package com.yingchong.hive.udtf;

import java.util.List;

import com.google.common.collect.Lists;

import org.apache.hadoop.hive.ql.exec.UDFArgumentException;

import org.apache.hadoop.hive.ql.metadata.HiveException;

import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;

import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;

import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;

import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;

import org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

public class ArrToMapUDTF extends GenericUDTF {

​

  private String[] obj = new String[2];

​

   /\*\*

    \* 返回类型为 String，string

    \*

    \* @param argOIs

    \* @return

    \* @throws UDFArgumentException

    \*/

  @Override

  public StructObjectInspector initialize(StructObjectInspector argOIs) throws UDFArgumentException {

      List<String> colName = Lists.newLinkedList();

      colName.add("key");

      colName.add("value");

      List<ObjectInspector> resType = Lists.newLinkedList();

      resType.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

      resType.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

      return ObjectInspectorFactory.getStandardStructObjectInspector(colName, resType);

  }

​

  @Override

  public void process(Object[] args) throws HiveException {

      if(args[0] == null) {

          return;

      }

      String arg1 = args[0].toString();

      String[] arr1 = arg1.split(",");

      String[] arr2 = null;

      if(args[1] != null) {

          arr2 = args[1].toString().split(",");

      }

      for(int i = 0; i < arr1.length ; i++ ) {

          obj[0] = arr1[i];

          if(arr2 != null && arr2.length > i) {

              obj[1] = arr2[i];

          } else {

              obj[1] = null;

          }

          forward(obj);

      }

  }

​

  @Override

  public void close() throws HiveException {

  }

}

package com.yingchong.udtf;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.Set;

import org.apache.hadoop.hive.ql.exec.UDFArgumentException;

import org.apache.hadoop.hive.ql.exec.UDFArgumentLengthException;

import org.apache.hadoop.hive.ql.metadata.HiveException;

import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;

import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;

import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;

import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;

import org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

import com.alibaba.fastjson.JSONArray;

import com.alibaba.fastjson.JSONObject;

public class Redis\_Gmp extends GenericUDTF{

@Override

public void close() throws HiveException {

// TODO Auto-generated method stub

}

@Override

public void process(Object[] arg0) throws HiveException {

String input = arg0[0].toString();

String[] split = input.split(",\\[");

String content\_id = split[0].replace("(", "");

String json\_str = split[1].replace(")", "");

json\_str = "["+json\_str;

JSONArray json\_arr = JSONArray.parseArray(json\_str);

for(int i =0 ;i < json\_arr.size();i++){

String[] result = new String[5];

result[0] = content\_id;

JSONObject ele = json\_arr.getJSONObject(i);

Set<String> ks = ele.keySet();

for(String k : ks){

result[1] = k;

}

result[2] = ele.getJSONObject(result[1]).getString("click");

result[3] = ele.getJSONObject(result[1]).getString("impression");

result[4] = ele.getJSONObject(result[1]).getString("ctr");

forward(result);

//System.out.println(result[0] + " " + result[1] + " " + result[2] +" "+result[3]+" "+result[4]);

}

}

@Override

public StructObjectInspector initialize(ObjectInspector[] args)

throws UDFArgumentException {

if (args.length != 1) {

throw new UDFArgumentLengthException(

"Redis\_Gmp takes only one argument");

}

if (args[0].getCategory() != ObjectInspector.Category.PRIMITIVE) {

throw new UDFArgumentException(

"Redis\_Gmp takes string as a parameter");

}

ArrayList<String> fieldNames = new ArrayList<String>();

ArrayList<ObjectInspector> fieldOIs = new ArrayList<ObjectInspector>();

fieldNames.add("content\_id");

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

fieldNames.add("app");

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

fieldNames.add("click");

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

fieldNames.add("impression");

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

fieldNames.add("ctr");

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

return ObjectInspectorFactory.getStandardStructObjectInspector(

fieldNames, fieldOIs);

}

public static void main(String[] args) throws HiveException {

//(119962233,[{"-1":{"impression":150,"click":0.3438084,"ctr":0.006597938}},{"coolpad":{"impression":56,"click":0.3438084,"ctr":0.018344998}},{"emui":{"impression":64,"click":0,"ctr":0}}])

//(98337176,[{"-1":{"impression":167,"click":0.9933209,"ctr":0.02424849}},{"ali":{"impression":163,"click":0.9933209,"ctr":0.025131164}}])

Redis\_Gmp redis\_gmp = new Redis\_Gmp();

String s1 = "(98337176,[{\"-1\":{\"impression\":167,\"click\":0.9933209,\"ctr\":0.02424849}},{\"ali\":{\"impression\":163,\"click\":0.9933209,\"ctr\":0.025131164}}])";

String s2 = "(119962233,[{\"-1\":{\"impression\":150,\"click\":0.3438084,\"ctr\":0.006597938}},{\"coolpad\":{\"impression\":56,\"click\":0.3438084,\"ctr\":0.018344998}},{\"emui\":{\"impression\":64,\"click\":0,\"ctr\":0}}])";

