定 Person 的人脸集合	正常
5	调用训练分组接口	在接口调用输入正确的情况下，返回训练分组成功响应结果	正常
6	调用获取分组训练状态接口	在接口调用输入正确的情况下，返回分组的训练状态	正常
7	调用大规模情况下新增人脸接口	在接口调用输入正确的情况下，返回大规模情况下新增人脸成功的响应结果。同时保证在新增人脸时，索引 Blob 中该 Person 的索引数据正确。	正常
8	调用大规模情况下删除人脸接口	在接口调用输入正确的情况下，返回大规模情况下删除人脸成功的响应结果。同时保证在删除人脸时，索引 Blob 中该 Person 的索引数据正确。。	正常
9	调用大规模情况下人脸识别接口	在接口调用输入正确的情况下，快速返回大规模情况下用户指定数目的人脸识别结果。	正常

6.2.2 API 应用效果

本文平台提供给用户多平台的 API 接口服务，用户可以通过对 API 接口的调用，满足其应用上对人脸识别算法的需求，在互联网的任何角落快速方便实现人脸识别的功能。下面我们分别以 WPF （Windows Presentation Foundation）应用程序和 IOS 应用程序为例，展示使用人脸识别云服务平台实现的应用效果。

首先是在 WPF 应用程序中的实现效果。图 6-1 显示了通过调用 API 服务建立人脸分组，并管理人脸分组中的分组个人的效果。图中左侧显示了在人脸分组 Group1 中有三个 Person 实体。

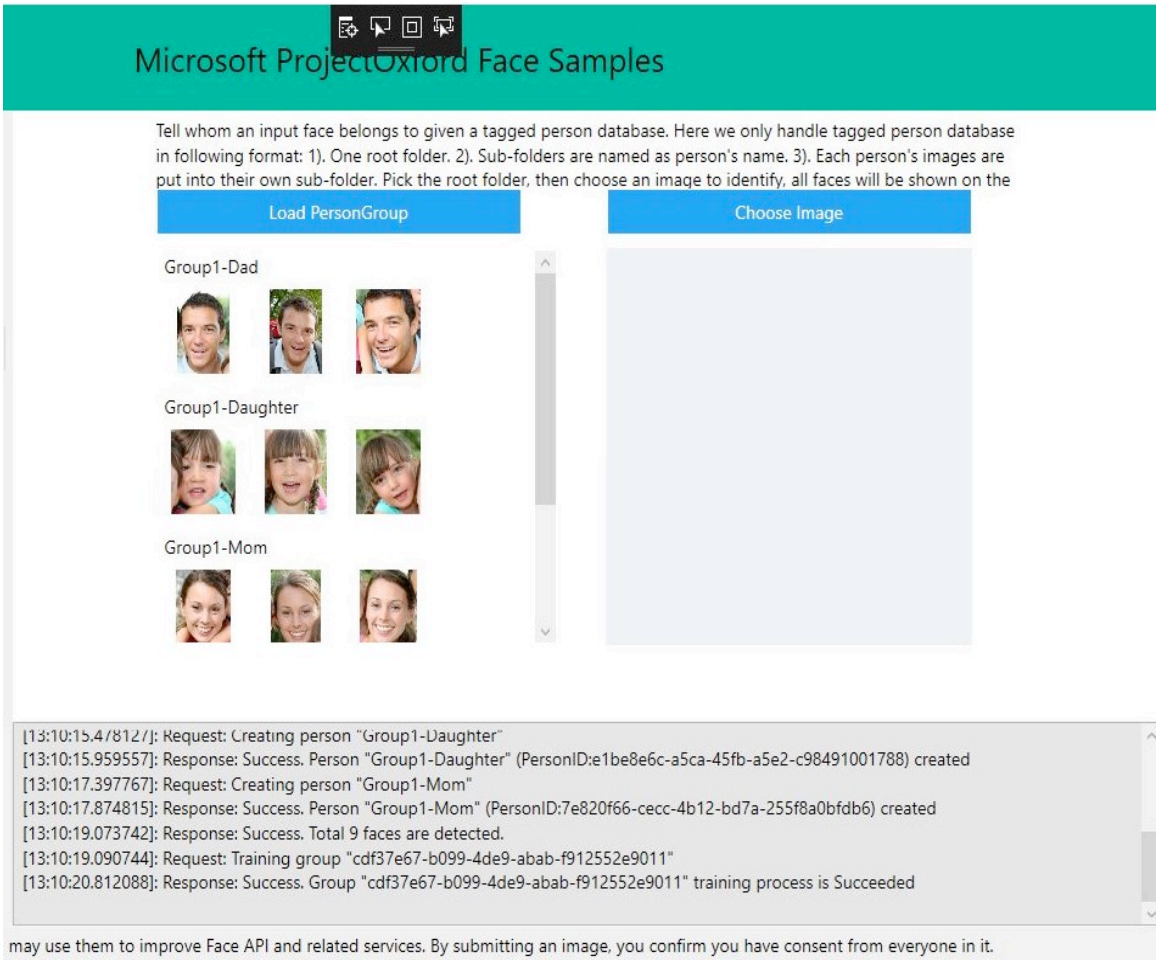

图 6-1 WPF 应用程序效果图 1

图 6-2 表示 WPF 程序调用人脸识别 API 接口在人脸分组 Group1 中实现的人脸识别效果图。如图所示，右侧矩形框中标出了被识别的人脸是属于左边列表显示的 Group1 中的女儿。

