**对象存储服务V1.0版本设计**

目录

[第一章 概述 1](#_Toc516038185)

[1.1 需求 1](#_Toc516038186)

[1.2 指标 2](#_Toc516038187)

[1.3 功能描述 2](#_Toc516038188)

[1.3.1 文件管理 2](#_Toc516038189)

[1.3.2 附加功能 2](#_Toc516038190)

[第二章 总体设计 2](#_Toc516038191)

[2.1 对象存储服务 2](#_Toc516038192)

[2.1.1 技术选型 2](#_Toc516038193)

[2.1.2 技术架构 3](#_Toc516038194)

[2.1.3 功能架构 5](#_Toc516038195)

# 概述

## 概述

随着数据中心以及物联网和移动设备的快速发展，我们已经来到一个信息爆炸的时代，据IDC报告，当下数据以每年50%左右的速度快速增长，截至2020年全球数据规模将达44ZB。如此天量的数据如何进行存储和备份？尤其是对于那些每天会产生海量图片、视频以及文件的公司或者政府机构而言，对象存储服务无疑是解决此类问题的利刃。

## 需求

1. 支持海量文件的存储，文件包括图片、文档、视频等非结构化数据。
2. 文档存储之后能够进行快速访问，通过Rest服务接口或者提供的SDK进行快速访问。
3. 添加权限控制，能够对访问的数据进行控制。

## 实现目标

1. 海量非机构话数据的存储，易扩展。
2. 满足高吞吐量的需要，支持海量用户并发访问，并对外提供高扩展（Scalability）、高持久（Durability）和高可用（Availability）的分布式海量数据存储服务。
3. 具有权限控制的能力。

## 功能描述

### 文件管理

上传、下载、删除、权限控制

### 附加功能

文件预览、接口以及SDK

# 总体设计

## 对象存储服务

对象存储是采用扁平数据组织形式并通过基于http协议的RESTful接口访问的分布式存储系统。数据作为单独的对象存储到一个大容器（Bucket）中，应用通过唯一地址来识别每个单独的数据对象，支持海量数据，高性能，可扩展，高可用。

