武汉理工大学毕业设计（论文）

**基于异步架构的图片管理系统后端设计和实现**

学院（系）：计算机科学与技术学院

专业班级：软件工程软件zy1501班

学生姓名： 刘曦光

指导教师： 陈明俊

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：

年 月 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理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。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于1、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名：

年 月 日

导师签名：

年 月 日

摘 要

在互联网、多媒体时代，图片无处不在，云计算服务商针对图片垂类的支持欠缺，各个企业重复开发现象严重。图片管理是CPU密集型、IO密集型场景，使用同步处理时延高、流程长，系统高峰期、低峰期负载相差较大，造成资源浪费。

使用异步架构能够很好地解决同步场景下的问题，将处理流程分为核心路径和非核心路径，核心路径使用同步处理，非核心路径通过消息队列异步化处理，缩短处理流程，降低处理时延，异步化还能起到削峰的作用。添加、删除、修改非核心路径的处理，不需改动核心路径代码，增大系统的可维护性和可扩展性。

本论文首先介绍开发背景，分析系统需求，针对功能需求和非功能需求，对系统进行架构设计，最后给出系统的实现和系统测试结果。架构设计将数据状态保存在独立组件中，本系统实现了服务无状态化，可实现水平扩展。

关键词：图片管理；异步架构；消息队列；分布式存储

**Abstract**

In the age of the Internet and multimedia, image is ubiquitous. The cloud service providers have not met the demand of image management, resulting in enterprise design and build their own system. Image process is an IO and CPU intensive scene. With synchronous model, it will cost a lot of time when the process is complex. And it waste system resources, because system load diff a lot between high and low peak period.

With asynchronous model, problem will be solved automatically. Divide processes into core and secondary. The core process uses synchronous model and the secondary processes are asynchronized by message queue. By simplify process, shorten the reply time and increase the throughput during high peak period. Update, insert or delete secondary processes without modify the code of core process that makes system more robust.

This paper describes the develop background firstly. Then analyses requirements. In order to meet functional and non-functional requirements, designs the architecture of image management system and introduces how to implement. Finally show the system test report. All durable storage maintain in independent components. The services built by this thesis are horizontally expandable.

**Key Words：**image management system; asynchronous model; message queue; distributed storage

目 录

[第1章 绪论 1](#_Toc9516682)

[1.1研究背景 1](#_Toc9516683)

[1.2 国内外研究现状 1](#_Toc9516684)

[1.3 研究目的及意义 2](#_Toc9516685)

[1.4 课题研究内容 2](#_Toc9516686)

[第2章 图片管理系统需求分析 4](#_Toc9516687)

[2.1需求分析概述 4](#_Toc9516688)

[2.1.1需求分析的目的 4](#_Toc9516689)

[2.1.2需求分析的流程图 4](#_Toc9516690)

[2.2开发背景 4](#_Toc9516691)

[2.3开发目标 5](#_Toc9516692)

[2.4可行性分析 5](#_Toc9516693)

[2.4.1经济可行性分析 5](#_Toc9516694)

[2.4.2工程上可行性分析 6](#_Toc9516695)

[2.5功能模块需求分析 7](#_Toc9516696)

[2.5.1用例图设计 8](#_Toc9516697)

[2.6非功能性需求 10](#_Toc9516698)

[2.7本章小结 11](#_Toc9516699)

[第3章 相关理论与技术 12](#_Toc9516700)

[3.1图片管理系统前端与后端 12](#_Toc9516701)

[3.2开发工具及环境介绍 13](#_Toc9516702)

[3.2.1Go编程语言 13](#_Toc9516703)

[3.2.2MySQL关系型数据库 14](#_Toc9516704)

[3.2.3Redis缓存系统 14](#_Toc9516705)

[3.2.4HDFS分布式存储 14](#_Toc9516706)

[3.2.5NSQ消息队列 16](#_Toc9516707)

[3.2.6Docker容器部署 16](#_Toc9516708)

[3.3通信协议 17](#_Toc9516709)

[3.4开发框架 17](#_Toc9516710)

[3.4.1消息队列NSQ客户端go-nsq 18](#_Toc9516711)

[3.4.2HDFS客户端WebHdfs 18](#_Toc9516712)

[3.4.3MySQL关系型数据库ORM Gorm 18](#_Toc9516713)

[3.4.4Redis缓存客户端go-redis 18](#_Toc9516714)

[3.5本章小结 18](#_Toc9516715)

[第4章 基于异步架构的图片管理网站的后端设计 19](#_Toc9516716)

[4.1概要设计 19](#_Toc9516717)

[4.1.1设计思想 19](#_Toc9516718)

[4.1.2系统功能结构 19](#_Toc9516719)

[4.1.3层次结构 20](#_Toc9516720)

[4.2架构设计 21](#_Toc9516721)

[4.2.1设计模式 21](#_Toc9516722)

[4.2.2系统组件选择 22](#_Toc9516723)

[4.2.3系统组件交互 22](#_Toc9516724)

[4.3处理流程设计 23](#_Toc9516725)

[4.3.1上传 23](#_Toc9516726)

[4.3.2删除图片 24](#_Toc9516727)

[4.3.3下载图片处理流程 25](#_Toc9516728)

[4.3.4查看用户上传历史 26](#_Toc9516729)

[4.3.4查看图片元数据 27](#_Toc9516730)

[4.4数据库设计 28](#_Toc9516731)

[4.4.1用户信息表设计 28](#_Toc9516732)

[4.4.2图片元数据表 29](#_Toc9516733)

[4.5存储系统中Key设计 30](#_Toc9516734)

[4.5.1Redis键设计 30](#_Toc9516735)

[4.5.2HDFS键设计 31](#_Toc9516736)

[4.6消息队列消息格式设计 31](#_Toc9516737)

[4.6.1图片上传消息设计 32](#_Toc9516738)

[4.6.2图片删除消息设计 32](#_Toc9516739)

[4.7本章小结 32](#_Toc9516740)

[第5章 基于异步架构的图片管理系统后端实现 33](#_Toc9516741)

[5.1接口调试 33](#_Toc9516742)

[5.1.1上传图片 33](#_Toc9516743)

[5.1.2删除图片 34](#_Toc9516744)

[5.1.3查看上传历史 35](#_Toc9516745)

[5.1.4查看图片元数据 36](#_Toc9516746)

[5.1.5下载图片 37](#_Toc9516747)

[5.2系统运维 38](#_Toc9516748)

[5.2.1NSQ运维 38](#_Toc9516749)

[5.2.2HDFS运维 40](#_Toc9516750)

[5.3系统测试 41](#_Toc9516751)

[5.3.1上传图片接口 41](#_Toc9516752)

[5.3.2删除图片接口 42](#_Toc9516753)

[5.3.3下载图片接口 42](#_Toc9516754)

[5.4本章小结 42](#_Toc9516755)

[第6章 总结与展望 44](#_Toc9516756)

[6.1研究总结 44](#_Toc9516757)

[6.2研究展望 45](#_Toc9516758)

[参考文献 46](#_Toc9516759)

[致谢 47](#_Toc9516760)

# 绪论

本章主要从研究背景、研究目的、研究意义、国内外研究现状、课题研究内容及预期目标几个方面进行相关的阐述。

## 1.1研究背景

在当今多媒体时代，图片借助其丰富的表达能力，被广泛应用于博客、即时通信、微博、新闻、帖子等场景。多媒体时代从PGC[1]（专业人员产出内容）逐渐转移到UGC（用户产生内容）。随着智能手机的发展，通过拍照、修图等方式产生图片的成本大大降低，微信、微博、抖音等平台提供用户方便地上传图片。阅读图文时，阅读图片的效率比文字高；即时通信中，用户倾向于使用图片表情进行表达；视频审核中，相比播放完整的视频，更高效的做法是在短时间内审核视频的关键抽帧；人机交互中，图片更容易吸引人的注意力，更好地引导用户操作，提高用户体验。

## 1.2 国内外研究现状

图片管理是IO密集型、CPU密集型的场景，面临如此场景，需研究出通用的方法解决该场景。在使用小众的APP时，常遇到图片上传、加载缓慢的情况，在极端的情况下，在设定的超时时间内，没有完成图片上传、加载给用户反馈是失败的场景，在高峰期间表现为服务不可用。异步化[2]处理能够很好地解决这个问题，将图片上传过程做拆分，借助消息队列[15]，将IO密集、CPU密集型处理异步到不同时间、不同机器中处理，从而达到缩短用户感知到的处理时延，提高用户体验，以及达到削峰的效果，更高效地利用机器资源。

图片管理是IO密集型、CPU密集型的场景。普通图片大小从几KB到几十MB不等，随着摄影技术的进步、显示器分辨率的增加，人们对于高质量的图片的追求，图片的大小、格式将会越来越多。IO密集型主要体现在图片在网络中传输、磁盘读写需要耗费大量的带宽；CPU密集型主要体现在在对图片进行格式转换、水印添加、压缩等处理。Facebook研发的Haystack[16]针对的是图片的存储的场景，其设计减少磁盘IO从而达到降低系统时延的目的，论文没有讲解异步化架构。

国内外云计算[3]发展迅速，云计算服务商提供的对象存储、块存储、文件存储、数据库存储服务能够解决最基本的存储问题，但对于用户体验的优化，为图片定制化的服务还不够完善，如添加水印、图片压缩、格式转化等。为图片存储设计一个专门的架构服务是值得的，鉴于其巨大的需求和技术挑战难度。腾讯云提供的行业解决方案中有直播解决方案、电商解决方案，但是缺少对图片这个场景的管理。直接将图片当作一个简单的文件处理还不够合适，因为图片除了基本的二进制数据，其中还需要存储大量的元数据，其数据格式表现为键值对。

小视频的表达能力比图片丰富，比GIF类动态图片多了音频的维度，抖音、微视、快手等产品将小视频推到风口浪尖，是不是小视频发展到一定程度就能完全替代图片呢？仔细研究后发现，每个小视频还需一张封面图片的，作为该视频的一个简单介绍。

在云计算尚未大面积普及下，大部分网站简单地将图片作为一个文件存储在文件系统中，较差的实践是将图片作为二进制流存储在数据库中，比如MySQL支持BLOB、MongoDB[4]支持GridFS，这种方案不仅对图片大小有限制，比如MongoDB因为行大小限制，存储文件必须小于16MB，对大文件的存储MySQL将其作为文件存储于文件系统中。将文件存储于数据库会增大数据库IO压力，在混合部署情况下，有可能影响到别的业务，造成数据库级别的抖动；增加数据迁移的成本，降低数据库的可扩展性。