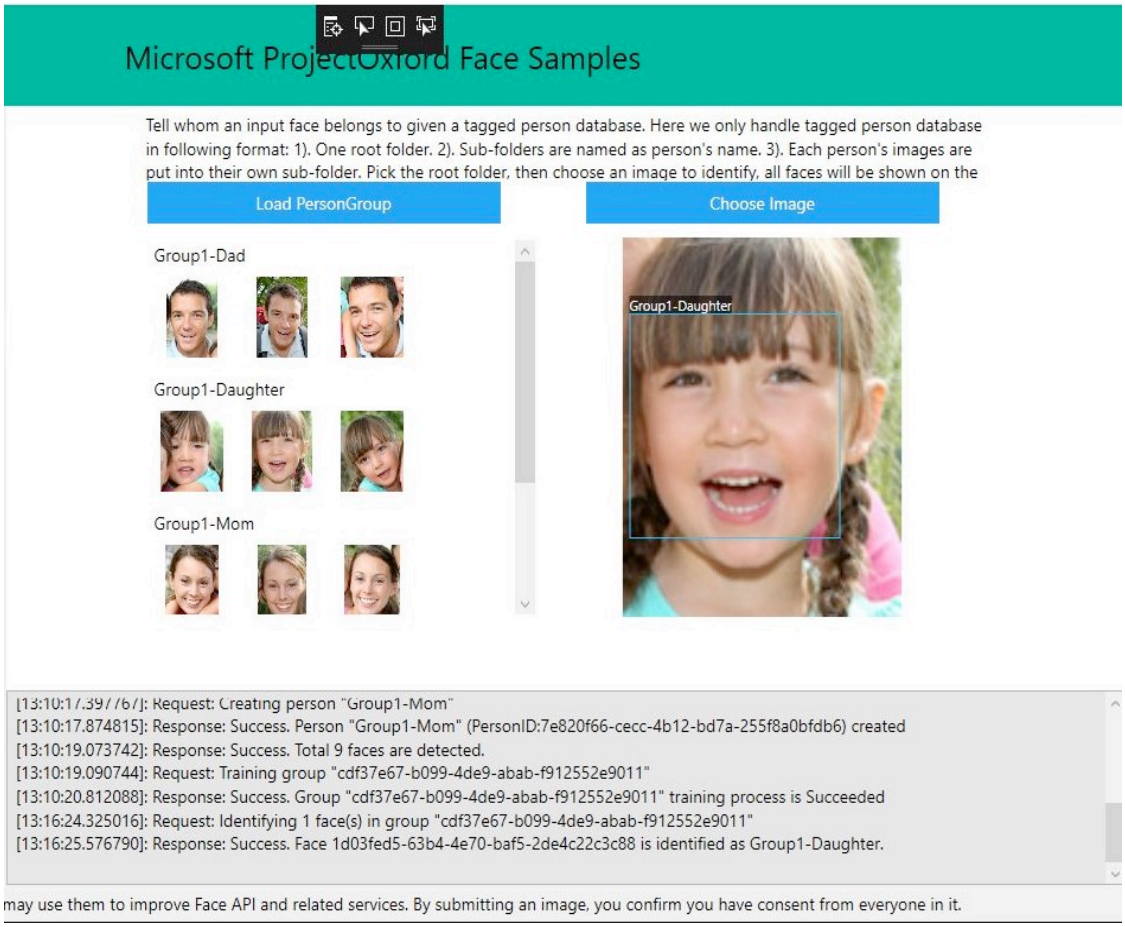

图 6-2 WPF 应用程序效果图 2

图 6-3 显示了在 IOS 应用程序中的实现效果，主要展示了调用人脸识别 API 接口实现的人脸识别的功能的结果。左图是我们用用户分组 Star 中已经存在的用户杨幂的一张已经添加的脸去进行人脸识别。可以看出返回的用户结果集中，用户杨幂的相似度最高，且高达 99%。而右图显示的则是用一张用户分组 Star 中不存在的用户杨幂的脸去进行人脸识别，可以看出返回的用户结果集中，用户杨幂的相似度虽然没有左图中显示的那么高，但是也是在相似度中排名第一的数据结果。

