基于rabbitmq和elasticsearch的分布式对象存储系统

1. **系统概述**

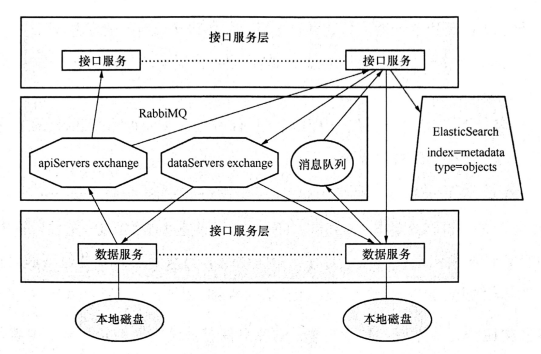


图1 分布式对象存储系统架构

1. 数据服务：数据存储的节点，提供数据的存储功能；
2. 接口服务：提供了对外的REST接口，用以处理客户端请求；
3. Rabbitmq：数据服务与接口服务之间的消息队列；
4. ElasticSearch：提供元数据服务的支持；

接口服务处理客户端的请求，然后向数据服务存取对象，数据服务处理来自接口服务的请求并在本地磁盘上存取对象，他们之间的消息传递通过rabbitmq来实现。

1. **元数据与对象存取规则**
2. 元数据

type MetaData struct{

Name string

Version int

Size int64

Hash string

}

1. 对象存入

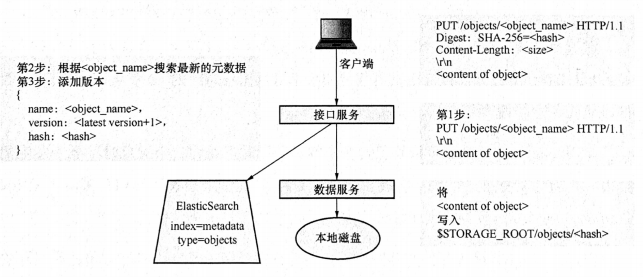


图2 对象存入流程

1. 对象在数据服务节点上存储时应以散列值作为目标对象名进行存储；
2. 每次对象存入都需要在元数据服务器中新增新版本的对象元数据；
3. 对象读取

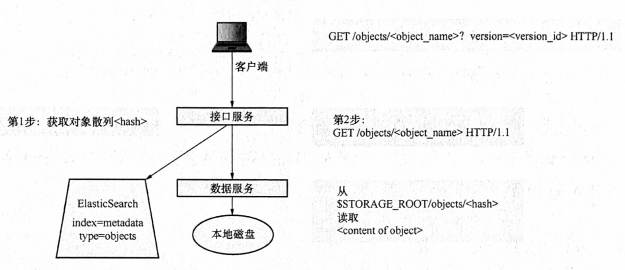


图3 对象获取流程

1. 可提供指定版本的对象读取，若无指定版本则读取最新版本；
2. 对象读取需先从元数据服务器中获取散列值并以此作为目标对象名请求数据服务；
3. 对象删除

删除的操作是在元数据服务器中新增新版本的对象元数据，且此元数据的大小为0、散列值为空，以此作为删除标记。

1. **接口服务**

接口服务需要监听来自数据服务的心跳信息，此功能通过一个循环运行的go协程来实现，同时对外开放objects接口、locate接口、versions接口，分别用于对象存取、对象定位和版本信息获取。

1. 数据服务层心跳检测

接口服务层会绑定名称为apiServers的exchange并持续监听，可通过此收到数据服务层发送到此exchange的心跳包。收到心跳包的数据服务则视为可用，会保存在map容器中，并对超时未收到心跳包的数据服务做移除处理。

1. locate定位信息

接口服务层会检查请求的object对象存在于哪一台数据服务节点中，方式为通过rabbitmq向名为datasServers的exchange发送请求消息，消息体中是请求目标对象的名称。绑定dataServers的消息队列都会收到这个消息，然后在数据服务层做检索处理并返回结果，接口服务层根据返回结果做存在性判断或获取到对应的数据服务节点地址。