小型网站中对于图片的处理大多是同步，一般图片上传步骤可以分为：

1. 用户选择图片进行上传
2. 反向代理、负载均衡将请求路由到具体处理的实例
3. 服务实例收到图片对图片进行压缩，格式转换，水印添加等
4. 元数据存储于数据库
5. 返回处理结果给用户

一般情况下，用户需要等待几秒钟才收到处理结果，在高峰期甚至会造成用户一直上传都无法成功的情况，极大地伤害了用户体验。

## 1.3 研究目的及意义

本次研究的目的在于，抽象图片的常用处理过程，拆分过程，优化处理流程，将没有依赖关系的过程改为并发处理，将非核心逻辑异步化。从而达到优化用户体验、系统削峰、增加系统可扩展性的目的。提供图片异步化架构的后台通用设计方案和实现，供有需要的开发者做参考。

## 1.4 课题研究内容

课题研究的主要内容如下：

1. 需求收集，从网上调研、用户调研收集、挖掘图片管理网站[5]常见需求。
2. 分析需求，设计图片管理流程。
3. 流程优化，将没有依赖关系的步骤并发进行，非核心路径异步化处理。
4. 架构设计，异步通过消息队列实现，调研相关存储系统。
5. 架构实现，编写代码，实现系统相关功能。
6. 结果测试，测试系统的正确性，以及系统的处理时延、并发量、吞吐量等数据。

# 第2章 图片管理系统需求分析

在实现系统前，对系统的目标用户需求做完整的调研分析是值得的。只有真实地理解了用户需求，才能够确保系统的正确实现，否则仅凭想象猜测用户需求，最后开发出的系统不是用户需要的，造成极大的人力浪费，无法交付能满足用户需求的系统。本章将对图片管理系统需求进行简单地概述，从功能性需求、非功能性需求、可行性分析等三个方面对基于异步架构的图片管理系统进行需求分析。

## 2.1需求分析概述

软件的需求分析是给后续的设计开发实现提供一个参考模型，描述系统需实现的功能，涉及功能的合理性、规范性与可行性，为后续系统开发提供基础依据。

### 2.1.1需求分析的目的

分析行业中常见的图片管理场景，调研本系统所需实现的基本功能，业务处理流程等，找出现有常见图片管理系统的不足，为这些不足提供优化的解决方案。需求分析目的是找出竞品的不足，并对其做优化，定位系统的目标用户群，系统运行环境条件及给外界暴露的接口。

### 2.1.2需求分析的流程图

对图片管理系统调研，总结出基于现有条件能够开发出的基于异步架构的图片管理系统的需求，基于结果分析业务需求，经过分析业务需求，与目标用户明确需求，设计系统的结构和功能。根据需求分析流程图每个步骤。

## 2.2开发背景

通过调研常见APP图片使用场景，以及用户调研，得出用户常见的需求有：

1. 图片的上传
2. 图片的下载
3. 图片的元信息查询
4. 图片的格式转换
5. 图片的压缩
6. 图片的水印添加
7. 图片的删除

图片被广泛使用，几乎每个网站都离不开图片管理。图片管理是CPU密集型、IO密集型的场景，处理过程中常需花费较长时间，造成用户长时间等待，用户体验大打折扣。系统使用同步处理方式无法达到削峰的目的，在高峰期造成服务不可用。

## 2.3开发目标

本系统是后端系统，除需考虑用户需求外，还需调研前端开发人员希望后端暴露的接口形式及接口的保证等。后端系统主要考量的标准有：

1. 接口处理时延
2. 接口支持并发度
3. 接口处理失败率
4. 接口的扩展性
5. 系统可维护性
6. 系统的容错性
7. 系统占用的资源

将图片处理流程做优化处理，借助消息队列实现异步化，将非核心路径流程转移到不同时间、不同机器上处理，降低系统的处理时延，高峰期起到削峰的作用。

## 2.4可行性分析

在真实开发前对需求进行可行性分析的作用是衡量项目的经济价值、工程实现上是否可能及所需成本。

### 2.4.1经济可行性分析

经济价值通过供需关系体现。

首先阐述需求，图片管理是一个非常普遍的需求，几乎每个网站都有该需求，在电子商务、社区论坛、微博等平台更是图片表达能力以及被阅读数大大高于文字。在智能手机的大面积普及下，用户产出图片的成本越来越低，速度也越来越快，通过微信、微博、头条图文等平台，用户将图片上传到云端并分享的行为越来越多。百度云盘、苹果iCloud提供云存储服务，帮助用户保存记录美好生活的图片，云端图片数据越来越大。

再来描述供给，图片管理系统的设计与实现由网站独立设计与实现的，对于大型网站其背后有复杂的系统架构，处理流程非常完善，但不提供给外界第三方使用，拥有类似需求的公司需独立开发类似的系统，造成大量的人力浪费，重复造轮子情况严重。小公司在人力匮乏，缺少高质量的程序员情况下，简单地将图片存储到本地文件系统，更不好的实现是将文件存储到数据库中，在网站规模小时，问题尚未暴露出来，当规模成长时，往往会遇到可扩展性、数据冗余存储、数据迁移等问题，成为阻碍业务发展的瓶颈。云计算服务提供商提供的是粒度更加粗的服务，比如对象存储、文件存储、块存储及数据库存储，对图片垂类的支持不够完善，比如图片防盗、异步化处理支持。云服务商很大的竞争力是保证存储的水平扩展，将存储直接对接CDN，大大地降低了使用方介入的成本。

综上所述，设计与实现基于异步架构的图片管理系统在经济上是可行的。

### 2.4.2工程上可行性分析

得益于开源社区的迅猛发展，数据库存储系统、分布式存储系统[6]、缓存系统[7]、消息队列系统和容器化管理已经非常完善，对于快速开发一个可靠的系统提供了极大的支持。图片管理系统常见的存储需求有：

1. 图片二进制数据存储
2. 图片元数据存储

图片的二进制信息可以直接存储到分布式存储系统上，或存储到云服务商提供的对象存储服务中。在企业中采用云服务商提供的对象存储服务更加稳定，减少企业的运维成本。但在探索更好的系统架构时，云服务商提供的对象存储服务像是一个黑盒，不了解其中的实现细节，在公网中传输数据会造成大量的流量带宽成本。分布式存储系统中HDFS尤为出色，被大数据生态广泛依赖，Spark、Hive等处理输入、中间结果、最终结果常保存在HDFS中，Kafka消息队列可以将消息dump到HDFS做持久化存储，企业级数据库通常借助HDFS做快照，所以分布式存储可以采用HDFS。

在编程实现中，应面向接口编程，而不是依赖具体实现。这能给系统带来更大灵活性，只需替换具体的实现即可，实现代码复用。在实现中，将存储接口抽象化，后续使用抽象接口作为编程调用对象。

图片的元数据可以分为固定的元数据和某张图片特有的元数据。固有的元数据有图片所有者、图片大小、图片文件名。某张图片特有的元数据是由用户对图片打标签，其表现形式为，地点：武汉、人物：自拍等，此类元数据可帮助用户分类图片，提供多维度的数据记录功能，以便实现图片筛选、搜索等功能。将图片固定的元数据保存在关系型数据库中，将特有的元数据序列化为JSON保存在extra字段中，从而达到系统扩展性要求。更好的实践是将特有的元数据保存在独立的键值对存储系统中，比如RocksDB等。关系型数据库系统开源社区较为成熟的选择为MySQL。

缓存系统将数据短暂存于内存，避免重复频繁地、耗时地去磁盘加载。业界使用最广泛的缓存有2种：

1. 独立内存缓存集群，比如Redis集群、Memcache集群
2. 编程语言实例级别内存缓存，比如HashMap等

缓存系统需要解决的问题是如何降低内存使用、提供内存超时失效机制、如何将Key进行打散负载均衡，怎么防止缓存被击穿，导致数据库存储系统雪崩。对于这种问题，业界已经提供了解决方案，比如Redis有Twemproxy、Redis Cluster机制，能够保证服务的可靠性、负载均衡等；对于内存缓存，可以在负载均衡中将请求按照打散规则路由到特定实例，从而实现LocalCache功能，在负载均衡随机分发请求的场景下，当服务实例数量达到一定阈值后，本地缓存通常不能起太大的作用，因为用户请求打到同一台机器上的可能性太低。

消息队列业界中的选择也非常多，Kafka、NSQ、RocketMQ、RabbitMQ、ZeroMQ，每个消息队列都有其特定的应用场景，开发中遇到的大部分场景都能够被很好地满足。

容器化管理在云计算中非常流行，容器化部署中的Docker、Docker-Compose、K8s能够快速方便地将服务进行部署，提供开发人员DevOps的能力，降低系统的运维成本。

鉴于基于异步架构的图片管理系统后端设计与实现中所需要用到的技术都非常成熟，需要解决的问题业界都已经提供了解决方案，在工程上是可行的。

## 2.5功能模块需求分析

图片管理网站的需求主要有：

1. 上传
2. 删除
3. 下载
4. 加水印
5. 格式转化
6. 压缩

通过分类可以分成：

1. 读模块
2. 写模块
3. 消费者模块

### 2.5.1用例图设计

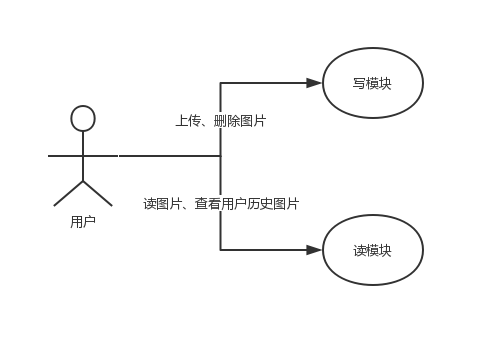
后端系统暴露给前端开发人员开发使用，用例图以接口的形式暴露，而不是像APP一样提供最终目的用户的人机交互界面。整个场景中需处理的请求类型非常简单，仅暴露少数几个有用接口。