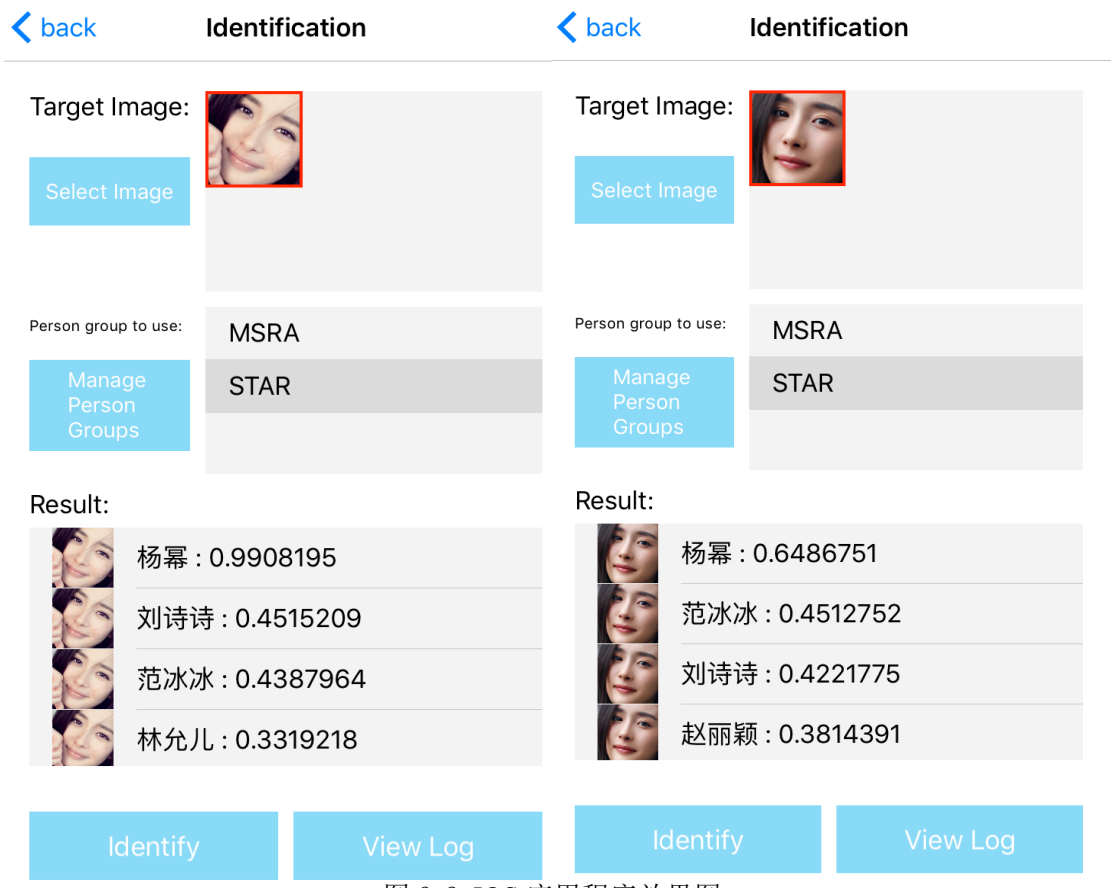

图 6-3 IOS 应用程序效果图

通过以上 API 接口调用的例子，可以看出本云服务平台提供的人脸识别服务可以被开发人员顺利地应用在自己的应用程序开发之中，并且开发人员可以根据需求定制自己的人脸识别应用。

6.3 平台的性能分析

6.3.1 平台参数性能分析

在云服务平台的实现过程中有很多环节需要综合实际情况，经过实验和论证选择出最合适的参数来进行下一步的平台搭建。以大规模人脸识别服务的实现为例，实现这个算法服务，我们需要找出合适的参数如 Index Top K， Search Top K 和 Index Dataset Number 等的值，从而使算法服务更好的在云平台上以

API 接口的形式开发给用户使用。通过多次实验，分析实验结果，选取性能最好的参数值。我们要明确以下几个问题：

1) 索引参考数据集的选择：在上一章节中我们介绍过如何筛选出最合适的索引参考数据集，在明确筛选规则后，我们需选择合适数据集的大小。
2) Index Top K 值的选择：Index Top K 主要会影响 Storage 中的 Blob 数据信息的存储，K 值越大，则占用的 Storage 越大，同时会导致运算时内存中需要处理的数据也越大，所以需要选择尽量小的 Index Top K 值。
3) Search Top K 值的选择：Search Top K 的值影响的是客户端的响应时间和用户的网络流量。在本平台服务中的策略是通过增大 Search Top K 的值来降低 Index Top K 的值。但需注意，也不能将其增的过大，若 Search Top K 值过大，则会影响用户调用 API 的响应时间。
4) Threshold 值的选择：Threshold 是用来作为相似度比对的阈值，主要是用来返回识别结果集给用户，它的大小直接影响了前 K 个返回值的排序方式。在平系统平台中主要是在 0.5 和 0.6 这两个值中选择，根据之前的实验结果表示在这两个阈值下的命中率较高。