Object[] arg0 = new Object[]{s2};

redis\_gmp.process(arg0);

}

}

package com.yingchong.service.yccloud.storage.service;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONObject;

import com.google.gson.Gson;

import com.yingchong.common.yccloud.commonapi.constants.SourceLimitConstants;

import com.yingchong.common.yccloud.commonapi.mapper.StorageVolumeMapper;

import com.yingchong.common.yccloud.commonapi.model.StorageVolume;

import com.yingchong.common.yccloud.commonapi.util.RedisUtils;

import com.yingchong.common.yccloud.remote.bean.ResponseBean;

import com.yingchong.common.yccloud.remote.service.monitor.LimitRemoteService;

import com.yingchong.service.yccloud.storage.util.StorageErrorCode;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Pointcut;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.context.request.RequestAttributes;

import org.springframework.web.context.request.RequestContextHolder;

import org.springframework.web.context.request.ServletRequestAttributes;

import javax.servlet.http.HttpServletRequest;

@Aspect

@Configuration

public class VolumeAspect {

private static final Logger logger = LoggerFactory.getLogger(VolumeAspect.class);

@Autowired

private LimitRemoteService limitRemoteService;

@Autowired

private RedisUtils redisUtils;

@Autowired

private StorageVolumeMapper volumeMapper;

// Pointcut

@Pointcut("execution(\* com.yingchong.service.yccloud.storage.service.VolumeStorageService.\*(..))")

public void excudeService() {

}

@Around("excudeService()")

public Object doAround(ProceedingJoinPoint pjp) throws Throwable {

RequestAttributes ra = RequestContextHolder.getRequestAttributes();

ServletRequestAttributes sra = (ServletRequestAttributes) ra;

HttpServletRequest request = sra.getRequest();

String url = request.getRequestURL().toString();

String method = request.getMethod();

String uri = request.getRequestURI();

String queryString = request.getQueryString();

//Map<String, String[]> parameterMap = request.

String sourceLimitSwitch = redisUtils.get(SourceLimitConstants.sourceLimitSwitch);

if("1".equals(sourceLimitSwitch)){

Object[] args = pjp.getArgs();

String methodName = pjp.getSignature().getName();

logger.info("请求开始, 各个参数, url: {}, method: {}, uri: {}, params: {}, methodName:{}", url, method, uri, queryString, methodName);

if (methodName.equals("createStorageVolumenAssetCost")) {//创建块存储

//CreateContainerParam createContainerParam = ((CreateContainerParam)args[0]);

int size = Integer.parseInt(String.valueOf(args[1])); // 共享存储大小

String userId = String.valueOf(args[2]);

ResponseBean<Boolean> responseBean = limitRemoteService.checkCanCreateVolume(userId, 1,size);

if (!responseBean.getData()) {//可以创建

logger.error("错误原因:{}",responseBean.getRetMsg());

ResponseBean<Integer> createVolumeResponse = new ResponseBean<>();

createVolumeResponse.setCodeAndMsg(responseBean.getRetCode(),responseBean.getRetMsg());

return createVolumeResponse;

}else {

Object result = pjp.proceed();

ResponseBean<Integer> response = (ResponseBean<Integer>) result;

if (response.getRetCode().equals(StorageErrorCode.SUCCESS.getCode())) {

limitRemoteService.updateCreateVolumeLimit(userId, 1, size);

}

return result;

}

}

if (methodName.equals("deleteVolume")) {//删除共享存储

Integer volumeId = Integer.parseInt(String.valueOf(args[0]));

Object result = pjp.proceed();

ResponseBean<Boolean> response = (ResponseBean<Boolean>) result;

if (response.getRetCode().equals(StorageErrorCode.SUCCESS.getCode())) {

StorageVolume storageVolume = volumeMapper.selectByPrimaryKey(volumeId);

limitRemoteService.updateCreateVolumeLimit(

storageVolume.getUserId(),

-1,

-storageVolume.getVolumeSize()/1024);

}

return result;

}

if (methodName.equals("updateVolume")) {//扩展共享存储

String userId = String.valueOf(args[0]);

Integer volumeId = Integer.parseInt(String.valueOf(args[1]));

Integer size = Integer.parseInt(String.valueOf(args[2]))/1024; //单位-G

StorageVolume storageVolume = volumeMapper.selectByPrimaryKey(volumeId);

int oldSize = storageVolume.getVolumeSize()/1024;

if (size < oldSize) {

logger.error("共享卷不能缩小,只能扩大");

ResponseBean<Boolean> createVhdImageResponse = new ResponseBean<>();

createVhdImageResponse.setCodeAndMsg(StorageErrorCode.CODE\_010106.getCode(),StorageErrorCode.CODE\_010106.getMsg());

return createVhdImageResponse;

}

ResponseBean<Boolean> responseBean = limitRemoteService.checkCanCreateVolume(userId, 0, size-oldSize);

if (!responseBean.getData()) {//返回值和源接口一直,直接返回

return responseBean;