图2.1系统用例图

图片作为一个不可修改内容的数据，对于平时数据库需支持的CRUD操作，仅需提供读、写、查询接口，不需提供更新操作的接口。

表2.1用户上传图片描述

|  |  |
| --- | --- |
| 用例名称 | 图片上传 |
| 参与者 | 需上传图片的用户 |
| 用例描述 | 用户选择需要上传的图片，给该图片添加特定的标签，点击上传按钮进行上传 |
| 前置事件 | 用户已处于登录状态 |
| 异常事件流 | 用户没有处于登录状态 |
| 基本事件流 | 图片上传成功，告知用户上传成功信息，并且返回对应的图片ID |

表2.2用户删除图片描述

|  |  |
| --- | --- |
| 用例名称 | 图片删除 |
| 参与者 | 需删除图片用户 |
| 用例描述 | 用户在查看历史上传的图片中选择一张或多张需要删除的图片，点击删除按钮 |
| 前置事件流 | 用户处于已登录状态，用户查看历史上传图片 |
| 异常事件流 | 用户尚未处于登录状态，用户尝试删除一张不属于他的图片 |
| 基本事件流 | 用户成功删除图片，该图片不会再出现在用户上传历史 |

表2.3用户查看上传图片历史描述

|  |  |
| --- | --- |
| 用例名称 | 查看上传图片历史 |
| 参与者 | 需查看上传图片历史用户 |
| 用例描述 | 用户点击查看历史图片，其中支持分页功能，用户可以自定义跳转到第N页 |
| 前置事件流 | 用户处于登录状态 |
| 异常事件流 | 用户尚未处于登录状态 |
| 基本事件流 | 用户能够通过分页功能浏览到第N页图片 |

表2.4用户下载图片描述

|  |  |
| --- | --- |
| 用例名称 | 下载图片 |
| 参与者 | 需下载图片的用户 |
| 用例描述 | 用户在浏览上传历史页面中找到需要下载的图片，然后点击下载按钮 |
| 前置事件流 | 用户处于登录状态 |
| 异常事件流 | 用户尚未处于登录状态，或者尝试下载没有权限的图片 |
| 基本事件流 | 用户下载某张图片，然后将其保存在本地文件系统，或在浏览器中查看 |

表2.5查看图片元数据描述

|  |  |
| --- | --- |
| 用例名称 | 查看图片元数据 |
| 参与者 | 需查看图片 |
| 用例描述 | 用户查看某特定图片ID的元数据 |
| 前置事件流 | 用户处于登录状态 |
| 异常事件流 | 用户尚未处于登录状态，或者尝试查看没有权限的图片ID的元数据 |
| 基本事件流 | 用户查看谋者特定图片的元数据 |

如上述，本系统只满足以上5个基本功能，向前端暴露5个基本接口。

## 2.6非功能性需求

上节描述系统的功能性需求，描述系统需提供用户怎样的能力去完成功能需求，是系统能看得到摸得着的功能。本节描述本系统的非功能性需求，从技术角度衡量系统，如系统的最大吞吐量、最大并发量、可用性、可维护性、测试覆盖率、可重用性、安全性等。

软件开发中不提倡过早优化，指不要过早优化非功能性需求，以及代码的细节。

以下主要从可用性、测试覆盖率、可重用性、安全性、可扩展性、吞吐量、处理时延、兼容性等角度描述系统的非功能需求：

1. 可用性，系统需要保证99.9%的时间是可用的，一天不可用时间为86.4s。通过消息队列异步化，达到削峰的目的，保证在高峰期间，保证系统99.9%的可用性。
2. 测试覆盖率，没经过测试的代码是不可信的，测试需覆盖所有的代码可能执行的流程，需要写单元测试，在持续集成时进行代码的自动化测试，保证合并到master分支的代码都是正确的，能够上线的。
3. 可重用性，编写代码中需面向接口编程，而不是依赖具体实现编程，从而能保证在替换实现时，不需改动原处理流程。系统向外暴露尽可能小的接口，保证原子性，具体更加复杂的功能，由上游组合实现。
4. 安全性，不允许未登录用户操作图片数据，不允许用户操作不属于该用户的数据。
5. 可扩展性，因为数据状态统一保存在独立组件中，本系统代码实现中都是无状态的，可水平扩展，扩展性基于开源组件的可扩展性解决方案。
6. 吞吐量，因为服务无状态，可实现水平扩展，所以这里描述的吞吐量为单机吞吐量，吞吐量需要保证在100QPS。
7. 处理时延，对于读请求，保证500ms；对于写请求，保证1s。
8. 兼容性，本系统仅兼容支持Docker[8]的操作系统，因为使用容器化部署，所以只需要操作系统支持Docker，则可运行本系统。

表2.6运行环境

|  |  |
| --- | --- |
| CPU | Intel Core i7-4710M |
| 内存 | DDR3 12GB |
| 硬盘 | 机械硬盘，7200转/分钟，容量520GB |
| 网卡 | Realtek PCIe GBE Family Controller |
| 操作系统 | Linux |

## 2.7本章小结

本章对基于异步架构的图片管理系统后端的开发背景、开发目标、可行性分析、功能需求和非功能需求等方面进行需求分析。在功能需求中，讲解系统开发为了应对的主要用户场景；在非功能需求中，描述系统应该对外界的保证。通过需求分析，能帮助用户进一步了解本系统。

# 第3章 相关理论与技术

异步化是将非核心路径的处理转移到不同服务实例、不同时间进行处理，达到提高用户体验，系统削峰、流程优化、提高系统可扩展性的目的。

本系统主要使用的技术有：

1. NSQ消息队列系统
2. MySQL关系型数据库系统
3. HDFS分布式存储系统
4. Redis缓存系统
5. Go编程语言
6. Docker容器管理系统

本章将对这些技术进行详细讲解，以及阐述具体的技术选型理由。

## 3.1图片管理系统前端与后端

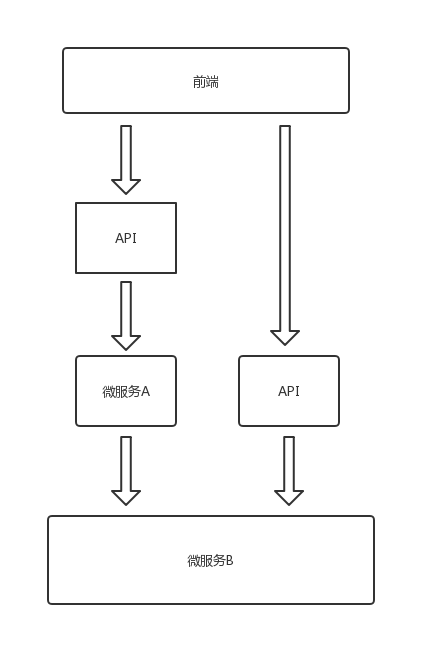
在微服务[9]架构中，后端调用者不仅有前端，也可以是其他微服务的后端的，调用关系如下：

图3.1微服务调用关系

服务以微服务接口供外界调用，通过RPC调用协议实现两个服务之间的通信，比如gRPC、Thrift、HTTP。这样做的好处是，实现服务的重用，避免不同的业务团队重复开发，提高人效。

本设计中系统的调用者主要有两类：

* 1. API层调用
  2. 微服务调用

API层的调用供前端开发人员调用的接口，HTTP协议是文本协议，可读性、通用性非常好，没有固定的Schema，非常灵活，能够支持文件传输、长连接等，在移动端、浏览器端交互常使用HTTP或HTTPs协议传输数据。但HTTP是一个没有固定Schema的协议，固定服务参数等不好统一，基于文本协议打包解包效率低下，企业级别的微服务通信协议广泛使用的有Thrift、gRPC，可以通过固定的协议文件，通过脚手架工具，生成相应的打包解包代码，打包内容为二进制级别的，传输效率更加高。从内网RPC协议转化为暴露给外网使用的HTTP协议需API层作为网关。

微服务调用来自其他业务团队，比如图文业务、电子商务业务、视频抽帧业务等，打造技术中台，这个思想最初来自SuperCell、Netflix等微服务的鼻祖，在技术中台打造成熟后，开发新业务成本就大大降低，像SuperCell中有大量的游戏，字节跳动的APP工厂，正是得益于中台技术的完善。

## 3.2开发工具及环境介绍

### 3.2.1Go编程语言

Go编程语言在微服务架构中非常火热，开源社区也极大地拥抱了Go编程语言，开源组件非常成熟，而且经过Google、AWS、字节跳动等多家公司实践验证，原生支持协程，适用于IO密集型场景。编译结果为一个可执行的二进制文件，没有虚拟机，执行效率比肩Java。

图片管理是IO密集型、CPU密集型的场景，与Go编程语言的定位正好匹配。服务对于每个新请求的处理流程主要有三种方式：

1. 每个请求创建一个新进程处理
2. 每个请求创建一个线程处理
3. 每个请求由一个线程处理，实现线程池，避免线程被频繁地创建、回收，造成额外的资源消耗
4. 独立线程epoll进行异步IO，将请求存放于队列中，线程从请求池中获取请求，然后处理
5. 与4类似，独立协程进行IO，每个请求使用协程处理

协程比线程更加轻量级，创建、销毁的成本更加低，只需要在一个栈以及相关上下文（栈指针、PC指针、相关寄存器）。

### 3.2.2MySQL关系型数据库

图片管理系统中图片二进制存储占用系统绝大部分的存储空间，但每个图片都需存储相关的元数据，如图片的ID、所有人、文件名、创建时间等，对于关系型的数据，存储进关系型数据库是一个好的选择。

在关系型数据库中，MySQL是开源社区中最被信赖、周边工具最完善的，经过工业级的验证。MySQL是一个行存储的数据库，每次都是从MySQL中获取行所有的数据，行存储的效率是最高的，且在可扩展性方面，开源社区对于MySQL的分库分表有很多可以选择的解决方案，比如增加Proxy中间层做sharding，在client-lib中做sharding等，在高可用方案上，实时备份、主从高可用可以通过binlog和浮动IP实现。

对业务特性，MySQL支持非常好，比如索引、外键约束、触发器、事务、过程调用等，能够很好地满足业务特性，对数据的一致性有保证。

### 3.2.3Redis缓存系统

缓存无处不在，比如浏览器、反向代理、DNS、数据库等，假如没有缓存，所有请求都直接传递到最后一层做处理，那么将会发生很多无谓的请求，增大数据库等组件的压力。在读多写少的场景，缓存能够将90%的请求直接返回，缓存数据保存在内存中，内存访问速度比磁盘快一个数量级，降低系统处理时延，提高用户体验。