明确了以上的问题后，就可以在对影响实验参数条件限定的情形下，将参数值以不同的组合进行实验，分析实验结果选择出性能最好的参数组合。表 6-3 详细展示了参数组合实验的结果，第一列条件组合列，是将 Index Dataset 大小与 Index Top K 和 Search Top K 的值组合；第二列命中率 A 表示的是在 Threshlod 值为 0.5 的情况下的命中率；第三列命中率 B 表示的是在 Threshlod 值为 0.6 的情况下的命中率。

将表 6-3 中的实验参数数据组合绘制成曲线图 6-4，其中图中蓝色曲线表示在 Threshold 为 0.5 的情况下，在各种条件组合下的人脸识别命中率变化趋势；而红色曲线则表示在 Threshold 为 0.6 的情况下，在各种条件组合下的人脸识别命中率变化趋势。

通过分析表 6-3 和图 6-4，可以得知 Threshold 值为 0.5 的情况下，条件组合值为 10000vs10vs10 和 10000vs5vs20 的这两个组合的命中率较高。考虑到我们之前提到的参数需求问题，我们最终选择的条件组合是 10000vs5vs20，即索引参考集的大小为 10000，Index Top K 的值为 5，Seach Top K 的值为 20，Threshold 的值为 0.5。这组参数是本云服务平台选出的性能最好的参数组合。

表 6-3 大规模实验参数值表

条件组合 Dataset vs Index K vs Search K	命中率 A Threshold = 0.5	命中率 B Threshold = 0.6
10000 vs 10 vs 10	0.983231	0.981840
10000 vs 5 vs 20	0.979834	0.978436
20000 vs 10 vs 10	0.974994	0.973516
20000 vs 5 vs 20	0.972243	0.970560
50000 vs 10 vs 10	0.959697	0.957829
50000 vs 5 vs 20	0.956665	0.954726

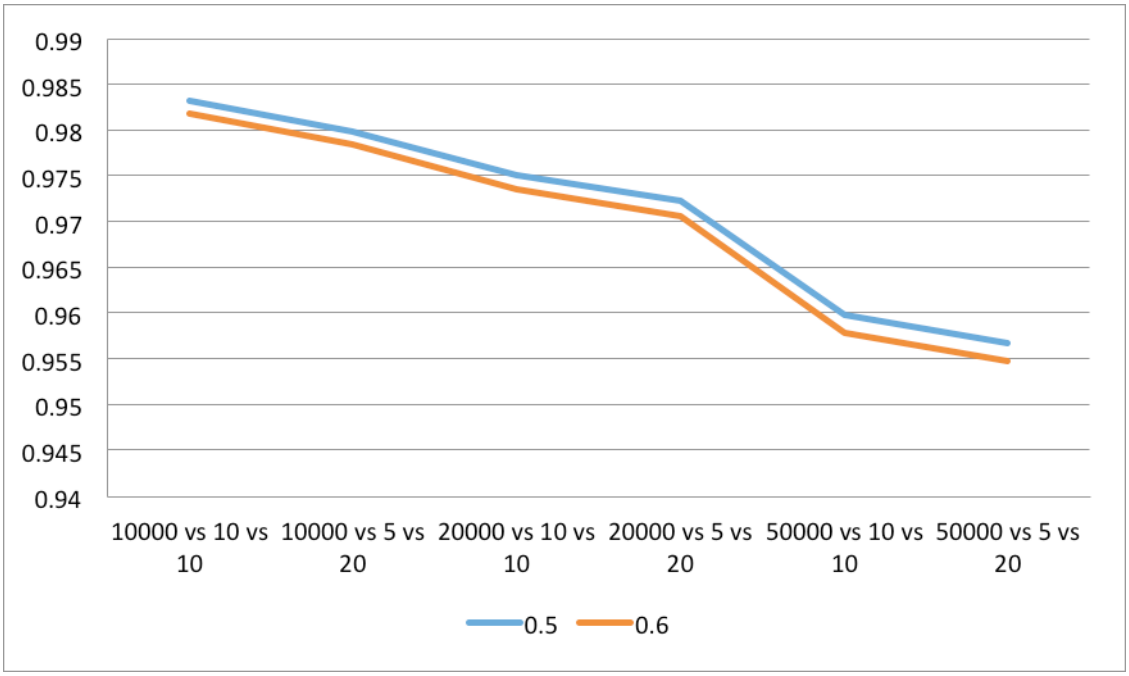

图 6-4 大规模实验参数曲线图

6.3.2 平台服务性能分析

云服务平台性能优越性体现在平台服务被用户使用的各种实际的应用场景中，在此就不一一复述。仅以大规模人脸识别服务为例，表 6-4 展示了在同样数量级（以 1000 为例，之前平台中人脸分组中最多有 1000 的 Person）的人脸分组中，使用人脸识别服务的响应时间比对。可以看出由于在新增和删除人脸的时候增加了对索引 Blob 的操作，所以改进后的响应时间比之前的略慢；但是在人脸识别的响应时间得到了明显的缩减，改进后的响应时间比之前快了很多。

表 6-4 大规模响应时间比较

接口调用	改进后的时间（ms）	改进前的时间（ms）
新增人脸	3690.75	2543.60
删除人脸	2364.93	2182.73
人脸识别	529.70	45370.26