}

}

if (methodName.equals("updateVolumeCallback")) {//更新共享存储回调

JSONObject result = (JSONObject) args[0];

JSONObject r = result.getJSONObject("result");

JSONObject jsonBaseParam = result.getJSONObject("baseParam");

StorageVolume storageVolume = JSON.toJavaObject( jsonBaseParam.getJSONObject("storageVolume"), StorageVolume.class);

// Map<String, Object> baseParamMap = r.getJSONObject("baseParam");

// StorageVolume storageVolume = (StorageVolume) baseParamMap.get("storageVolume");

//String costPurchase = baseParamMap.get("costPurchase").toString();

Object oldSize = jsonBaseParam.get("oldSize");

int old = Integer.parseInt(String.valueOf(oldSize==null?"0":oldSize));

if (StorageErrorCode.SUCCESS.getCode().equals(r.getString("status"))) {

limitRemoteService.updateCreateVolumeLimit(storageVolume.getUserId(),0,(storageVolume.getVolumeSize() - old)/1024 );

}

}

}

// result的值就是被拦截方法的返回值

Object result = pjp.proceed();

Gson gson = new Gson();

logger.info("请求结束，controller的返回值是 " + gson.toJson(result));

return result;

}

}

package com.yingchong.service.yccloud.storage.service;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONObject;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import com.yingchong.common.yccloud.commonapi.bean.NoticeBean;

import com.yingchong.common.yccloud.commonapi.bean.RabbitmqParam;

import com.yingchong.common.yccloud.commonapi.constants.Constants;

import com.yingchong.common.yccloud.commonapi.mapper.CloudVolumeContainerContactMapper;

import com.yingchong.common.yccloud.commonapi.mapper.StorageVolumeMapper;

import com.yingchong.common.yccloud.commonapi.model.\*;

import com.yingchong.common.yccloud.commonapi.util.DateUtil;

import com.yingchong.common.yccloud.commonapi.util.MD5;

import com.yingchong.common.yccloud.commonapi.util.ZkLockUtil;

import com.yingchong.common.yccloud.remote.bean.ResponseBean;

import com.yingchong.common.yccloud.remote.bean.scheduler.ResourceSchedulerBean;

import com.yingchong.common.yccloud.remote.bean.storage.OutTimeVolume;

import com.yingchong.common.yccloud.remote.service.scheduler.SchedulerRemoteService;

import com.yingchong.common.yccloud.remote.service.transaction.AccountRemoteService;

import com.yingchong.common.yccloud.remote.service.transaction.AssetCostRemoteService;

import com.yingchong.service.yccloud.storage.mapper.MyStorageVolumeMapper;

import com.yingchong.service.yccloud.storage.util.StorageDataUtil;

import com.yingchong.service.yccloud.storage.util.StorageDateUtil;

import com.yingchong.service.yccloud.storage.util.StorageErrorCode;

import org.apache.commons.collections.map.HashedMap;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.recipes.locks.InterProcessMutex;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.amqp.rabbit.core.RabbitTemplate;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import javax.annotation.Resource;

import java.math.BigDecimal;

import java.util.\*;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description : 共享存储服务

\* ========================================================

\*/

@Service

public class VolumeStorageService {

private static final Logger logger = LoggerFactory.getLogger(VolumeStorageService.class);

@Autowired

private StorageVolumeMapper storageVolumeMapper;

@Autowired

private CloudVolumeContainerContactMapper cloudVolumeContainerContactMapper;

@Autowired

private MyStorageVolumeMapper myStorageVolumeMapper;

@Autowired

private CuratorFramework framework;

@Autowired

RabbitTemplate rabbitTemplate;

@Autowired

SchedulerRemoteService schedulerRemoteService;

@Resource(name = "volumePath")

String volumePath;

@Autowired

StorageDataUtil storageDataUtil;

@Autowired

StorageDateUtil storageDateUtil;

@Autowired

AssetCostRemoteService assetCostRemoteService;

@Autowired

AccountRemoteService accountRemoteService;

/\*\*

\* 查询这个共享存储是否被绑定

\*

\* @param volumeId

\* @return

\*/

public ResponseBean<Boolean> isHasContact(Integer volumeId) {

CloudVolumeContainerContactExample selEvcc = new CloudVolumeContainerContactExample();

selEvcc.createCriteria().andVolumeIdEqualTo(volumeId);

List<CloudVolumeContainerContact> cvccList = cloudVolumeContainerContactMapper.selectByExample(selEvcc);

if (cvccList != null && !cvccList.isEmpty()) {

return new ResponseBean(StorageErrorCode.CODE\_010105.getCode(), StorageErrorCode.CODE\_010105.getMsg());

}

return new ResponseBean(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

/\*\*

\* 查询共享存储

\*

\* @param userId

\* @return

\*/

public ResponseBean<PageInfo<Map<String, Object>>> findVolumeStorageByUser(String tel, String userId, Integer page, Integer pageNum, String dateSta, String dateEnd, Integer status, String keyword) {