1. objects数据存取

接口服务的objects包负责通过rabbitmq与数据服务层通讯，用以存取对象。

其中存入对象操作首先会根据请求头中的hash值作为对象存入数据服务节点时的名称建立地址并随机选择一个数据服务节点，以此创建一个文件写入流putstream，将对象内容通过io.Copy写入到指定文件中，故此文件就被写入到某个数据服务节点所在的设备本地中去。其次将对象的名称、大小以及散列值作为元数据存入元数据服务器es中，es封装的接口会自动为版本号做加1操作，保证此次存入对象为最新对象。

而读取对象则首先则从请求url中获取对象名与版本号，其次在元数据服务中通过对象名查找到响应的元数据从而获取其散列值，然后以此散列值作为对象名调用locate获取到存储此对象的数据服务节点地址，并以此创建文件读取流io.Reader，即可将对象内容读取并保存在response中从而响应对客户端请求。

另外删除操作即通过es搜索到最新版本的对象，将其size与hash置为初始值以此标记为删除。

1. versions版本服务

提供获取对象所有版本的功能，通过es的search功能来实现，并将所有版本作为http响应发送给请求方。

1. **数据服务**

数据服务节点需要循环不断的对接口服务发送心跳包以保持活性，同时需要监听来自接口服务的locate请求以查询本地是否存储目标对象，因此需要两个go协程循环执行来实现。同时数据服务层开放objects接口，用以处理来自接口服务的对象存取的请求。

1. 数据服务层心跳消息

数据服务节点向接口服务的心跳检测模块发送心跳包的方式是向名称为apiServers的exchange发送自己的地址消息，接口服务绑定此exchange的queue都会收到此消息，并从队列中取出消息进行心跳检测处理。

1. locate定位

数据服务为接口服务提供定位检测的服务，因此数据服务节点需要绑定名为dataServers的exchange，接口服务向此exchange发送的定位请求消息会转发到每一个绑定此exchange的数据服务节点。数据服务节点从消息队列中取出目标对象名，即可通过文件存在性判断是否需要向对应接口服务回应自己的地址信息，回应则视为目标对象存储在本数据服务节点，可向接口服务层提供对象读取服务。

1. objects数据存取

数据服务层提供的数据存取功能就是根据文件路径进行文件存取并对请求做出响应。

1. **Rabbitmq：消息队列**
2. 安装与环境设置
3. sudo apt-get install rabbitmq-server 下载并安装rabbitmq-server
4. sudo rabbitmq-plugins enable rabbitmq\_management 下载并安装应用管理工具
5. wget localhost:15672/cli/rabbitmqadmin 下载到当前目录
6. 创建exchange规则：apiServers与dataServers

python3 rabbitmqadmin declare exchange name=apiServers type=fanout

python3 rabbitmqadmin declare exchange name=dataServers type=fanout

所有发往apiServers和dataServers的消息会被转发到所有连到对应exchange的消息队列中。

1. 创建用户

sudo rabbitmqctl add\_user test test

1. 给test用户添加所有访问exchange的权限

sudo rabbitmqctl set\_permissions -p / test ".\*" ".\*" ".\*"

至此，rabbitmq的初始化工作完成，5672将被rabbitmq-server打开并监听请求(netstat -tnlp)

tcp6 0 0 :::5672 :::\* LISTEN -

1. rabbitmq驱动包的封装

代码包下载到本地：go get -v github.com/streadway/amqp

1. New:：创建一个RabbitMQ结构体实例，并为其进行队列初始化声明QueueDeclare
2. Bind：将exchange与当前队列绑定，发往exchange的消息都会在队列中找到
3. Send：将消息单发到指定队列
4. Publish：将消息发送到exchange，从而对所以绑定的队列进行转发
5. Consume：生成一个amqp.Delivery的channel，可从队列中取消息进行处理
6. Close：关闭mq的channel
7. **Elasticsearch：元数据服务器**