从时间上的比较可以看出，大规模人脸识别在识别速度上与之前的人脸识别相比快了很多，但是这种方法的采用也间接导致识别正确率的折损。所以，我们还需要比较大规模人结果的命中率精确度与直接在人脸库中搜索人脸的命中率折损率，看看是否在平台的误差接受范围内。

首先，对实验中所有的不同的脸之间的距离进行了计算，统计出它们在[0.0,1.0]范围中每 0.1 大小区间出现的频率。然后，对能够被顺利识别的人脸做统计，同样计算出它们在[0.0,1.0] 范围中每 0.1 大小区间出现的频率。如表 6-5 所示，将相同区间内的频率放在一起比较，从而可以看出相同区间使用索引方法的人脸识别的命中率的情况。通过观察表 6-5，发现在云服务平台使用大规模人脸识别方法跟之前比较存在轻微的命中率损失。

接下来，对表 6-5 中得到的数据进行分析，计算大规模人脸识别在累加区间与普通人脸识别的命中率，得出表 6-6。

表 6-5 不同区间命中频率对比表

区间	大规模识别命中频率	普通识别命中频率
0.0 - 0.1	248974	248979
0.1 - 0.2	2335310	2338981
0.2 - 0.3	1671860	1689620
0.3 - 0.4	310223	336736
0.4 - 0.5	39607	53486
0.5 - 0.6	4486	7883
0.6 - 0.7	411	2541
0.7 - 0.8	23	4298
0.8 - 0.9	0	1416
0.9 - 1.0	0	5297

表 6-6 累加区间命中率表

累加区间	命中率
0.0 - 0.1	0.999979918
0.0 - 0.2	0.998579576
0.0 - 0.3	0.994988755
0.0 - 0.4	0.989608644
0.0 - 0.5	0.986754365
0.0 - 0.6	0.986050172
0.0 - 0.7	0.985602448
0.0 - 0.8	0.984702695
0.0 - 0.9	0.984405009
0.0 - 1.0	0.983293018

最后，将表 6-6 中的数据转换成分布图 6-5，可以清晰看出在 [0.0,1.0] 范围内的 10 个累加区间的命中率分布，通过图中的分布曲线我们可以得知采用索引方式的大规模人脸识别的命中率与原先的命中率相比精确度损失非常小，但是时间上却缩短了太多，表明我们可以通过这一方法来实现大规模人脸识别服务。

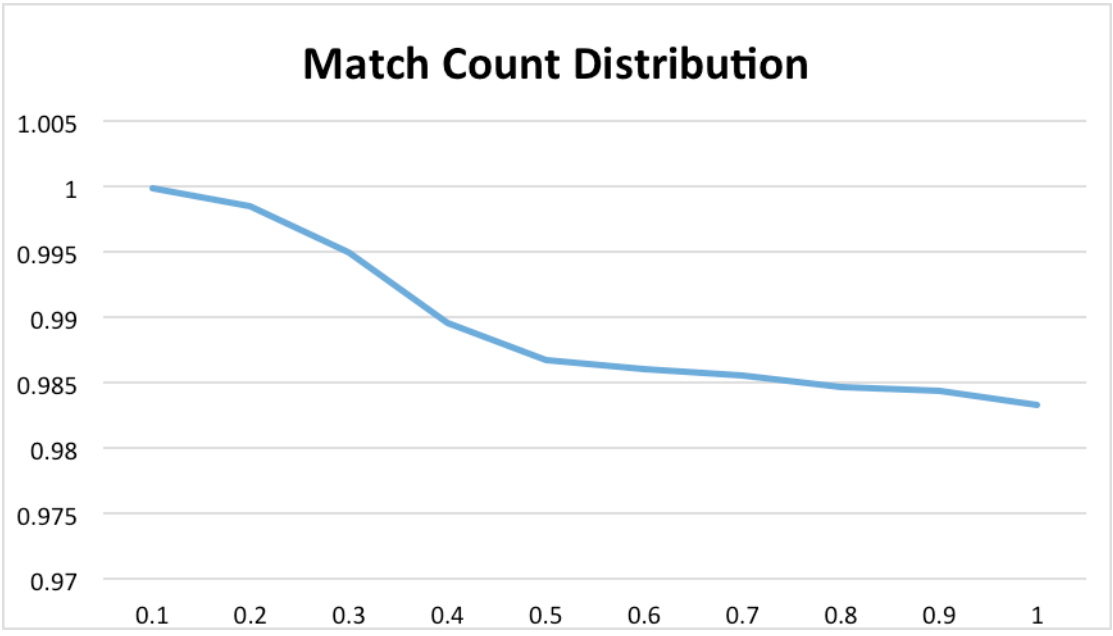

图 6-5 大规模识别命中率分布图

6.4 本章小结

本章主要是对云服务平台进行功能测试和性能分析。首先，对平台提供的服务进行功能测试，主要是针对 API 的测试。之后，以 WPF 平台和 IOS 平台上的应用为例，展示了云服务平台提供的服务应用效果。除了对云服务平台进行了功能上的测试，还从性能上对平台进行了分析，包括平台参数性能和平台服务性能两方面。通过分析，可以看出本云服务平台通过选取性能最佳的参数来实现各项服务功能，同时它的服务性能也足以满足用户对云服务平台的需求