### 技术选型

HBase可以很好的支持非结构化数据的存储，且基于HDFS保证数据的高可用

HBase 吞吐量非常高，足够支持业务

HBase基于Hadoop集群，不需要重新维护其他数据存储服务

不足：

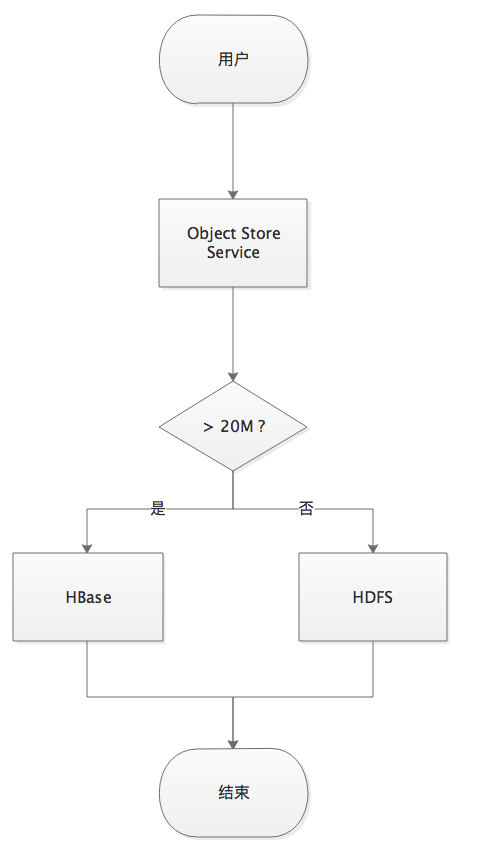
HBase对小文件支持很好，但是大文件无法满足

因为HBase的设计，HBase会发生compact和split操作，文件存储会比较频繁的触发此类耗时操作。

HBase不适合复杂的检索操作，功能上可以受限制

解决方案

HDFS适合存储大文件,将大文件存储到HDFS中， HBase只是存储索引信息



Object Store Service On HBase设计；

Web框架选取Spring Boot框架；

服务本身支持分布式，通过zookeeper实现分布式锁；

提供Restfull API和SDK供用户使用。

#### HBase优化方案

将Hbase的memstore尽量调大，避免文件上传频繁flush

关闭自动major compact功能，改为手动合并，或者写脚本处理

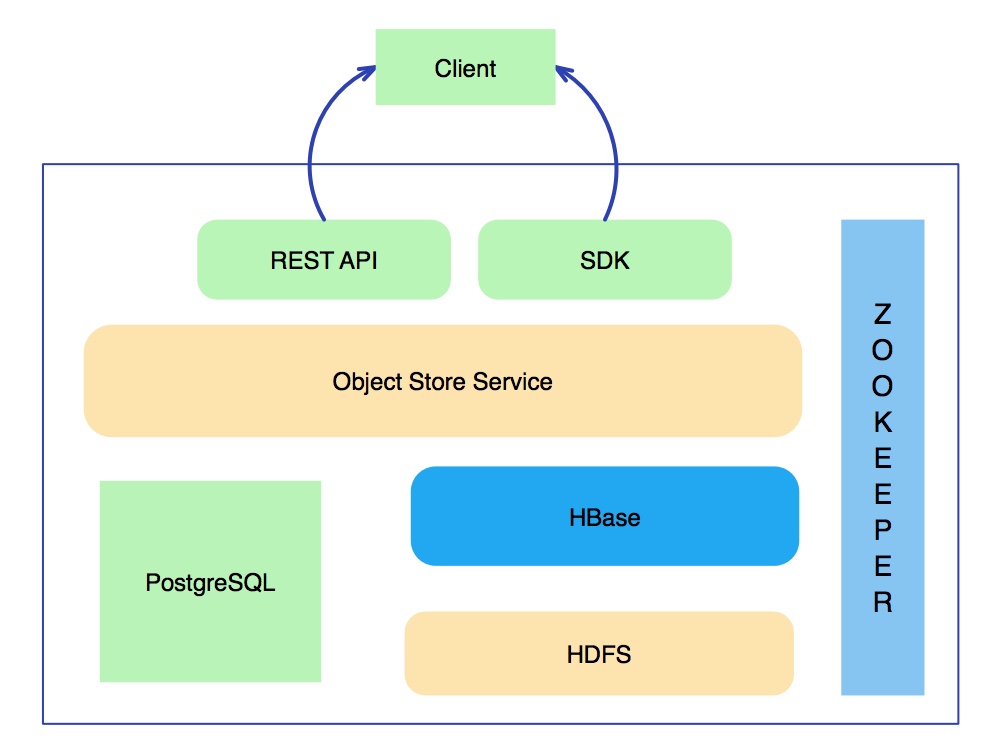
将hbase-site.xml中Region的大小设置的大一些，建表时预选分区。

复杂查询支持

Hbase本身根据字典排序，用户自行在文件夹文件名上做处理，使rowkey能够满足检索需要；

通过Rowkey设计，使其支持起始文件索引，文件前缀匹配等

### 技术架构

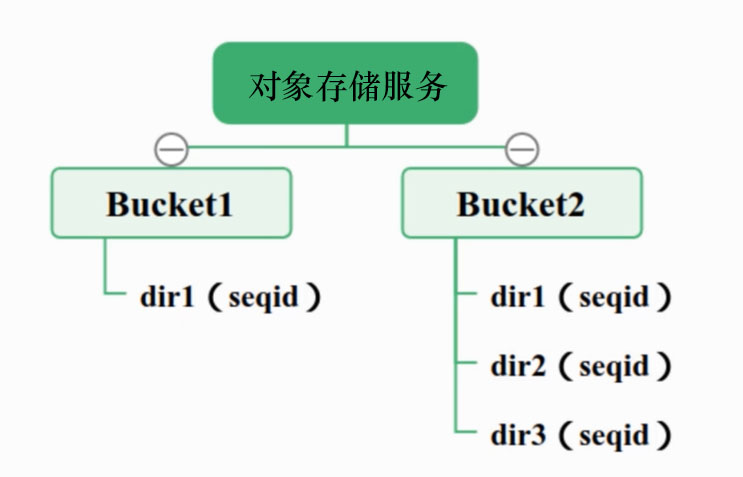


底层存储基于hadoop和hbase，系统本身的数据库采用PostgreSQL，上层为Object Store Service（对象存储服务）服务， hadoop、hbash 和 Object Store service 都需要zookeeper作为支持，在Object Store Service之上，提供Rest API和 SDK的支持。