Redis支持丰富的数据结构，比如链表、哈希表、字符串、跳表、集合、HyperLogLog和位图，Redis不仅可以存储字符串，还可以存储二进制内容，可缓存图片的二进制数据和元数据。

在可扩展性方面，Redis有Redis Cluster、Codis、TwemProxy等方案，支持Redis集群的水平扩展。在高可用方面，Redis通过复制日志进行主从同步，通过Redis Sentinel模式，检测主库是否不能正常工作，支持备库切换为主库。

### 3.2.4HDFS分布式存储

分布式存储是将多台机器组合成一个集群，然后对外提供文件级别的存储服务，业界使用较多的两个开源方案有：

1. HDFS
2. Ceph[14]

以下主要讲解HDFS和Ceph的工作原理，以及选择HDFS的理由。

HDFS是模仿Google Big Table研发的，在Hadoop[10]生态中扮演着关键存储的角色，Hive、Spark的输入、输出、中间处理结果都保存在HDFS中，支持机架感知，支持冗余灾备，将文件按照固定大小切割成固定大小的块，然后按照冗余备份的数量将文件冗余在多个实例。HDFS中主要有两类主要角色，DataNode和NameNode，DataNode负责保存文件的二进制数据，NameNode负责保存文件的元数据，比如某个文件的各个分片在哪个实例上等，NameNode一般是一个单点，通过主备来实现备份容灾，根据GFS描述，元数据大小不会超过单机内存限制，所以一个NameNode实例存储空间已足够。架构图如下：



图3.2HDFS架构图

Ceph是自调整的存储，提供对象存储、文件存储、块存储功能，被广泛应用到工业界，内部通过Paxos实现强一致性，避免单点失败问题，冗余复制在Ceph内部实现。Ceph在基于对象存储功能基础上，再封装出文件存储功能。

HDFS和Ceph的目标都是通过水平扩展的能力，将集群部署在廉价的机器上，通过冗余实现灾备，通过Paxos强一致性算法保证没有单点故障，向外提供强一致性的服务。HDFS比Ceph发展的时间更加久远，支持Docker运维部署，WebHDFS允许外界通过Restful[11] HTTP接口访问HDFS，提供DashBoard方便监控HDFS数据以及运行状况等。Ceph功能很全面，本系统实现中并不需如此丰富增大运维难度的特性，相对HDFS无明显优势，client的支持还不是很完善，所以使用HDFS作为分布式存储服务。

### 3.2.5NSQ消息队列

本系统设计核心是异步化，异步借助消息队列实现。

开源社区中常用的消息队列及其面向的应用场景有：

1. Kafka：支持消息持久化，面向场景是高吞吐量、高时延的场景，如日志收集处理。
2. RocketMQ：支持丰富的特性，是金融级别的消息队列，支持全局消息有序、消息持久化、事务性消息、支持消息重放，面向事务型场景。
3. NSQ：分布式、无中心，正常情况下消息保存在内存中，投递处理时延低，支持延迟发送，消息处理失败重试，提供可视化界面，大大降低了运维成本，面向实时处理场景。
4. ZeroMQ：集成到客户端lib，没有独立组件，是点对点的通信，支持的特性较为简单，面向点对点的RPC场景。

考虑运维部署成本和实时性，采用NSQ作为消息队列。

### 3.2.6Docker容器部署

虚拟化部署方案业界流行有两种：

1. 虚拟机：操作系统级别隔离，完全冗余了一份操作系统，资源利用率低。

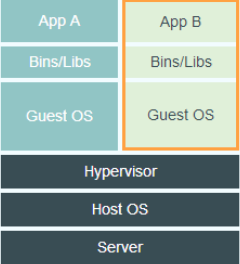


图3.3虚拟机

1. 容器：通过cgroup、Namespace、UFS实现。cgroup控制Docker容器对于内存、CPU的利用；Namespace实现进程级别的隔离，A容器无法访问B容器的进程；UFS通过挂载的方式复用父文件系统，从而实现文件系统的重用。

## 3.3通信协议

图3.4容器

鉴于通用性和灵活性，本系统对外开放的接口均采用HTTP协议，HTTP是最通用的通信协议，基于文本可读性非常好，能够使用Postman等工具进行调试，大大地降低了开发成本。

Go编程语言原生支持HTTP协议，实现了一个简单的HTTP server。

## 3.4开发框架

在软件工程中，框架就是轮子。使用开源框架能大大地提升开发效率，避免重复性工作，本系统中使用的框架主要目的是和其他组件进行交互。

### 3.4.1消息队列NSQ客户端go-nsq

NSQ是一个无中心的消息队列，比如延迟发送、处理失败重新入队等都是通过go-nsq客户端实现的，是一个胖客户端。生产者、消费者都是通过同一个client实现，借助官方提供的客户端大大地降低了接入成本。

### 3.4.2HDFS客户端WebHdfs

与HDFS实例交互需要特定的RPC，而Apache官方没有推出Go编程语言的客户端。HDFS提供Restful HTTP的方式和外界交互，降低外界的交互成本，可以通过cURL工具发起简单的测试请求。

### 3.4.3MySQL关系型数据库ORM Gorm

数据库关系映射是将数据库表中一行记录映射到编程语言中的一个对象中，避免开发者在使用数据库时重复地写映射代码。Go是一门较为新的编程语言，没有Java那么多ORM选择，Gorm是一个经过行业验证的方案。

### 3.4.4Redis缓存客户端go-redis

Redis数据库与客户端交互有固有的协议，类似MySQL协议，go-redis将Redis协议和Go编程语言进行了封装，降低了开发成本。

## 3.5本章小结

本章介绍本系统需涉及的技术，包括组件、通信协议及编程库，将具体技术进行简单的讲解，阐述某个技术的可选项和具体技术选型的理由，为后续讲解系统设计与实现做铺垫。

# 第4章 基于异步架构的图片管理网站的后端设计

第二章讲解了图片管理系统的后端需求，第三章讲解了本系统使用到的技术。本章讲解基于异步架构的图片管理系统的设计。在了解到本系统的需求后，进入软件设计阶段，讲解本系统中涉及的组件如何交互，对每个需求的处理流程分解、优化、异步化。

## 4.1概要设计

概要设计根据需求分析的用例图和用例描述转换成系统的功能结构和数据结构，是把需求转化为具体的设计方案，本节概要设计包括：系统功能结构、层次结构、接口设计。

### 4.1.1设计思想

将系统模块划分地尽可能小，解除模块的耦合关系。将处理流程进行梳理、优化，将处理流程中的步骤区分为核心路径和非核心路径，一次请求中只保证核心路径能够处理完成，将非核心路径中的处理步骤，借助消息队列异步化至消费者模块处理。将模块划分足够小，能促进模块的复用，以及降低测试、排错的难度。

### 4.1.2系统功能结构

图片管理系统后端的主要需求：

1. 上传
2. 下载
3. 格式转化
4. 添加水印
5. 查看图片元数据
6. 查看上传历史

为了满足需求，设计出系统模块为：

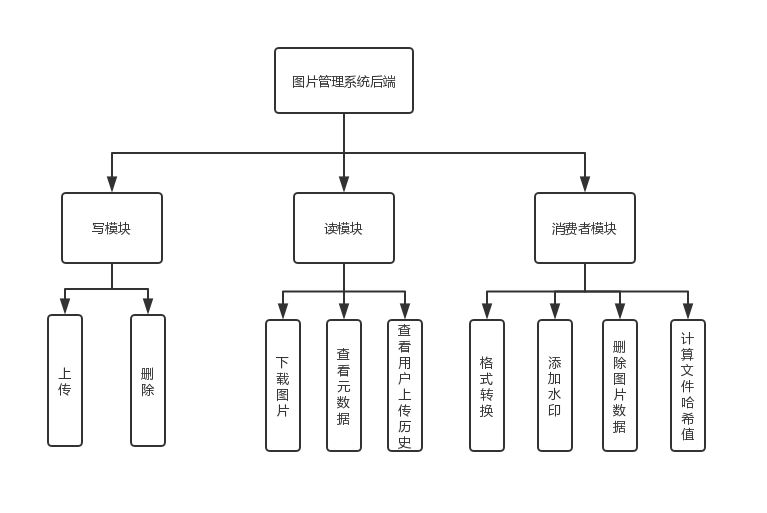
从处理的请求类型看，可将整个系统划分为三大模块：

图4.1系统模块设计

1. 写模块
2. 读模块
3. 消费者模块

写模块处理需写存储的请求，如上传、删除图片，其处理核心流程，将非核心流程通过消息队列异步化到消费者模块处理。

读模块提供接口供外部调用访问图片的二进制数据和元数据，仅处理读存储的请求。

消费者模块异步完成非核心流程处理，将处理均匀打散到不同时间、不同机器上处理，异步化达到降低处理时延、负载均衡、削峰的目的。

### 4.1.3层次结构

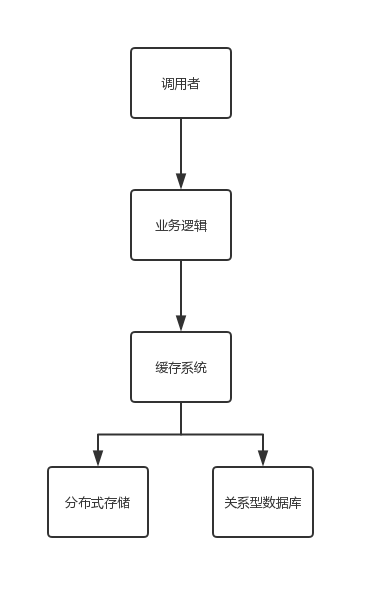
本系统仅向外提供JSON接口，不返回HTML/CSS/JS等可视化数据，相对于经典的MVC架构仅有MC，也就是只有模型层、控制层，如下图：

图4.2层次结构

将业务逻辑处理集中在业务逻辑层，也就是本系统中代码实现主要部分。

## 4.2架构设计

本节主要阐述设计模式选择、系统组件选择和组件是如何交互的。

### 4.2.1设计模式

本系统强调异步架构，异步通过消息队列实现，将非核心处理步骤异步化到不同时间、不同机器上做处理。因为采用消息队列，所以选择生产者消费者模式进行处理。

为了在开发前期中测试阶段满足测试需要，在还没接入HDFS分布式存储时，将图片简单地存储在本地文件系统中，方便后续替换不需要修改大量的代码，采用面向接口编程，对于每个实现独立测试。在后续实现中需替换分布式存储时，可将代码改动降低到最小。如需将本地存储替换为HDFS、HDFS替换为Ceph。