第 7 章 总结与展望

7.1 论文总结

本论文主要是基于 Microsoft Azure 的云平台设计与实现人脸识别算法的云服务平台。Azure 云平台为本项目提供了基础设施和服务平台，同时提供了相应的云计算、云存储等服务。这些都为本文平台提供了极大的技术支持，在此基础上，采用松耦合的模块化设计方式，并结合算法的可复用、可扩展的特性，本项目实现了人脸识别的云服务平台。在具体的设计与实现中，本论文主要完成了以下工作：

1) 使用 Azure 云平台存储管理数据：数据对平台服务而言是关键因素之一。我们通过 Azure 平台上的提供云存储服务，使用其 Table Storage 和 Blob Storage 的服务，来实现在数据不断增长的情况下保证数据一致性并实现有效的数据扩展。
2) 设计与实现基于 Azure 云平台的人脸识别 API：主要是将人脸识别算法在云平台上以服务的方式提供给用户，从而用户可以遵循 HTTP 协议来调用所需的接口，将人脸识别算法应用到用户自己的各种应用程序中。
3) 大规模人脸识别的设计与实现：通过实验论证大规模人脸识别算法在平台上的可行性，主要是通过建立人脸索引，在确保识别正确率没有受到太大的影响下，兼顾请求响应时间等因素，反复实验选出性能最佳的参数值。通过设计和实现大规模人脸识别 API 接口，优化了用户在大规模人脸库中发起请求的响应时间，提高了识别效率。
4) 提供多种算法云服务接口：通过云平台提供算法服务，用户只需通过调用算法接口，仅用较少的网络流量即可获取多种复杂的算法服务。并且在互联网任意角落通过自己的终端定制跟自己平台相匹配的应用程序。

7.2 进一步工作和展望

本文实现的基于云平台的人脸识别云服务平台，在 Azure 平台的基础上有效实现了算法资源分配、数据存储、API 接口服务以及大规模人脸识别等功能。对于算法在云平台上的应用以及用户通过互联网在自己的应用程序中实现复杂的算法计算都是非常有意义的。当然，目前的云服务平台也是存在可以进一步优化的地方，比如以下两项：

1) 由于使用了 Azure 平台提供的云存储服务对平台数据进行存储和管理。这表明用户在对算法调用时，需要频繁的从云存储中上传、下载和查询数据，这些都是通过网络数据传输实现的，对云平台的数据存储和传输造成一定压力，从而增加了服务请求的响应时间，影响了服务效率。可以考虑在之后的工作中优化算法和数据结构来解决相应问题。
2) Azure 云服务负责在云平台的 VM 上运行算法实现所需要的计算节点，通过 Azure 平台可以实现对计算节点的管理和配置，但在具体的计算任务的分配问题上，目前并没有采取措施实现最有效的负载均衡。

在未来的研究中，我们希望能够在云平台上实现更多的与人脸识别算法服务相关的功能，比如人脸图像的分类、人脸信息的统计分析等功能。此外，在算法扩展这一方面，希望能够找到更方便的算法配置管理方式，比如使用文件配置等，来实现算法的快速有效的扩展。这些都是我们在之后的研究中会关注的点，目的是使云平台更好地为客户服务，从而使算法得到更多利用和推广。

参考文献

[1] Jain, Anil K, Stan Z Li. Handbook of face recognition[M]. New York: springer, 2011:19-49

[2] Yang G, Huang T S. Human face detection in a complex background[J]. Pattern Recognition, 1994, 27(1):53-63

[3] Abbas A, Khalil M I, Abdel-Hay S, et al. Expression and illumination invariant preprocessing technique for Face Recognition[C]// International Conference on Computer Engineering & Systems. IEEE, 2008:59-64

[4] Missbach M, Stelzel J, Gardiner C, et al. A Short History of Cloud Computing[M]// SAP on the Cloud. Springer Berlin Heidelberg, 2013:1-13

[5] Mell P, Grance T. The NIST definition of cloud computing[J]. Communications of the Acm, 2011, 53(6):50-50

[6] Metz R. Cloud Computing Explained.[J]. Educause Quarterly, 2010, 33(2):13

[7] 冯海超. Windows Azure:微软押上未来[J]. 互联网周刊, 2012(9):60-63

[8] 罗达强. 探析 Windows Azure Platform 微软云计算平台[J]. 硅谷, 2010(16):9-10

[9] Weinhardt C, Anandasivam A, Blau B, et al. Business Models in the Service World[J]. It Professional, 2009, 11(2):28-33