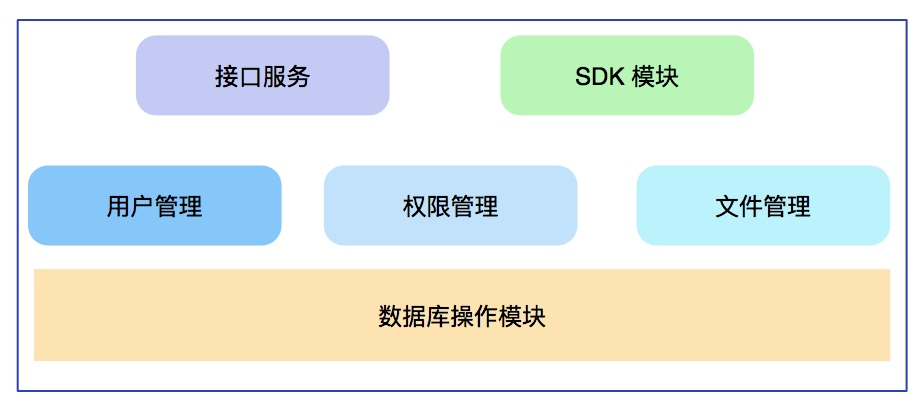
#### 基于zookeeper的分布式锁

采用zookeeper的分布式锁机制实现目录管理和文件管理的创建与删除操作，解决同时对目录以及文件删除操作带来的并发问题。

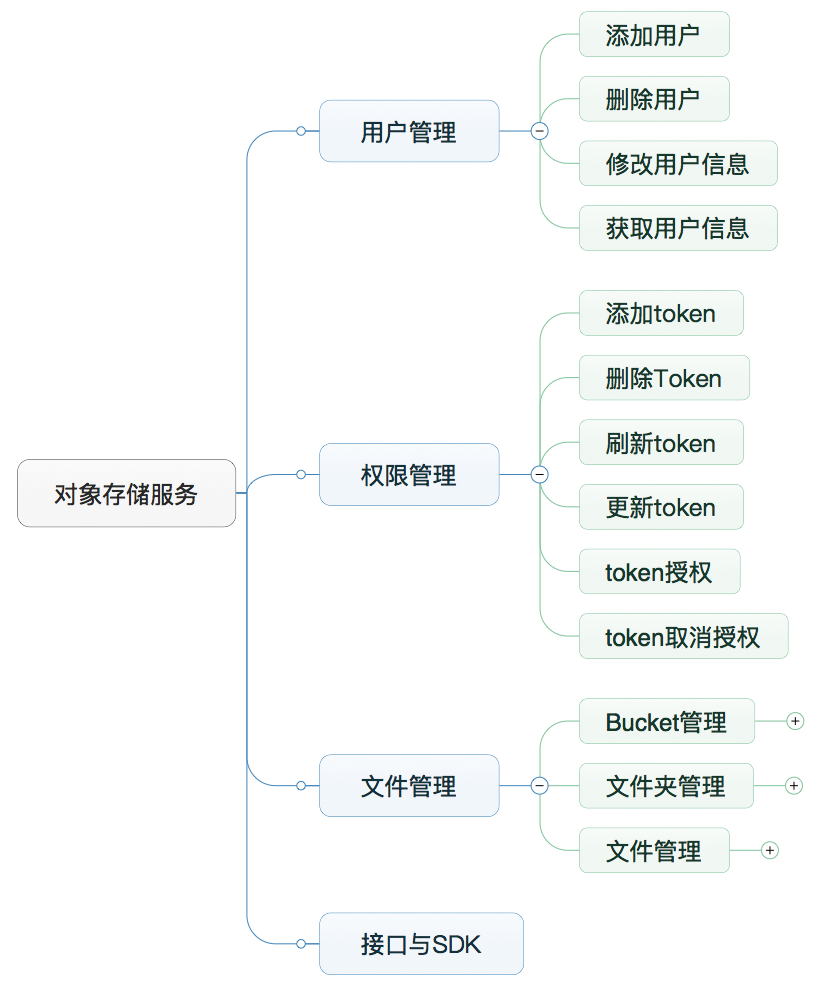
#### 目录结构



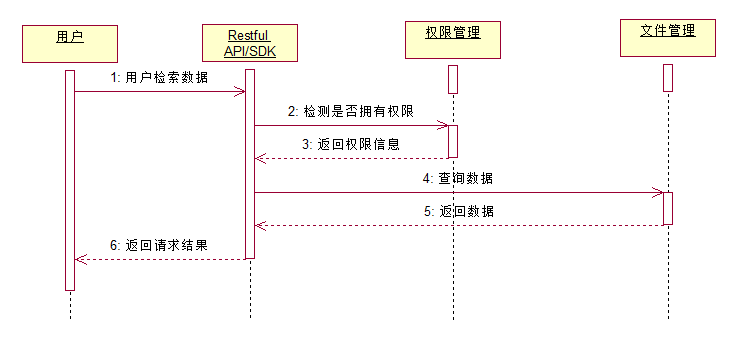
### 功能架构



功能点



文件查询流程



#### 用户管理模块

用户管理表设计

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 描述 |
| USER\_ID | VARCHAR(32) | 主键，用户自己的token |
| USER\_NAME | VARCHAR(32) | 用户名，唯一 |
| PASSWORD | VARCHAR(64) | 用户密码，加密后 |
| SYSTEM\_ROLE | VARCHAR(32) | 用户所属角色 |
| DETAIL | VARCHAR(255) | 描述信息 |
| CREATE\_TIME | TIMESTAMP | 创建时间 |

用户可以添加token，并且设定token的过期时间

可以将bucket的访问权限赋予给某个token

用户创建bucket的时候，默认将自己的ID作为token对自己的授权

创建用户是，添加一条永久的token到TOKEN表

删除用户时，删除用户的TOKEN以及TOKEN与BUCKET的映射

#### 权限管理模块

权限管理基于token

Token表结构设计

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 描述 |
| TOKEN | VARCHAR(32) | 主键，用户自己的token |
| EXPIRE\_TIME | INT | 过期的时间单位，默认1天 |
| REFRESH\_TIME | TIMESTAMP | TOKEN的刷新日期 |
| ACTIVE | TINYINT | TOKEN是否可用 |
| CREATOR | VARCHAR(32) | 创建时间 |

权限表结构设计

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 描述 |
| BUCKET\_NAME | VARCHAR(32) | BUCKET的名称 |
| TARGET\_TOKEN | VARCHAR(32) | 被授权的TOKEN |
| AUTH\_TIME | TIMESTAMP | 授权时间 |

用户可以添加token，并且设定token的过期时间

用户可以经bucket的访问权限授权给某个token

用户创建bucket的时候，默认将自己的ID作为token对自己授权

#### 文件管理模块

Bucket信息存到到关系型数据库，文件以及文件夹存储到Hbase中（为什么放到关系型数据库中，因为对bucket只需要列出基本的信息，并本身和用户以及权限进行挂钩）