开发中采用Go语言推荐的实践，使用组合模式而不是集成模式。

因为WebHdfs中采用Restful HTTP协议交互，如果每个请求编程中都使用重复代码拼接HTTP请求，不仅代码复用率低，而且容易出错。将WebHdfs交互部分使用适配器模式进行抽离。达到每次只需调用适配器的代码，即能够访问WebHdfs的目的。

### 4.2.2系统组件选择

表4.1本系统使用的组件

|  |  |  |
| --- | --- | --- |
| 组件 | 选择技术 | 作用 |
| 消息队列 | NSQ | 异步化非核心路径处理步骤 |
| 关系型数据库 | MySQL | 存储图片元数据 |
| 分布式存储系统 | HDFS | 存储图片二进制数据 |
| 缓存系统 | Redis | 缓存图片二进制数据和元数据 |
| 编程语言 | Go | 负责与其他组件交互，实现业务逻辑 |

具体组件的选择可以参考第三章中系统术语和解释。

### 4.2.3系统组件交互

系统组件的交互逻辑控制在Go编写的代码中，整体关系如下图：

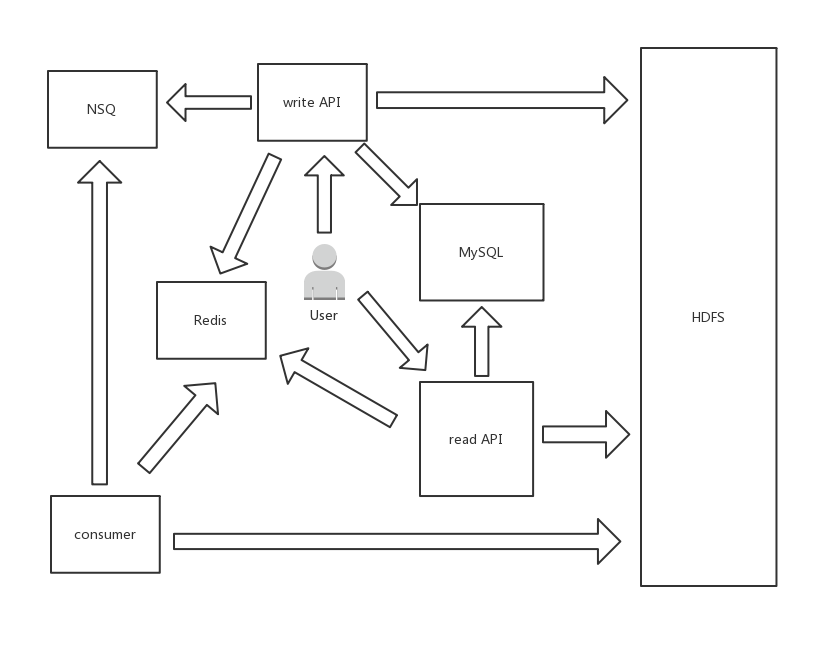
图中小写字母开头的模块为本系统实现的模块，大写字母开头的是开源社区提供的组件，图中User为前端调用或微服务调用。图中箭头方向为调用关系。

图4.3系统交互图

## 4.3处理流程设计

### 4.3.1上传

上传分为核心路径和非核心路径，核心路径为：

1. 开启事务[12]
2. 将图片元数据存储到MySQL，将图片二进制数据存储到HDFS
3. 如果2步骤处理成功，则提交事务，跳转到步骤5；否则跳转至步骤4
4. 删除步骤2存储到HDFS的图片，回滚事务，结束
5. 发送消息到消息队列，将非核心逻辑转移到消费者执行，结束

上传图片非核心路径：计算哈希值、格式转化、添加水印、压缩等功能作为独立消费者消费消息队列中的消息，然后将存储结果存储到HDFS、MySQL中。

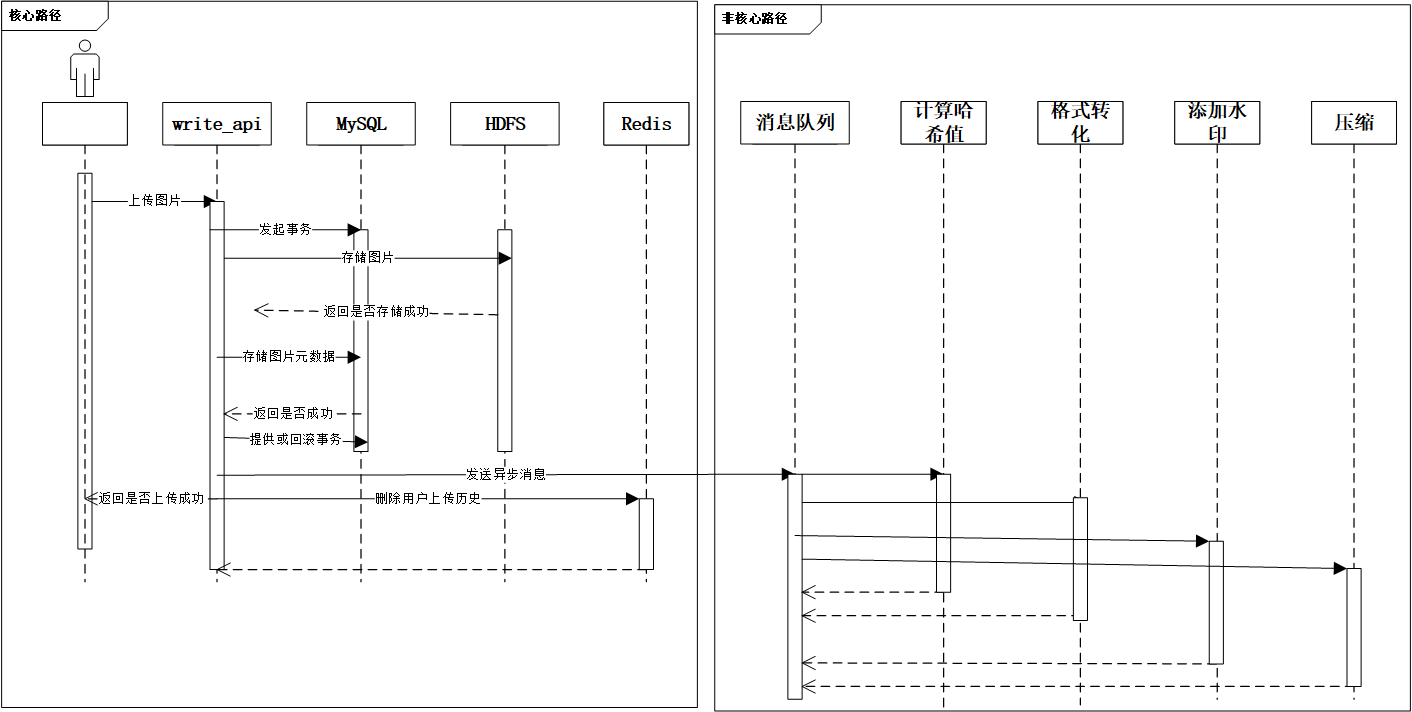
图片上传路径处理流程如下图：

图4.4上传流程

图4.5上传路径

### 4.3.2删除图片

删除图片的核心路径：

1. 开启事务，跳转到步骤2
2. 删除MySQL元数据、Redis缓存，跳转到步骤3
3. 若步骤2都处理成功，则提交事务，跳转到步骤4，否则跳转步骤5
4. 发送消息到消息队列
5. 返回客户端是否删除成功

删除图片非核心路径：删除图片多个格式的二进制图片数据，释放存储空间。

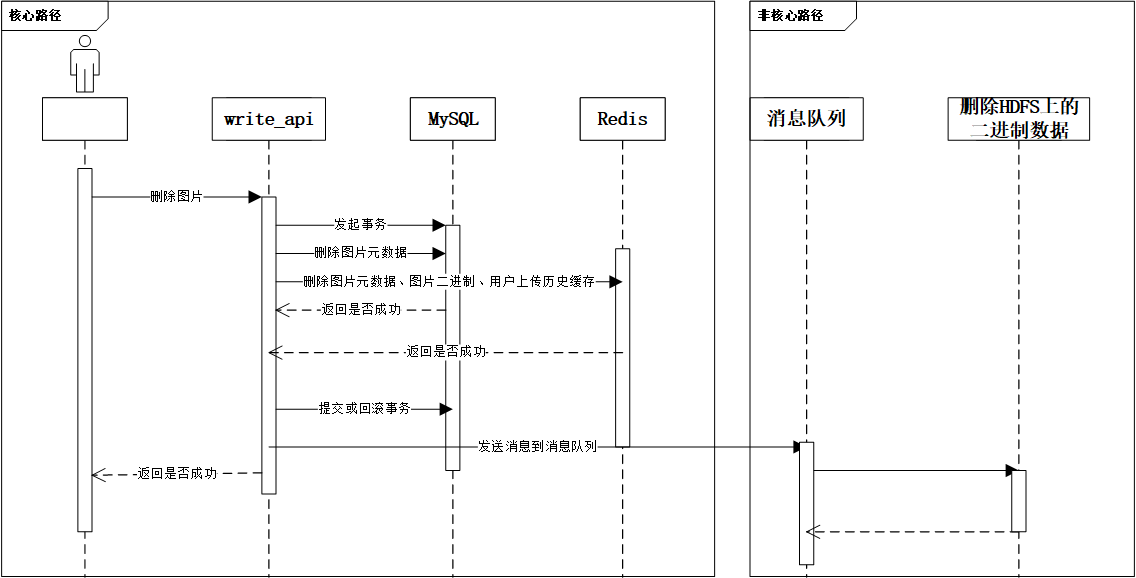
图片删除的处理流程如下图：

图4.5删除图片核心路径

### 4.3.3下载图片处理流程

下载图片处理流程：

1. 尝试并发从缓存获取图片二进制数据、元数据，跳转至步骤2
2. 若从缓存中获取二进制数据成功，则跳转至步骤6，否则跳转至步骤4
3. 若从缓存中获取元数据成功，则跳转至步骤6，否则跳转至步骤5
4. 从MySQL获取元数据，跳转至步骤6
5. 从HDFS获取二进制数据，跳转至步骤6
6. 等待获取二进制数据、元数据结束，将二进制数据、元数据打包返回给前端