ResponseBean<PageInfo<Map<String, Object>>> retBean = new ResponseBean<>();

PageHelper.startPage(page, pageNum, "update\_time DESC");

Map<String, Object> param = new HashMap<>();

param.put("userId", userId);

param.put("dateSta", dateSta);

param.put("dateEnd", dateEnd);

param.put("status", status);

param.put("keyword", keyword);

param.put("tel", tel);

try {

List<Map<String, Object>> dataList = myStorageVolumeMapper.findStorageVolume(param);

for (Map<String, Object> data : dataList) {

if (!data.containsKey("webName")) {

data.put("webName", data.get("default\_mount\_path"));

}

if (Integer.valueOf( data.get("end\_time\_day").toString())<=0 ){

data.put("end\_time\_day\_str","已到期");

}

if (Integer.valueOf( data.get("end\_time\_day").toString())>0 && Integer.valueOf( data.get("end\_time\_day").toString())<=30 ){

data.put("end\_time\_day\_str",data.get("end\_time\_day")+"天后到期");

}

}

PageInfo<Map<String, Object>> pageInfo = new PageInfo<>(dataList);

retBean.setData(pageInfo);

retBean.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

} catch (Exception e) {

return new ResponseBean<>(StorageErrorCode.CODE\_010101.getCode(), StorageErrorCode.CODE\_010101.getMsg());

}

return retBean;

}

/\*\*

\* 更新共享存储状态

\*

\* @param volumeId

\* @param type

\* @return

\*/

public ResponseBean<Boolean> updateVolumeStatus(Integer volumeId, Integer type) {

StorageVolume storageVolume = new StorageVolume();

storageVolume.setId(volumeId);

ResponseBean<Boolean> data = new ResponseBean<>();

int status = storageVolumeMapper.selectByPrimaryKey(volumeId).getStatus();

if (type.intValue() == Constants.UNUSED) {

//判断是否可以恢复

if (status == Constants.DELETED) {

storageVolume.setStatus(Constants.UNUSED);

storageVolumeMapper.updateByPrimaryKeySelective(storageVolume);

return new ResponseBean(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

} else if (type.intValue() == Constants.REAL\_DELETE) {

if (status == Constants.DELETED) { //判断是否可以删除

ResponseBean<Boolean> rb = deleteVolume(volumeId, false);

if ("000000".equals(rb.getRetCode())) {

return new ResponseBean(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

}

}

return new ResponseBean(StorageErrorCode.CODE\_010105.getCode(), StorageErrorCode.CODE\_010105.getMsg());

}

/\*\*

\* 创建共享存储 只在表中插入数据

\*

\* @param userId

\* @param name

\* @param size

\* @param endTime

\* @return

\*/

public ResponseBean<Integer> createVolumeStorage(String userId, String name, Integer size, String mountPath, Long endTime) {

ResponseBean<Integer> data = new ResponseBean<Integer>();

InterProcessMutex lock = ZkLockUtil.getInterProcessMutex(framework, "/" + MD5.encode("addVolumeStorage " + userId + name));

try {

lock.acquire();

StorageVolumeExample sve = new StorageVolumeExample();

sve.createCriteria().andVolumeNameEqualTo(name).andUserIdEqualTo(userId);

List<StorageVolume> svmList = storageVolumeMapper.selectByExample(sve);

// 验证名称是否重复

if (null != svmList && !svmList.isEmpty()) {

data.setCodeAndMsg(StorageErrorCode.CODE\_010203.getCode(), StorageErrorCode.CODE\_010203.getMsg());

} else {

String iName = MD5.uuid();

StorageVolume sv = new StorageVolume();

sv.setUserId(userId);

sv.setVolumeName(name);

sv.setVolumeNameText("/" + userId + "/volume/" + iName);

sv.setVolumeSize(size);

sv.setVolumePath(volumePath);

sv.setStatus(Constants.UNUSED);

sv.setMountPath(mountPath);

Date date = new Date();

sv.setCreateTime(date);

sv.setUpdateTime(date);

sv.setEndTime(new Date(endTime));

storageVolumeMapper.insert(sv);

data.setData(sv.getId());

data.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

} catch (Exception e) {

logger.error("发生异常", e);

data.setCodeAndMsg(StorageErrorCode.CODE\_010102.getCode(), StorageErrorCode.CODE\_010102.getMsg());

} finally {

try {

lock.release();

} catch (Exception e) {

logger.error("释放锁异常", e);

}

}

return data;

}

/\*\*

\* 删除共享存储 PhysDeletion 是否真实删除

\*

\* @param volumeId

\* @return

\*/

public ResponseBean<Boolean> deleteVolume(Integer volumeId, Boolean PhysDeletion) {

try {

CloudVolumeContainerContactExample cvExample = new CloudVolumeContainerContactExample();

cvExample.createCriteria().andVolumeIdEqualTo(volumeId);

List<CloudVolumeContainerContact> cloudVolumeList = cloudVolumeContainerContactMapper.selectByExample(cvExample);