元数据指的是对象的描述信息，例如对象的名字、版本、大小以及散列值等，这些都是系统定义的元数据。除了系统定义的元数据以外，用户也可以为这个对象添加自定义的元数据，通常是以键值对的形式保存的任意描述信息，比如一张照片的拍摄时间和拍摄地点，一首歌的作者和演唱者等。本系统为对象增加了元数据处理的功能，可管理对象的名称、版本、大小和散列值，其中，版本管理可以通过获取最新版本的对象从而一定程度上缓和对象的一致性问题（彻底解决需要一致性算法），而大小和散列值可以在删除时作为墓碑。

1. 环境搭建
2. 安装

wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.16.0-amd64.deb

wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.16.0-amd64.deb.sha512

shasum -a 512 -c elasticsearch-7.16.0-amd64.deb.sha512

sudo dpkg -i elasticsearch-7.16.0-amd64.deb

1. 启动

sudo systemctl enable elasticsearch.service

sudo systemctl start elasticsearch.service

至此es启动，es的运行端口9200与9300被打开并监听请求(netstat -tnlp)：

tcp6 0 0 ::1:9200 :::\* LISTEN -

tcp6 0 0 ::1:9300 :::\* LISTEN -

也可通过curl 127.0.0.1:9200/ -v进行请求，es会给出正确回应。

1. 请求url
2. 查询目标对象的元数据

"http://<ES\_SERVER>/metadata/objects/<name\_versionId>/\_source”

1. 获取目标对象最新版本的元数据

"http://<ES\_SERVER>/metadata/\_search?q=name:<name>&size=1&sort=version:desc"

1. 上传文件元数据

"http://<ES\_SERVER>/metadata/objects/<name\_versionId>?op\_type=create"

1. 查询所有元数据

"http://<ES\_SERVER>/metadata/\_search?sort=name,version&from=<from>&size=<size>"

1. 删除元数据

"http://<ES\_SERVER>/metadata/objects/<name\_versionId>"

1. **接口**
2. GET:/objects/<object\_name>?version=<version\_id>
3. PUT

URL:/objects/<object\_name>

Header:-Digest:SHA-256=<对象散列值的Base64编码>

Header:-Content-Length:<对象内容长度>

Header:-Content-Type: application/json

1. DELETE:/objects/<object\_name>
2. GET:/versions/
3. GET:/versions/<object\_name>
4. **演示**
5. 绑定多个地址，模拟多台设备
6. ifconfig确认本机网路接口名ens33，本机ip:192.168.187.129
7. 执行地址绑定

其中，数据服务节点6个：

sudo ifconfig ens33:1 192.168.1.1

sudo ifconfig ens33:2 192.168.1.2

sudo ifconfig ens33:3 192.168.1.3

sudo ifconfig ens33:4 192.168.1.4

sudo ifconfig ens33:5 192.168.1.5

sudo ifconfig ens33:6 192.168.1.6

接口服务节点1个

sudo ifconfig ens33:7 192.168.2.1

执行sh bindnewaddress.sh即可

1. 启动元服务，创建metadata索引以及objects类型的映射（putmapping.sh）

sudo systemctl enable elasticsearch.service

sudo systemctl start elasticsearch.service

curl localhost:9200/metadata -H "Content-Type: application/json" -XDELETE

curl localhost:9200/metadata -H "Content-Type: application/json" -XPUT-d'

{

"mappings": {

"properties": {

"name": {

"type": "text",

"index": "false"

},

"version": {

"type": "integer"

},

"size": {

"type": "integer"

},

"hash": {

"type": "text"

}

}

}

}'

即执行sh putmapping.sh即可

1. 执行 sh startenv.sh，包括创建存储目录以及开启相应的服务。

for i in `seq 1 6`;do mkdir -p ../src/tmp/$i/objects;done

export RABBITMQ\_SERVER=amqp://test:test@localhost:5672

export ES\_SERVER=localhost:9200

LISTEN\_ADDRESS=192.168.1.1:12345 STORAGE\_ROOT=../src/tmp/1 go run ../src/dataServer/dataServer.go &