下载图片流程如下图：

### 4.3.4查看用户上传历史

图4.6下载图片流程

查看用户上传历史处理流程：

1. 尝试从Redis缓存获取用户上传历史，缓存命中跳转至步骤，否则跳转至步骤3
2. 从MySQL中获取用户上传历史，跳转至步骤3
3. 返回历史元数据给用户

查看用户历史处理流程如下图：

### 4.3.4查看图片元数据

图4.7查看用户历史

查看图片元数据处理流程：

1. 尝试从Redis缓存中获取图片的元数据信息，若成功跳转至步骤3，否则跳转至步骤2
2. 从MySQL中尝试获取图片元数据信息，跳转至步骤3
3. 返回图片元数据给调用者

查看图片元数据处理流程如下图：

## 4.4数据库设计

图4.8查看图片元数据

本节介绍本系统将图片元数据存储到关系型数据库MySQL中，如何设计数据库表结构的。

### 4.4.1用户信息表设计

用户信息表用于简单地存储用户信息，其结构如下表：

表4.1用户信息表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 注释 |
| uid | bigint | 用户ID |
| status | tinyint | 用户状态 |
| extra | varchar(255) | 存储json串 |

创建表的语句如下：

CREATE TABLE `user` (

`uid` bigint(20) NOT NULL,

`status` tinyint(4) NOT NULL DEFAULT '0',

`extra` varchar(255) DEFAULT NULL,

PRIMARY KEY (`uid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

### 4.4.2图片元数据表

图片元数据表用于存储图片的元数据，其结构如下表：

表4.2图片元数据表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 注释 |
| fid | bigint | 图片id |
| uid | bigint | 对应用户表的中uid，标识文件的所有者 |
| name | varchar(255) | 文件名 |
| size | bigint | 文件大小 |
| md5 | char(32) | 文件MD5哈希值 |
| create\_time | timestamp | 上传文件的unix时间戳 |
| update\_time | timestamp | 修改图片数据的unix时间戳 |
| extra | varchar(255) | 存储json串，用户自定义的数据 |
| status | tinyint | 图片状态 |

创建表语句如下：

CREATE TABLE `file` (

`fid` bigint(20) NOT NULL,

`uid` bigint(20) NOT NULL,

`name` varchar(255) NOT NULL DEFAULT '',

`size` bigint(20) NOT NULL,

`md5` char(32) NOT NULL,

`create\_time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`update\_time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

`extra` varchar(255) DEFAULT NULL,

`status` tinyint(4) DEFAULT NULL,

PRIMARY KEY (`fid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

## 4.5存储系统中Key设计

本系统中采用Redis键值对存储系统和HDFS存储系统，如何设计Key是一个关键，怎么防止键冲突以及保证可读性[13]。

### 4.5.1Redis键设计

本系统使用到Redis缓存的地方有：

1. 图片元数据
2. 图片二进制数据
3. 用户上传历史缓存

表4.3Redis键设计

|  |  |  |  |
| --- | --- | --- | --- |
| 场景 | 键模式 | Redis数据结构 | 存储数据 |
| 缓存图片元数据 | f%d  （%d为图片id） | string | 将元数据对象序列化为json字符串 |
| 缓存图片二进制数据 | c\_%d\_%d  （第一个%d为图片id，第二个%d为图片格式） | string | 图片二进制数据 |
| 用户上传历史 | p\_%d  （redis中对应的键名称，%d为用户id）  %d\_%d  （hash中键，第一个%d为offset，第二个%d为limit） | hash | 将图片元数据对象序列化为json字符串 |

### 4.5.2HDFS键设计

HDFS是一个分布式文件存储系统，如何区分目录、标识文件名至关重要，在本系统中将图片二进制数据作为一个文件存储到HDFS中。

每个文件有多个格式：

1. 原图
2. JPEG格式
3. PNG格式
4. 带水印的JPEG格式
5. 带水印的PNG格式

对于不同格式的图片定义以下枚举：

const (

InvalidImageFormat ImageFormat = iota

Jpeg

Png

WaterMarkJpeg

WaterMarkPng

)

下表中%d代表图片ID

表4.4HDFS图片格式文件名

|  |  |
| --- | --- |
| 格式 | 键格式 |
| 原图 | %d |
| JPEG格式 | %d\_1 |
| PNG格式 | %d\_2 |
| 带水印JPEG格式 | %d\_3 |
| 带水印PNG格式 | %d\_4 |

## 4.6消息队列消息格式设计

消息即事件，记录who-when-what三个维度的信息，消息队列中消息类型：

1. 图片上传消息
2. 图片删除消息

### 4.6.1图片上传消息设计

表4.5图片上传消息格式

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| Fid | int64 | 图片ID |
| Uid | int64 | 用户ID |
| Timestamp | int64 | UNIX时间戳 |
| Extra | string | 对于本消息特殊说明 |

### 4.6.2图片删除消息设计

表4.6图片删除消息设计

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| Fid | int64 | 图片ID |
| Uid | int64 | 用户ID |
| Timestamp | int64 | UNIX时间戳 |
| Extra | string | 对于本消息特殊说明 |

## 4.7本章小结

本章讲解了基于异步架构的图片管理系统的后端设计方案，描述了系统使用的组件，以及组件之间如何进行交互。通过UML时序图描述图片上传的处理步骤，如何将处理流程拆分为核心路径和非核心路径，核心路径中在当前请求生命周期内完成，非核心路径通过消息队列异步到不同时间、不同实例上完成。介绍MySQL数据表设计、消息队列消息的格式设计、Redis缓存键组合设计和HDFS分布式存储系统的文件名设计。通过本章能了解到本系统的设计细节，也是论文的重点。

# 第5章 基于异步架构的图片管理系统后端实现

上一章讲解了基于异步架构的图片管理系统后端设计，这一章在设计的基础上讲解具体的实现，主要从接口格式、输入输出进行讲解。

本系统对外暴露主要有5个接口：

1. 上传图片
2. 删除图片
3. 下载图片
4. 查看图片元数据
5. 查看用户上传历史

## 5.1接口调试

本节阐述接口调试，使用截图方式展现接口处理结果。服务接口路径说明：

表5.1接口路径说明

|  |  |
| --- | --- |
| 接口名 | 路径 |
| 上传图片 | <http://127.0.0.1:10003/post> |
| 删除图片 | <http://127.0.0.1:10003/delete> |
| 查看上传历史 | <http://127.0.0.1:10002/userFile> |
| 下载图片 | <http://127.0.0.1:10002/get> |
| 查看图片元数据 | <http://127.0.0.1:10002/meta> |

### 5.1.1上传图片

HTTP请求表单数据说明：

表5.2上传图片表单

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| uid | int | 用户ID |
| file | 文件 | 用户在浏览器选择需要上传的文件 |

上传截图：

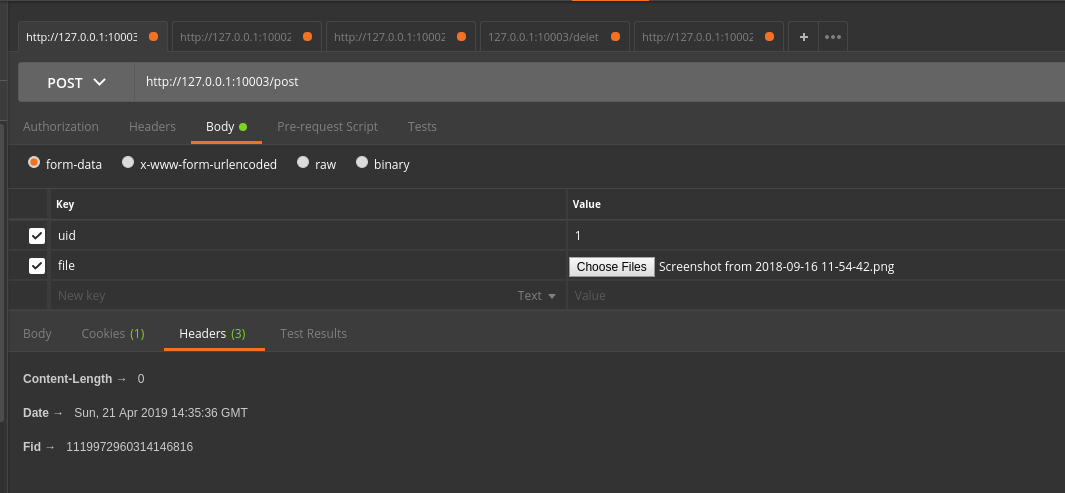
HTTP响应头中会说明成功上传的图片ID，用户可以凭借该ID查看图片元数据和下载图片。

图5.1上传图片截图

### 5.1.2删除图片

HTTP请求表单数据说明：

表5.3删除图片表单

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| uid | int | 用户ID |
| fid | int | 图片ID |

其中fid可以出现多行，接口支持同时删除多个图片。

删除图片截图：

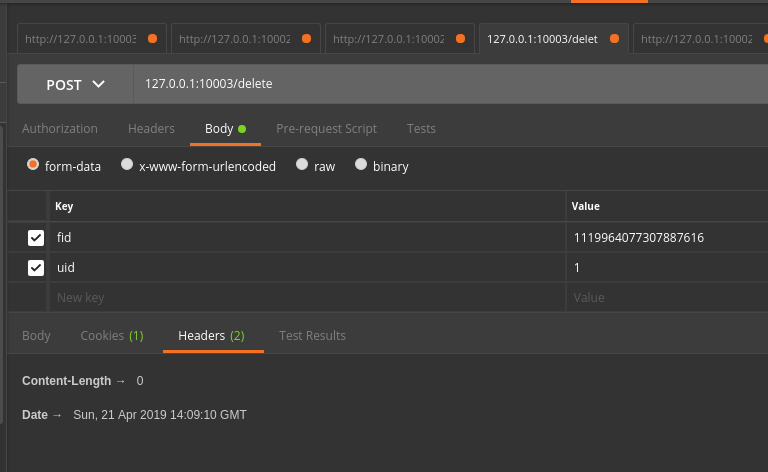
根据HTTP状态码是否为200OK判断是否已经上传成功。

图5.2删除图片截图

### 5.1.3查看上传历史

HTTP请求表单数据说明：

表5.4查看用户历史表单

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| uid | int | 用户ID |
| offset | int | 偏移量，用于分页 |
| limit | int | 每页条数，用于分页 |