//判断共享存储是否绑定

if (null != cloudVolumeList && !cloudVolumeList.isEmpty()) {

return new ResponseBean<>(StorageErrorCode.CODE\_010105.getCode(), StorageErrorCode.CODE\_010105.getMsg());

} else {

if (PhysDeletion) {

StorageVolume sv = new StorageVolume();

// 删除

sv.setStatus(Constants.DELETED);

sv.setId(volumeId);

storageVolumeMapper.updateByPrimaryKeySelective(sv);

return new ResponseBean<>(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

} else {

return deleteVolumeStorage(volumeId);

}

}

} catch (Exception e) {

return new ResponseBean<>(StorageErrorCode.CODE\_010101.getCode(), StorageErrorCode.CODE\_010101.getMsg());

}

}

/\*\*

\* 更新共享存储空间

\*

\* @param volumeId

\* @param size

\* @return

\*/

public ResponseBean<Boolean> updateVolume(String userId, Integer volumeId, Integer size) {

ResponseBean<Boolean> data = new ResponseBean<>();

StorageVolume storageVolume = storageVolumeMapper.selectByPrimaryKey(volumeId);

ResponseBean<ResourceSchedulerBean> responseBean = schedulerRemoteService.resourceSchedulerNullParam();

if (StorageErrorCode.SUCCESS.getCode().equals(responseBean.getRetCode())) {

// 消费

ResponseBean<BigDecimal> responseBeanExpansionCostPurchase = getExpansionCostPurchase(volumeId, size / 1024);

logger.info("花费共计：【{}】", responseBeanExpansionCostPurchase.getData().doubleValue());

ResponseBean<Void> consumeResult = accountRemoteService.consume(

userId, Constants.ACCOUNT\_TYPE\_BUYSHARGESTORYGE, responseBeanExpansionCostPurchase.getData().toString(), String.format("类型： 共享存储， 名称： %s，磁盘大小：%s 【扩容共享存储】", volumeId, size));

logger.info("扣费结果【{}】", consumeResult);

if (!consumeResult.getRetCode().equals(StorageErrorCode.SUCCESS.getCode())) {

logger.info("扣费失败！");

return new ResponseBean<>(consumeResult.getRetCode(), consumeResult.getRetMsg());

}

Map<String, Object> baseParam = new HashMap<>();

baseParam.put("oldSize", storageVolume.getVolumeSize());

baseParam.put("costPurchase", responseBeanExpansionCostPurchase.getData().toString());

storageVolume.setVolumeSize(size);

baseParam.put("storageVolume", storageVolume);

ResourceSchedulerBean resourceSchedulerBean = responseBean.getData();

Map<String, Object> paramMap = new HashMap<>();

paramMap.put("user\_dir", storageVolume.getVolumeNameText());

paramMap.put("size", size);

paramMap.put("volume\_path", storageVolume.getVolumePath());

paramMap.put("volname", storageDataUtil.getVolname(storageVolume.getVolumePath()));

RabbitmqParam rp = new RabbitmqParam();

//自带的参数需要后端传回

rp.setBaseParam(baseParam);

//本次操作的类型

rp.setActiveType(Constants.ACTIVE\_TYPE\_QUOTA\_VOLUMES);

//调用后端的方法

rp.setMethodType(Constants.QUOTA\_VOLUMES);

//调用方法传的参数

rp.setMethodParam(paramMap);

rabbitTemplate.convertAndSend(Constants.TASK\_QUEUE + resourceSchedulerBean.getServiceId(), rp.toJson());

data.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

} else {

data.setCodeAndMsg(StorageErrorCode.CODE\_010204.getCode(), StorageErrorCode.CODE\_010204.getMsg());

}

return data;

}

/\*\*

\* 更新共享存储

\*

\* @return

\*/

public void updateVolumeCallback(JSONObject result) {

logger.info("收到的数据>>>>>>>>>>>>>>>" + result.toJSONString());

JSONObject r = result.getJSONObject("result");

JSONObject jsonBaseParam = result.getJSONObject("baseParam");

StorageVolume storageVolume = JSON.toJavaObject(jsonBaseParam.getJSONObject("storageVolume"), StorageVolume.class);

String costPurchase = jsonBaseParam.get("costPurchase").toString();

NoticeBean nb = new NoticeBean();

nb.setUserId(storageVolume.getUserId());

nb.setActiveType(Constants.ACTIVE\_TYPE\_QUOTA\_VOLUMES);

if (!StorageErrorCode.SUCCESS\_code.getCode().equals(r.getString("status"))) {

if (!"5315007".equals(r.getString("status"))) { // 5315007编号代表当前存储文件不存在

//扩容失败退款

ResponseBean<Void> rechargeResult = accountRemoteService.recharge(storageVolume.getUserId(), Constants.ACCOUNT\_TYPE\_REFUND, costPurchase.toString(), String.format("类型：用户: %s ,存储，磁盘大小：%s 【购买共享存储失败退款】", storageVolume.getUserId(), storageVolume.getVolumeSize()));

logger.info("退款结果：【{}】", rechargeResult.toString());

nb.setRetCode(rechargeResult.getRetCode());

nb.setRetMessage(rechargeResult.getRetMsg());

rabbitTemplate.convertAndSend(Constants.NOTICE\_WEB\_QUEUE, nb.toJson());

return;