[10] 祝秀萍, 吴学毅, 刘文峰. 人脸识别综述与展望[J]. 计算机与信息技术, 2008(4)：53-56

[11] Tistarelli M, Bicego M, Grosso E. Dynamic face recognition: From human to machine vision[J]. Image & Vision Computing, 2009, 27(3):222-232

[12] Turk M.A, Pentland A.P. Face recognition using eigenface[J]. Proc.ieee Conf.computer Vision & Pattern Recognition, 2011, volume 84(9):586-591

[13] 张翠平, 苏光大. 人脸识别技术综述[J]. 中国图象图形学报, 2000, 5(11):885-894

[14] 耿艳萍. 浅谈人脸识别技术及其应用[J]. 科学之友旬刊, 2011(7):131-133

[15] Redkar T, Guidici T. Windows Azure Platform[J]. Springer, Berlin, 2012:1-47

[16] WECHSLER, Harry, et al. Face recognition: From theory to applications[J]. Springer Science & Business Media, 2012:51-72

[17] Dobrea D M, Maxim D, Ceparu S. A face recognition system based on a Kinect sensor and Windows Azure cloud technology[C]// International Symposium on Signals, Circuits and Systems. 2013:1-4

[18] Copeland M, Soh J, Puca A, et al. Microsoft Azure and Cloud Computing[M]// Microsoft Azure. Apress, 2015:3-26

[19] Bohn R B, Messina J, Liu F, et al. NIST Cloud Computing Reference Architecture[C]// IEEE World Congress on Services. IEEE Computer Society, 2011:594-596

[20] Grossman R L, Gu Y, Sabala M, et al. Compute and Storage Clouds Using Wide Area High Performance Networks[J]. Future Generation Computer Systems, 2009, 25(2):179-183

[21] Luo J Z, Jin J H, Song A B, et al. Cloud computing:architecture and key technologies[J]. Journal on Communications, 2011, 32(7):3-21

[22] Wilder B. Cloud Architecture Patterns: Using Microsoft Azure[J]. Oreilly & Assoc Inc, 2012:15-35

[23] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. International Journal of Computers & Technology, 2013, 4(4):50-58

[24] Calder B, Wang J, Ogus A, et al. Windows Azure Storage: a highly available cloud storage service with strong consistency[C]// ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October. 2011:143-157

[25] Hill Z, Li J, Mao M, et al. Early observations on the performance of Windows Azure[J]. Scientific Programming, 2011, 19(2-3):121-132

[26] 仇新红. 对云计算与云存储技术研究的探讨[J]. 电子技术与软件工程, 2015(21):31

[27] 杨娜. 云计算与云存储技术研究[J]. 黑龙江科学, 2014(12):234-234

[28] Bjørner N, Jayaraman K. Checking Cloud Contracts in Microsoft Azure[M]// Distributed Computing and Internet Technology. Springer International Publishing, 2015:21-32

[29] Masse M. REST API Design Rulebook[J]. 2011:23-65

[30] Pautasso C, Wilde E. RESTful web services:principles, patterns, emerging technologies[C]// International Conference on World Wide Web. ACM, 2010:1359-1360

[31] Pautasso C, Zimmermann O, Leymann F. RESTful Web Services vs. "Big" Web Services - Making the Right Architectural Decisions[C]// International Conference on World Wide Web, WWW 2008, Beijing, China, April. 2008:805-814

[32] Jon Galloway, Phil Haack, Brad Wilson, et al. Professional ASP.NET MVC 4[J]. Wiley & Sons, 2014(3):1-2

[33] 刘晓玲, 陈云海, 林立宇等. 人脸识别服务云计算化技术方案分析[J]. 广东通信技术, 2016(1):14-17

[34] Tsai W T, Sun X, Balasooriya J. Service-Oriented Cloud Computing Architecture[C]// Seventh International Conference on Information Technology: New Generations. IEEE, 2010:684-689

[35] Fielding R T. Architectural styles and the design of network-based software architectures[C]// University of California, Irvine, 2000:303

[36] Kurtz J. ASP.NET MVC 4 and the Web API: Building a REST Service from Start to Finish[J]. 2013

[37] Friedman J H. An Algorithm for Finding Best Matches in Logarithmic Expected Time[J]. Acm Transactions on Mathematical Software, 1977, 3(3):209-226

[38] Schwartz W R, Guo H, Choi J, et al. Face identification using large feature sets.[J]. IEEE Transactions on Image Processing A Publication of the IEEE Signal Processing Society, 2012, 21(4):2245-2255

[39] Rizzo G, Troncy, Rapha. NERD: a framework for unifying named entity recognition and disambiguation extraction tools[C]// Demonstrations at the, Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2012:73-76

[40] Wagner A, Wright J, Ganesh A, et al. Toward a Practical Face Recognition System: Robust Alignment and Illumination by Sparse Representation[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2012, 34(2):372-386

[41] Cox D, Pinto N. Beyond simple features: A large-scale feature search approach to unconstrained face recognition[C]// IEEE International Conference on Automatic Face & Gesture Recognition and Workshops. 2011:8-15