查看用户历史请求截图：

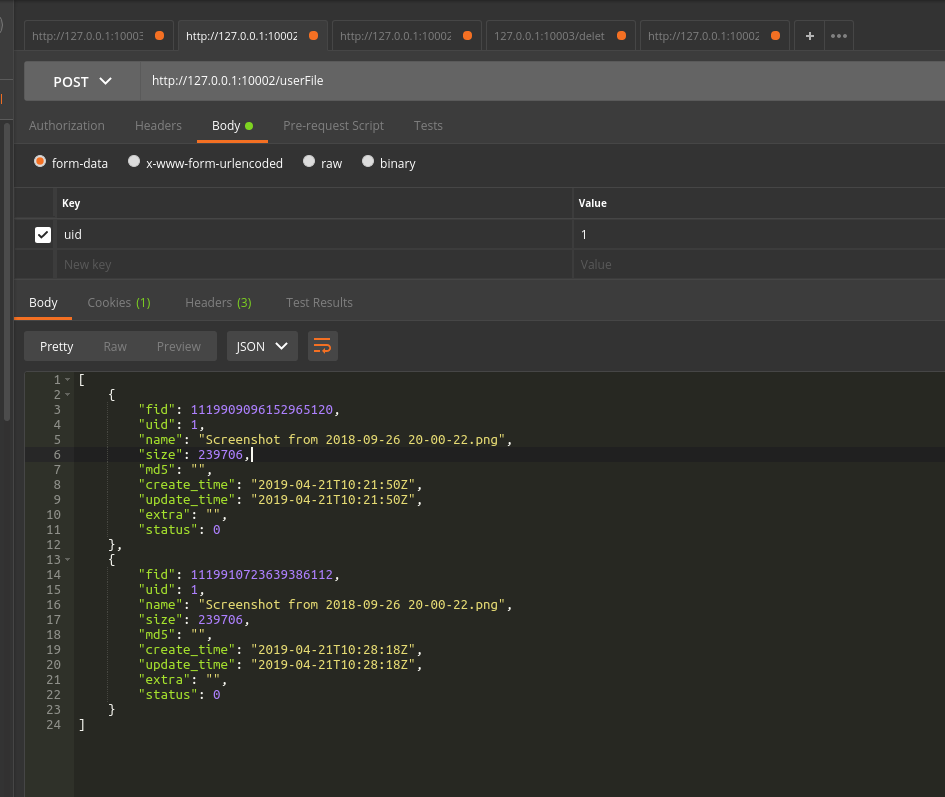
HTTP响应体中为用户上传历史的图片元数据，其中包括图片ID、用户ID、文件名、字节数、哈希值、创建时间、修改时间和状态等数据。

图5.3查看用户上传历史截图

### 5.1.4查看图片元数据

HTTP请求表单说明：

表5.5查看图片元数据表单

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| uid | int | 用户ID |
| fid | int | 图片ID |

其中fid可以重复出现，代表查看多个图片的元数据。

查看图片元数据截图：

### 5.1.5下载图片

图5.4查看图片元数据

不像其他读接口，本接口仅支持下载一个图片，不支持同时下载多个图片。HTTP请求表单说明：

表5.6下载图片表单

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| uid | int | 用户ID |
| fid | int | 图片ID |
| format | string | 需要下载图片的格式，下载原图不传递这字段 |

下载图片截图：

## 5.2系统运维

图5.5下载图片截图

系统启动通过Dokcer-Compose完成，Redis、MySQL、HDFS均通过Docker一键部署，通过执行docker-compose up –d启动项目。MySQL、Redis通过命令行交互，NSQ和HDFS可通过配套的可视化工具运维，以下将简单介绍使用NSQ、HDFS可视化工具运维。

### 5.2.1NSQ运维

NSQ有三大组件：

1. Nsqd：负责接收、分发消息
2. NsqLookupd：负责Topic、Channel的服务发现
3. NsqAdmin：提供可视化管理界面

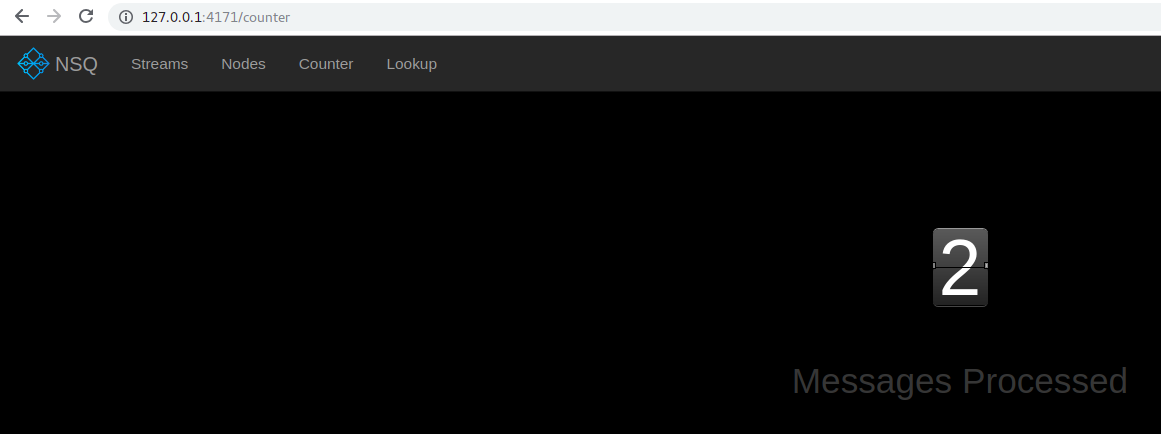
查看消息队列接受到的消息数量：

图5.6查看nsq消息数量

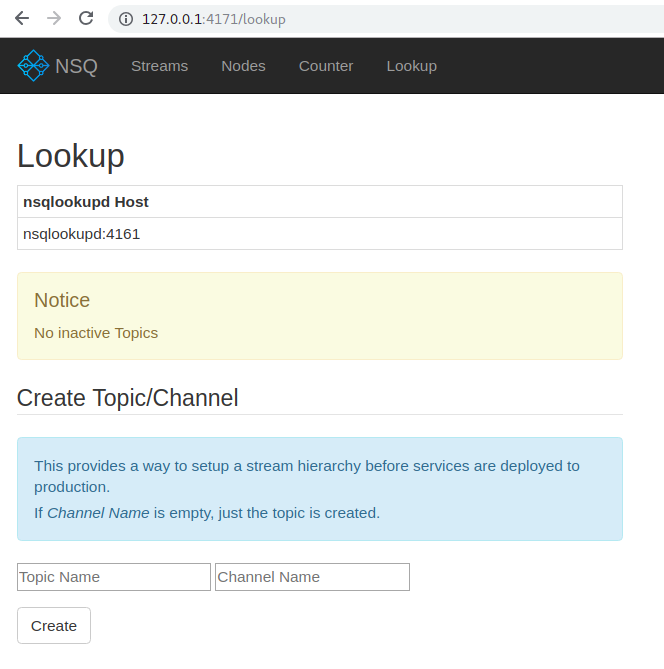
Nsq创建Topic/Channel：

图5.7创建Topic/Channel

图5.7nsq创建topic/channel

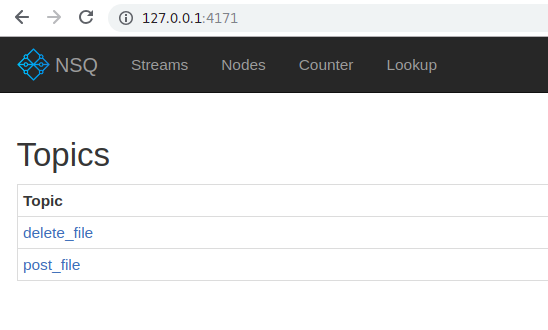
Nsq查看消息队列的Topic：

图5.8查看消息队列

### 5.2.2HDFS运维

通过WebHDFS提供的Web服务，能够更加方便地管理HDFS，比如查看DataNode、NameNode的状态，通过可视化查看HDFS文件系统中有什么文件，文件的具体信息等。