}

}

storageVolumeMapper.updateByPrimaryKeySelective(storageVolume);

nb.setRetCode(StorageErrorCode.SUCCESS.getCode());

nb.setRetMessage(StorageErrorCode.SUCCESS.getMsg());

rabbitTemplate.convertAndSend(Constants.NOTICE\_WEB\_QUEUE, nb.toJson());

}

public List<StorageVolume> queryContainerVolume(Integer containerId) {

return myStorageVolumeMapper.queryContainerVolume(containerId);

}

/\*\*

\* 查询当前用户的共享卷不分页

\*

\* @param userId

\* @return

\*/

public ResponseBean<List<StorageVolume>> findVolumeStorageByUserAll(String userId) {

ResponseBean<List<StorageVolume>> data = new ResponseBean<>();

try {

StorageVolumeExample storageVolumeExample = new StorageVolumeExample();

List<Integer> i = new ArrayList<>();

i.add(Constants.UNUSED);

i.add(Constants.USED);

storageVolumeExample.createCriteria().andUserIdEqualTo(userId).andStatusIn(i);

List<StorageVolume> dataList = storageVolumeMapper.selectByExample(storageVolumeExample);

data.setData(dataList);

data.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

} catch (Exception e) {

data.setCodeAndMsg(StorageErrorCode.CODE\_010101.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

return data;

}

/\*\*

\* 查询当前容器下挂在的存储

\*

\* @param containerId

\* @return

\*/

public ResponseBean<List<StorageVolume>> finVolumeStorageByContainer(Integer containerId) {

ResponseBean<List<StorageVolume>> data = new ResponseBean<>();

/\*

\* 查询关联数据

\*/

CloudVolumeContainerContactExample cloudVolumeContainerContactExample = new CloudVolumeContainerContactExample();

cloudVolumeContainerContactExample.createCriteria().andContainerIdEqualTo(containerId);

List<CloudVolumeContainerContact> cloudVolumeContainerContactList = cloudVolumeContainerContactMapper.selectByExample(cloudVolumeContainerContactExample);

List<Integer> values = new ArrayList<>();

for (CloudVolumeContainerContact cloudVolumeContainerContact : cloudVolumeContainerContactList) {

values.add(cloudVolumeContainerContact.getVolumeId());

}

/\*\*

\*查询共享存储数据

\*/

if (values.size() > 0) {

StorageVolumeExample example = new StorageVolumeExample();

example.createCriteria().andIdIn(values);

List<StorageVolume> storageVolumeList = storageVolumeMapper.selectByExample(example);

data.setData(storageVolumeList);

}

data.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

return data;

}

/\*\*

\* @param volumeStorageId

\* @param name

\* @return 修改镜像名称

\*/

public ResponseBean<Boolean> updateVolumeStorageName(Integer volumeStorageId, String name, String userId) {

ResponseBean<Boolean> rb = new ResponseBean();

if (null != name && !name.equals("")) {

StorageVolumeExample storageVolumeExample = new StorageVolumeExample();

storageVolumeExample.createCriteria().andVolumeNameEqualTo(name).andUserIdEqualTo(userId);

List<StorageVolume> storageVolumeList = storageVolumeMapper.selectByExample(storageVolumeExample);

if (!storageVolumeList.isEmpty()) {

rb.setCodeAndMsg(StorageErrorCode.CODE\_010203.getCode(), StorageErrorCode.CODE\_010203.getMsg());

for (StorageVolume storageVolume : storageVolumeList) {

if (storageVolume.getId().equals(volumeStorageId)) {

rb.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

}

} else {

StorageVolume sv = new StorageVolume();

sv.setVolumeName(name);

sv.setId(volumeStorageId);

storageVolumeMapper.updateByPrimaryKeySelective(sv);

rb.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

return rb;

}

return new ResponseBean<>(StorageErrorCode.CODE\_010201.getCode(), StorageErrorCode.CODE\_010201.getMsg());

}

private ResponseBean deleteVolumeStorage(Integer storageVolumeId) {

ResponseBean<ResourceSchedulerBean> responseBean = schedulerRemoteService.resourceSchedulerNullParam();

if ("000000".equals(responseBean.getRetCode())) {

StorageVolume storageVolume = storageVolumeMapper.selectByPrimaryKey(storageVolumeId);

Map<String, Object> paramMap = new HashedMap();

paramMap.put("user\_dir", storageVolume.getVolumeNameText());

paramMap.put("volume\_path", storageVolume.getVolumePath());

paramMap.put("user\_id", storageVolume.getUserId());

RabbitmqParam rp = new RabbitmqParam();

rp.setBaseParam(storageVolume);

rp.setActiveType(Constants.ACTIVE\_TYPE\_DELETE\_VOLUMES\_TO\_TRASH);