[42] 胡香娟. 基于云计算的海量人脸特征图像大规模对比技术[J]. 科技通报, 2013, 29(2):154-156

[43] Taigman Y, Yang M, Ranzato M, et al. Web-scale training for face identification[J]. 2015:2746-2754

致谢

时光荏苒，在中国科学技术大学的求学时光即将结束。回顾在科大的丰富多彩的校园生活，历历在目。对于我个人而言，不仅在学识水平上有了提升，同时也结识了许多优秀的老师和同学，这都是我人生中非常宝贵的经历。为此，我非常感谢科大，正是由于老师和同学们的帮助，才让我在软件工程领域学的更加扎实。与此同时，我也要感谢微软亚洲研究院为我提供这一难得的实习机会，在 MSRA 实习期间，我感受到了活泼的工作环境和严谨的研究作风，学到了很多在学校无法学习到的技能。

在完成本次毕业设计以及论文撰写期间，我的校内导师叶勇副教授和企业导师 MSRA 高级工程师王超都对我提供了无私的帮助。叶勇老师从开题报告开始就在百忙之中抽空对我的项目进行了详尽而又明确的指导，他严谨的工作态度深深感染了我。与此同时，本篇论文也离不开 MSRA 高级工程师王超的倾力指导，从云服务平台的设计架构到功能实现细节，王老师都会耐心地引导我，给我指出正确的方向。在这段过程中我对于软件系统设计的理解得到了新的提升，也学会了很多为人处事的道理，这些是都与老师们的辛勤教导是分不开。

此外，我也要感谢 MSRA Face Team 的同事们以及任俊涛、林萍萍、王璨、吴凯琳、林镇安等同学，他们曾在项目遇到瓶颈的时候或给予我指导和鼓励，帮助我一起攻坚克难，项目最后的完整实现离不开他们的帮助。

最后，我还要感谢我的父母和家人，在项目实现以及论文撰写的过程中，他们一直默默地支持着我，每当我遇到瓶颈，他们给予无私的爱和鼓励，帮助我顺利完成了这篇论文。

论文修改说明

对于专家一提出问题的修改说明：

1. 作者所做的工作主要是云服务平台API的设计实现，未涉及平台人脸识别的核心算法，论文题目用“云服务平台”的设计实现，题目偏大。

说明：参考专家修改意见，现将题目更改为：基于云平台的人脸识别服务的设计与实现。

2. 论文内容与 Project Oxford 人脸识别项目的工作内容基本相同，不知作者跟这个项目是什么关系。

说明：实习所在单位为微软亚洲研究院，参与项目为Project Oxford，现已更名为微软认知服务人脸识别。

对于专家二提出问题的修改说明：

1. 论文题目中“云平台”和“云服务平台”重复。

说明：参考专家修改意见，更改为：基于云平台的人脸识别服务的设计与实现，避免重复表述。

2. 摘要中声称：“3）解决了人脸识别算法….算法保密性和推广协议手续等问题”，是不正确的。保密性和协议手续不能通过云服务来自动解决，仍需要额外的手续来执行。

说明：摘要中第三点为：“3）解决了人脸识别算法在实际应用中涉及的算法保密性和推广协议手续等问题；”。原先想表达的是通过提供基于云平台的人脸识别服务，省去了开发人员在实际使用人脸识别算法时涉及的算法保密性和推广协议手续等问题。参考专家意见，考虑到表述的歧义，现将这点删除。

3. 论文设计的API考虑为开发人员提供多种算法的云服务接口。首先，具体的人脸识别算法的实现是否在本论文工作之内？其次，在大规模人脸识别优化时，似乎是针对特定的算法进行的优化，而非一般性的算法。需要对这些问题加以说明。

说明：首先，具体的人脸识别算法的实现不在本论文的工作之内；其次，针对大规模人脸识别算法，考虑到其运算量大的问题，本文借鉴K近邻搜索分类的原理，通过建立分类索引的方式对大规模人脸识别比对过程进行改进优化，并提出了相关参数的选择方法；通过重新设计和实现算法中的特征脸合成方法，维护代表性特征脸集合，省去了原先在Project Oxford人脸分组API中的

训练分组方法，简化了服务的步骤，优化了接口请求时间。已在文中增加了说明。

4. 在进一步工作和展望中，应对数据存储加以分析展望：不同的用户使用不同的参考数据库，这些数据库都需要上传到云平台，对云平台的存储和传输造成一定压力，如何解决?

说明：针对专家提出的数据存储问题，首先，本文项目中不同用户使用的是相同的参考数据库，不需要重复上传。其次，对于不同的用户而言，为了降低存储和传输的压力，本文项目中做了以下工作：1）在云平台上的索引数据仅维护分组个人的代表性特征脸集合的索引；2）在对云平台索引数据操作之前，计算比较代表性特征脸集合在操作前后是否会发生变化，如果未发生变化，则无需对云平台索引数据进行操作。这些措施，一定程度上缓解了一部分的存储和传输压力。