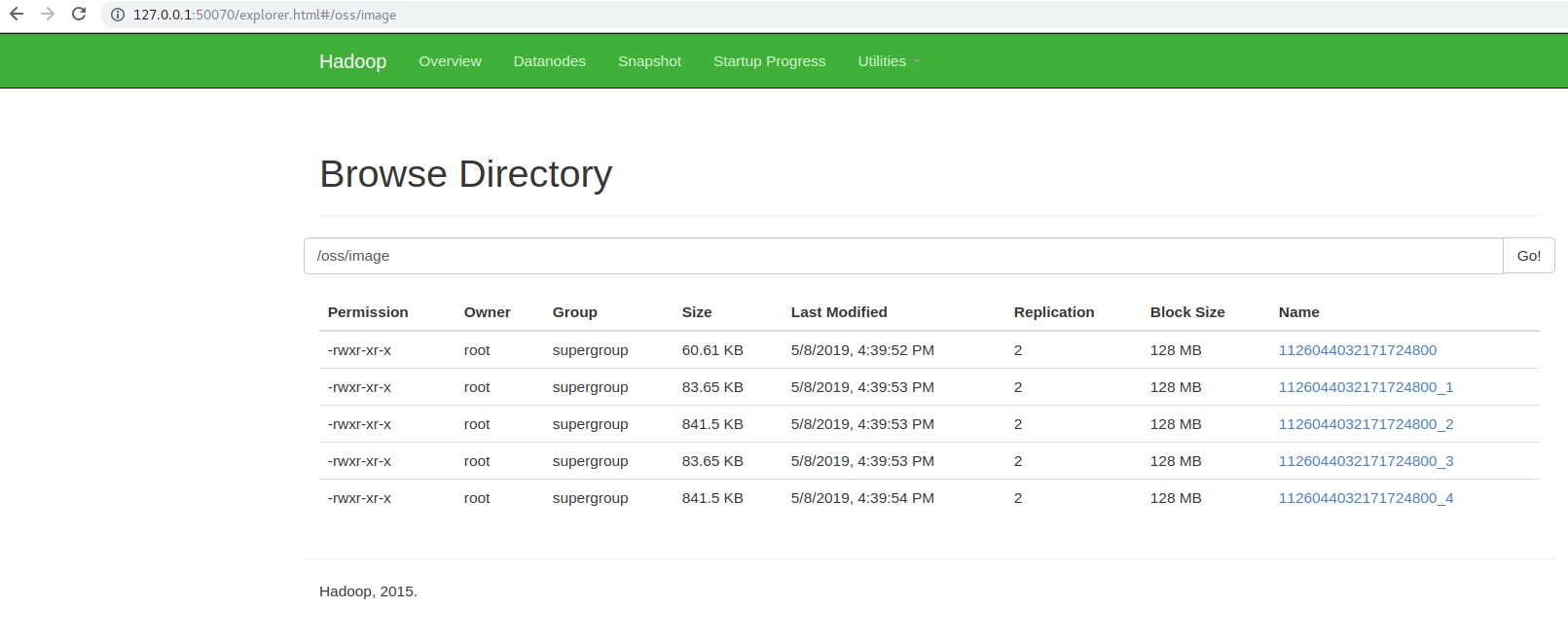
HDFS可视化查看文件系统：

图5.9 HDFS查看文件系统信息

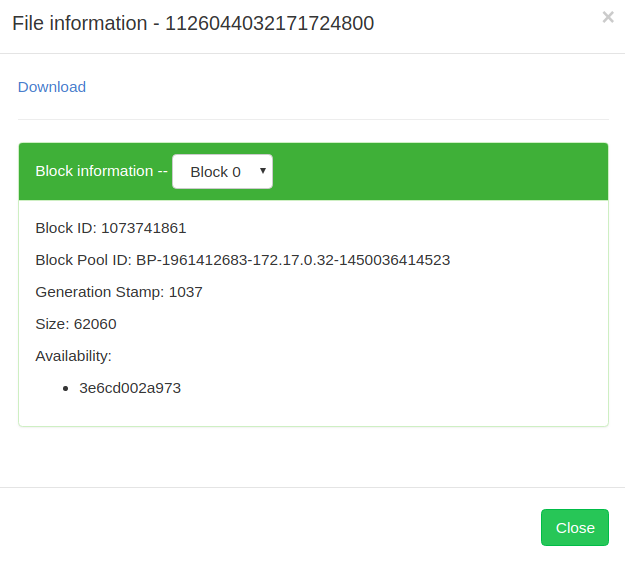
HDFS查看单个文件详情信息：

图5.10 HDFS查看文件详情

## 5.3系统测试

没经过测试的代码是不值得信赖的。本系统采用的测试方法是黑盒测试、白盒测试。调试每一个接口，以及每一个可能的执行路径。对于逻辑正确性测试在开发阶段完成，本节重点讲解吞吐量、系统请求时延两个维度的测试。

测试中，将所有组件、服务都部署在同一物理机器上，本机配置：

1. Intel i7 4710M四核处理器
2. 12GB内存
3. 固态硬盘，mSata3.0

HDFS/NSQ/MySQL/Redis/read\_api/write\_api/consumer均通过docker部署在本地，都是单实例运行。

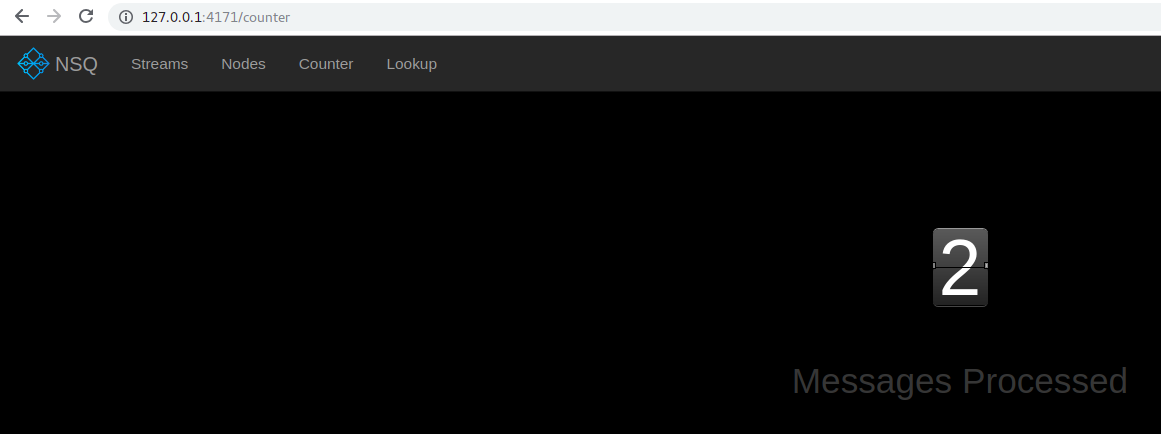
用于测试的图片（大小为24KB）：

图5.11系统测试图片

### 5.3.1上传图片接口

图5.12上传图片接口测试结果

结论：

1. 接口平均处理时延：149.5ms
2. 吞吐量：6.6qps

### 5.3.2删除图片接口

图5.13删除图片接口测试结果

结论：

1. 接口平均处理时延：22.7ms
2. 吞吐量：43.1qps

### 5.3.3下载图片接口

图5.14下载图片接口测试结果

结论：

1. 接口平均处理时延：4.3ms
2. 吞吐量：227.2qps

## 5.4本章小结

本章根据基于异步架构的图片管理系统后端需求和设计给出各个模块的实现细节，通过截图展示实现结果，通过基本测试保证每个模块的逻辑的正确性，以及通过压力测试得出每个接口的处理时延、吞吐量的具体数值。

# 第6章 总结与展望

本研究课题通过前期需求收集、调研业界相关存储系统的设计与实现，完成架构的设计，编写代码完成后进行详细的系统测试，最后撰写本论文。在这过程中，笔者对软件工程有了更加清晰的认识。

## 6.1研究总结

在互联网时代，图片无处不在。云计算服务商对图片垂类支持不够完善，各个企业重复造轮子，且在传统同步处理场景下，处理时延高，高峰期、低峰期系统负载相差甚远，造成资源浪费。基于异步架构的图片管理网站能降低系统的处理时延、增大高峰期吞吐量，本系统正是朝着这目标前进的。

图片管理系统的需求较为简单，实现难点不在于业务逻辑的复杂性，而在于保证系统的非功能需求，提高系统的SLA，通过系统测试测量出系统的吞吐量、处理时延数据。

本系统是一个偏技术型的课题，前期在学习业界相关系统文论和博客等资料的学习上花费了不少时间，确定了整体架构模型后，对每个技术方向进行选型，包括分布式存储、消息队列、缓存、关系型数据库、测试工具和编程语言。本次研究的代码成果在<https://github.com/g10guang/graduation>开源。经过本次系统设计与实现，个人技术深度、广度都得到一定的提升，不光了解到分布式存储、消息队列等组件的实现原理，还动手进行运维部署，并借助组件提供的服务进行开发。

在开发阶段中，发现每次协调多个组件部署等情况十分复杂，每次进行重复性的工作降低开发效率，通过调研发现通过Docker容器化部署和Docker-Compose容器管理工具能够方便地进行组件的管理，通过简单的配置文件解决了组件、服务之间的依赖关系。

在系统测试阶段，调研发现业界提供的测试工具技术方案并不多，笔者决定使用Python编写测试脚本，编写简单的请求、解析、分析代码完成测试。由于在开发中遇到各种问题，实现过程中，花费在测试上的时间并不比编写代码的时间少，在真实项目开发中需要预留足够的时间进行系统测试。

在最后的论文撰写阶段，重新组织语言，将整个系统架构设计做一个描述，从项目背景、需求分析，到系统的设计与实现，包括最后的测试，这个过程不仅是对这个系统做一个总结，更像是重新认识、实现了本系统，对系统的设计有了更深的认识。

## 6.2研究展望

由于硬件资源有限、知识储备不足，本系统的实现最终仅在本地部署，HDFS、Redis、MySQL、NSQ没有实现独立的部署，在单机情况下，很多情况无法模拟，比如网络拥塞、网络丢包，最终系统测试得出的系统处理时延会偏低。

在分布式存储的选型上，为了避免开发中遇到困难，选择业界使用较多的HDFS，HDFS被广泛使用在Hadoop大数据生态，但相对于在线服务，HDFS更加适用于离线的处理场景。分布式存储可替换为Ceph，Ceph在业界中有被用作对象存储在线服务的，处理时延相对HDFS会有优化，而且Ceph除了基本的文件存储还支持对象存储、块存储，HDFS仅支持文件存储，所以需要将更多业务定制信息存储在独立的数据库中，数据分离存储增大系统的维护成本。

# 参考文献

1. 金璞，张仲荣，互联网运营之道[M]，电子工业出版社，2016.
2. 维基百科，消息队列[W].

https://zh.wikipedia.org/wiki/%E6%B6%88%E6%81%AF%E9%98%9F%E5%88%97

1. Michael Mille，云计算[M]，机械工业出版社，2009.
2. Kristina Chodorow， Michael Dirolf，MongoDB: The Definitive Guide[M]，O’Reilly Media, 2011.
3. Introducing Spring Boot2.0 Asynchronous[M].
4. 维基百科，分布式文件系统[W].

https://zh.m.wikipedia.org/zh-sg/%E5%88%86%E6%95%A3%E5%BC%8F%E6%AA%94%E6%A1%88%E7%B3%BB%E7%B5%B1

1. 于君泽，曹洪伟，邱硕，深入分布式缓存：从原理到实践[M]，机械工业出版社，2017.
2. 杨保华，戴玉剑，曹亚仑，Docker技术入门与实战[M]，机械工业出版社，2015.
3. Sam Newman，微服务设计[M]，人民邮电出版社，2016.
4. Tom White, Hadoop[M], O’Reilly Media, 2012.
5. Leonard Richardson，Mike Amundsen，RESTful Web APIs[M]，O’Reilly Media，2014.
6. Jim Gray, Andreas Reuter[M], 事务处理，机械工业出版社，2004.
7. Josiah L. Carlson，Redis in Action[M], 人民邮电出版社，2015.
8. Karan Singh，Ceph Cookbook[M]，电子工业出版社，2016.
9. 倪炜，分布式消息中间件实践[M]，电子工业出版社，2018.
10. Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel. Finding a needle in Haystack: Facebook’s photo storage.OSDI, 2010.

# 致谢

感谢父母在我成长过程中耐心地教育我，给予我更大的自由做选择。

感谢辅导员在大学期间中的指导与帮助，让我大学度过得更加充实。

感谢毕业设计指导老师陈明俊，在系统设计初期提供相关参考文献帮助我整理思路，耐心阅读我的毕业设计论文，并提出改进意见。

感谢身边的同学，在学习、生活中给予我的帮助。