//调用后端的方法

rp.setMethodType(Constants.DELETE\_VOLUMES\_TO\_TRASH);

//调用方法传的参数

rp.setMethodParam(paramMap);

logger.info("删除共享卷发送主机ID: 【{}】", responseBean.getData().getServiceId());

rabbitTemplate.convertAndSend(Constants.TASK\_QUEUE + responseBean.getData().getServiceId(), rp.toJson()); // TODO

return new ResponseBean(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

} else {

return new ResponseBean(responseBean.getRetCode(), responseBean.getRetMsg());

}

}

public ResponseBean deleteVolumeStorage(JSONObject result) {

logger.info("删除共享卷回调: 【{}】", result.toJSONString());

//解析返回值

JSONObject r = result.getJSONObject("result");

JSONObject baseParam = result.getJSONObject("baseParam");

StorageVolume storageVolume = JSONObject.toJavaObject(baseParam, StorageVolume.class);

//通知前端配置参数

NoticeBean nb = new NoticeBean();

nb.setUserId(storageVolume.getUserId());

nb.setActiveType(Constants.ACTIVE\_TYPE\_DELETE\_VOLUMES\_TO\_TRASH);

nb.setRetCode(StorageErrorCode.SUCCESS.getCode());

nb.setRetMessage(StorageErrorCode.SUCCESS.getMsg());

//验证底层是否执行成功

if ("success".equals(r.getString("status"))) {

rabbitTemplate.convertAndSend(Constants.NOTICE\_WEB\_QUEUE, nb.toJson());

} else {

JSONObject error = r.getJSONObject("error");

String error\_code = error.getString("error\_code");

//底层没有没有这个文件直接删除库数据即可

if ("5315007".equals(error\_code)) {

rabbitTemplate.convertAndSend(Constants.NOTICE\_WEB\_QUEUE, nb.toJson());

} else { // 删除失败

nb.setRetCode(StorageErrorCode.CODE\_010104.getCode());

nb.setRetMessage(StorageErrorCode.CODE\_010104.getMsg());

rabbitTemplate.convertAndSend(Constants.NOTICE\_WEB\_QUEUE, nb.toJson());

return new ResponseBean();

}

}

storageVolume.setStatus(Constants.REAL\_DELETE);//真实删除

storageVolumeMapper.updateByPrimaryKeySelective(storageVolume);

return new ResponseBean();

}

/\*\*

\* 修改默认挂载点

\*/

public ResponseBean updateMountpath(Integer volumeId, String newMountpath, String userId) {

StorageVolume storageVolume = new StorageVolume();

ResponseBean rb = new ResponseBean<>();

storageVolume.setId(volumeId);

storageVolume.setMountPath(newMountpath);

storageVolume.setUpdateTime(new Date());

storageVolumeMapper.updateByPrimaryKeySelective(storageVolume);

return new ResponseBean(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

/\*\*

\* 查询当前共享存储信息

\*/

public ResponseBean<StorageVolume> queryVolumeStaorageById(Integer id) {

ResponseBean<StorageVolume> rb = new ResponseBean<>();

StorageVolume storageVolume = storageVolumeMapper.selectByPrimaryKey(id);

rb.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

rb.setData(storageVolume);

return rb;

}

/\*\*

\* 查询当前共享卷下绑定的主机

\*/

public ResponseBean<PageInfo<Map<String, Object>>> getStorageVolumenBindContainer(Integer page, Integer pageNum, Integer storageVolumenId) {

ResponseBean<PageInfo<Map<String, Object>>> retBean = new ResponseBean<>();

PageHelper.startPage(page, pageNum, " b.update\_time DESC");

try {

List<Map<String, Object>> dataList = myStorageVolumeMapper.queryStorageVolumenBindContainer(storageVolumenId);

PageInfo<Map<String, Object>> pageInfo = new PageInfo<>(dataList);

retBean.setData(pageInfo);

retBean.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

} catch (Exception e) {

retBean.setCodeAndMsg(StorageErrorCode.CODE\_010101.getCode(), StorageErrorCode.SUCCESS.getMsg());

}

return retBean;

}

/\*\*

\* 创建存储收费业务

\*

\* @param storageName

\* @param size

\* @param userId

\* @param mountPath

\* @param months

\* @return

\*/

public ResponseBean<BigDecimal> createStorageVolumenAssetCost(String storageName, Integer size, String userId, String mountPath, Integer months) {

ResponseBean<BigDecimal> costPurchase = getCostPurchase(size, months);

logger.info("花费共计：【{}】", costPurchase.getData());

ResponseBean<Void> consumeResult = accountRemoteService.consume(userId, Constants.ACCOUNT\_TYPE\_BUYSHARGESTORYGE, costPurchase.getData().toString(), String.format("类型： 存储， 名称： %s，磁盘大小：%s 【购买共享存储】", storageName, size));

logger.info("扣费结果【{}】", consumeResult);

if (!consumeResult.getRetCode().equals(StorageErrorCode.SUCCESS.getCode())) {

logger.info("扣费失败！");

costPurchase.setCodeAndMsg(consumeResult.getRetCode(), consumeResult.getRetMsg());

return costPurchase;