LISTEN\_ADDRESS=192.168.1.2:12345 STORAGE\_ROOT=../src/tmp/2 go run ../src/dataServer/dataServer.go &

LISTEN\_ADDRESS=192.168.1.3:12345 STORAGE\_ROOT=../src/tmp/3 go run ../src/dataServer/dataServer.go &

LISTEN\_ADDRESS=192.168.1.4:12345 STORAGE\_ROOT=../src/tmp/4 go run ../src/dataServer/dataServer.go &

LISTEN\_ADDRESS=192.168.1.5:12345 STORAGE\_ROOT=../src/tmp/5 go run ../src/dataServer/dataServer.go &

LISTEN\_ADDRESS=192.168.1.6:12345 STORAGE\_ROOT=../src/tmp/6 go run ../src/dataServer/dataServer.go &

LISTEN\_ADDRESS=192.168.2.1:12345 go run ../src/apiServer/apiServer.go &

netstat -tnlp查看服务启动情况

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 192.168.1.4:12345 0.0.0.0:\* LISTEN 50347/dataServer

tcp 0 0 192.168.1.5:12345 0.0.0.0:\* LISTEN 50330/dataServer

tcp 0 0 192.168.187.1:12345 0.0.0.0:\* LISTEN 50348/apiServer

tcp 0 0 192.168.1.2:12345 0.0.0.0:\* LISTEN 50351/dataServer

tcp 0 0 192.168.1.6:12345 0.0.0.0:\* LISTEN 50346/dataServer

tcp 0 0 192.168.1.1:12345 0.0.0.0:\* LISTEN 50340/dataServer

tcp 0 0 192.168.1.3:12345 0.0.0.0:\* LISTEN 50329/dataServer

数据服务节点：6

接口服务节点：1

1. 上传对象
2. 计算散列值

echo -n "this is object test1" | openssl dgst -sha256 -binary | base64

IOauwRgpTIu9pF6S4rBdoc9FI7bjs+QXkAaz+I0WlKk=

1. Curl上传

curl -H "Digest: SHA-256=IOauwRgpTIu9pF6S4rBdoc9FI7bjs+QXkAaz+I0WlKk=" -v 192.168.2.1:12345/objects/test1 -XPUT -d"this is object test1"

1. 上传新版本

echo -n "this is object test1 version 2" | openssl dgst -sha256 -binary | base64

djUaW64po2ocPsdZD7w1Egjz55obF4moycXWce6Ac9U=

curl -v 192.168.2.1:12345/objects/test1 -XPUT -d"this is object test1 version 2" -H "Digest: SHA-256=djUaW64po2ocPsdZD7w1Egjz55obF4moycXWce6Ac9U="

1. 查看
2. 所有版本

curl 192.168.2.1:12345/versions/test1

{"Name":"test1","Version":1,"Size":20,"Hash":"IOauwRgpTIu9pF6S4rBdoc9FI7bjs+QXkAaz+I0WlKk="}

{"Name":"test1","Version":2,"Size":30,"Hash":"djUaW64po2ocPsdZD7w1Egjz55obF4moycXWce6Ac9U="}

1. 最新版本

curl 192.168.2.1:12345/objects/test1

1. 指定版本

curl 192.168.2.1:12345/objects/test1?Version=2

1. 定位

curl 192.168.2.1:12345/locate/djUaW64po2ocPsdZD7w1Egjz55obF4moycXWce6Ac9U=

"192.168.1.2:12345"

1. 删除

curl -v 192.168.2.1:12345/objects/test1 -XDELETE

curl 192.168.2.1:12345/objects/test1 不存在，无返回值

curl 192.168.2.1:12345/versions/test1

{"Name":"test1","Version":1,"Size":20,"Hash":"IOauwRgpTIu9pF6S4rBdoc9FI7bjs+QXkAaz+I0WlKk="}

{"Name":"test1","Version":2,"Size":30,"Hash":"djUaW64po2ocPsdZD7w1Egjz55obF4moycXWce6Ac9U="}