文件存储基于Hbase可以快速的读取指定rowkey的文件

可以基于Hbase过滤器实现前缀、起止文件名的过滤操作

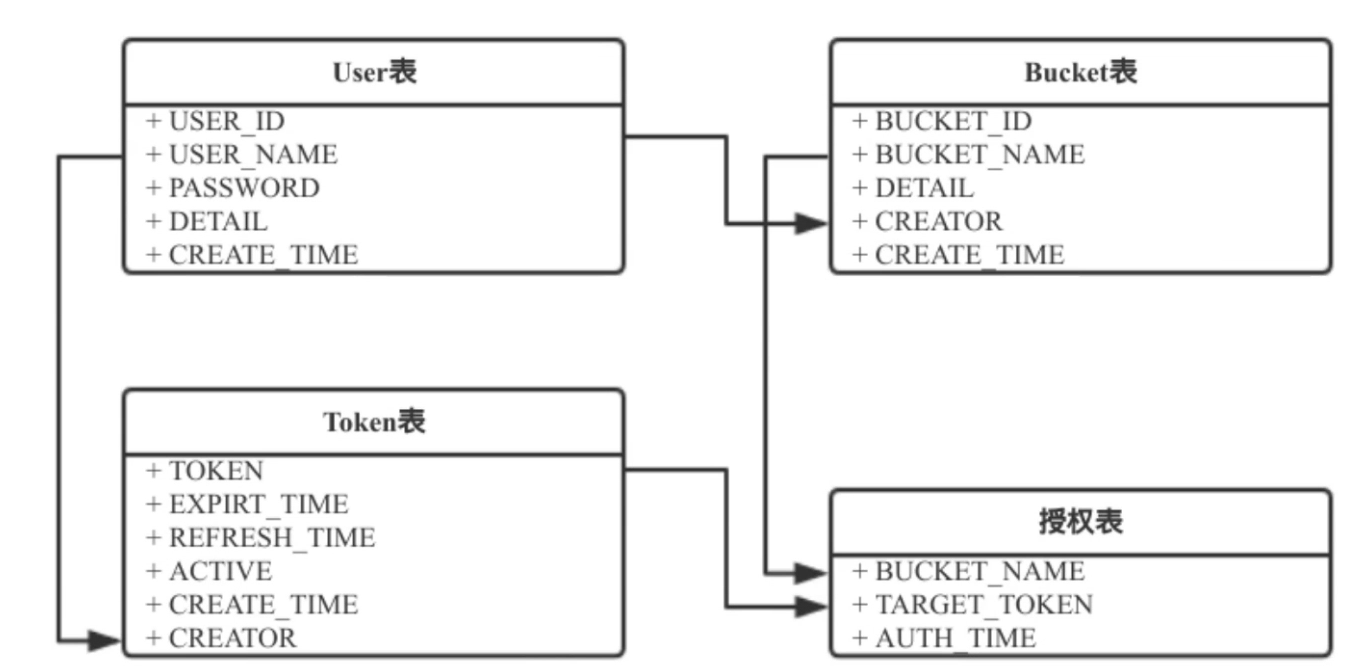
Bucket表结构设计

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 描述 |
| BUCKET\_ID | VARCHAR(32) | BUCKET的ID，主键 |
| BUCKET\_NAME | VARCHAR(32) | BUCKET的名称 |
| DETAIL | VARCHAR(255) | 描述信息 |
| CREATOR | VARCHAR(32) | 创建者 |
| CREATE\_TIME | TIMESTAMP | 创建时间 |

Bucket操作

1. 创建Bucket的时候默认创建两个HBase表，一个为目录表，一个为文件包，并在HDFS创建对应的目录。
2. 删除Bucket的时候删除对应的HBase及HDFS文件夹
3. 访问Bucket时候需要验证Token的权限。

表之间管理关系



##### 文件和文件夹管理



设计思路

1. RowKey：防止热点问题，保证检索效率
2. 列族：名称尽可能的短，不同类型数据分开，不超过2个列族；
3. 将目录和文件分开存储,同时rowkey以文件路径的方式生成来满足检索效率；
4. 创建Bucket时同时会在Hbase中创建目录表和文件表，命名方式以bucket名称结尾（例如os\_dir\_bucket1, os\_obj\_bucket1）

##### 目录表结构

|  |  |
| --- | --- |
| RowKey | Column:Qualifier |
| /dir1 | sub:dir2=1 sub:dir3=1  cf:creator=lisi cf:seqid=0001 |
| /dir1/dir2 | sub:dir4=1 cf:creator=lisi  cf:seqid=0002 |
| /dir1/dir2/dir4 | sub:dir5=1 cf:creator=lisi |

Rowkey为目录的全路径;

Sub的列族为子目录的名称;

Cf的列族是目录的详细信息，包括创建者(creator)和序列值(seqid)

Seqid作用

SeqID 方便检索文件，Hbase支持字典排序，所以文件名是有序的，这时可以对文件可以进保证了有序性，避免了一些热点问题。

##### 文件表结构

|  |  |
| --- | --- |
| RowKey | Column:Qualifier |
| 0001\_file1 | c:content=bytes  cf:creator=lisi cf:size cf:type |
| 0001\_file2 | c:content=bytes  cf:creator=lisi cf:size cf:type |
| 0002\_file3 | c:content=bytes  cf:creator=lisi cf:size cf:type |

Rowkey的设计 采用seqid\_文件名

C列族为文件内容

cf列族为文件信息信息（创建者、大小、类型等）

行起止操作

文件上传操作

1. 判断有无需要上传的文件夹，无则创建；
2. 新建文件夹需要在目录表中插入一条记录外，还需要添加到父目录的列
3. 获取目标文件夹的SeqId,构建file表的RowKey来存储文件。

文件下载操作

1. 根据目标地址获取SeqId，构造file的RowKey
2. 根据file的RowKey快速查询到文件并读取文件内容。

文件浏览过滤操作

1、首选需要读取某文件夹下子文件夹的数据，读取子文件的数据。然后进行数据展示（首页的startkey为parentPath+minbytes，下一页的startkey为第一页的最后的rowkey）

2、Rowkey的设计支持startrow stoprow过滤，支持前缀过滤，

#### seqId生成策略

seqid采用HBase表的方式进行生成

|  |  |
| --- | --- |
| RowKey | Column:Qualifier |
| Bucket1 | s:s=0 |
| Buckert2 | s:s=3 |

Seqid表需要在用户创建Bucket的情况下同时生成表，并插入对应的bucket名称，并初始化对应的值，在进行目录文件上传的时候自动生成并更新。