}

ResponseBean<Integer> addVolumeStorageData = createVolumeStorage(userId, storageName, size \* 1024, mountPath, DateUtil.formatStringToDate(storageDateUtil.getEndChargingTime(months)).getTime());

if (!StorageErrorCode.SUCCESS.getCode().equals(addVolumeStorageData.getRetCode())) {

logger.info("创建失败！");

ResponseBean<Void> rechargeResult = accountRemoteService.recharge(userId, Constants.ACCOUNT\_TYPE\_REFUND, costPurchase.getData().toString(), String.format("类型：用户: %s ,存储， 名称： %s，磁盘大小：%s G【购买共享存储失败退款】", userId, storageName, size));

logger.info("退款结果：【{}】", rechargeResult.toString());

costPurchase.setCodeAndMsg(addVolumeStorageData.getRetCode(), addVolumeStorageData.getRetMsg());

return costPurchase;

}

// StorageVolume storageVolume = new StorageVolume();

// storageVolume.setId(addVolumeStorageData.getData().intValue());

// storageVolume.setEndTime(DateUtil.formatStringToDate(storageDateUtil.getEndChargingTime(months)));

// storageVolumeMapper.updateByPrimaryKeySelective(storageVolume);

return new ResponseBean<>(addVolumeStorageData.getRetCode(), addVolumeStorageData.getRetMsg());

}

/\*\*

\* 计算存储费用

\*

\* @param size 大小

\* @param months 购买时常 (几月)

\* @return

\*/

public ResponseBean<BigDecimal> getCostPurchase(Integer size, Integer months) {

logger.info("开始计费时间{}, 结束计费时间 {}", storageDateUtil.getStartChargingTime(), storageDateUtil.getEndChargingTime(months));

long m = storageDateUtil.getMinute(storageDateUtil.getStartChargingTime(), storageDateUtil.getEndChargingTime(months));

logger.info("一共{}分钟", m);

ResponseBean<BigDecimal> data = assetCostRemoteService.assetCostVolume(size, storageDateUtil.getMinute(storageDateUtil.getStartChargingTime(), storageDateUtil.getEndChargingTime(months)));

logger.info("共{}收费元", data.getData().doubleValue());

return data;

}

/\*\*

\* 查询续费价格

\*

\* @param vhdImageId

\* @param months

\* @return

\*/

public ResponseBean<BigDecimal> getStorageVolumenAssetRenewPrice(Integer vhdImageId, Integer months) {

ResponseBean<BigDecimal> responseBean = new ResponseBean<>();

StorageVolume volume = this.storageVolumeMapper.selectByPrimaryKey(vhdImageId);

Date startDate = volume.getEndTime();

Date endDate = DateUtil.addMonth(startDate, months);

Long time = (endDate.getTime() - startDate.getTime()) / (1000 \* 60);//分钟数

ResponseBean<BigDecimal> amount = assetCostRemoteService.assetCostVolume(volume.getVolumeSize() / 1024, time);

logger.info("续费 花费共计：【{}】", amount.getData());

responseBean.setData(amount.getData());

responseBean.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

return responseBean;

}

/\*\*

\* zhang.shuai

\* <p>

\* 计算扩容费用

\*

\* @param size 大小

\* @return

\*/

public ResponseBean<BigDecimal> getExpansionCostPurchase(Integer vhdImageId, Integer size) {

ResponseBean<BigDecimal> responseBean = new ResponseBean<>();

try {

StorageVolume storageVolume = this.storageVolumeMapper.selectByPrimaryKey(vhdImageId);

if (null == storageVolume) {

return new ResponseBean<>(StorageErrorCode.CODE\_010205.getCode(), StorageErrorCode.CODE\_010205.getMsg());

}

Date startDate = new Date();

Date endDate = storageVolume.getEndTime();

logger.info("开始计费时间{}, 结束计费时间 {}", startDate.toString(), endDate.toString());

Long time = (endDate.getTime() - startDate.getTime()) / (1000 \* 60);//分钟数

logger.info("一共{}分钟", time);

//得到当前剩余余额

BigDecimal balance = assetCostRemoteService.assetCostVolume((storageVolume.getVolumeSize() / 1024), time).getData();

//得到当前增加的费用2048

BigDecimal currentOst = assetCostRemoteService.assetCostVolume(size.intValue(), time).getData();

responseBean.setData(currentOst.subtract(balance));

responseBean.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

} catch (Exception e) {

return new ResponseBean<>(StorageErrorCode.CODE\_010205.getCode(), StorageErrorCode.CODE\_010205.getMsg());

}

return responseBean;

}

/\*\*

\* 查询过期共享存储

\*

\* @param date 相对时间

\* @return List<OutTimeStorage>

\*/

public ResponseBean<List<OutTimeVolume>> findOutTimeVolume(Date date, int type) {

List<OutTimeVolume> outTimeVolumes = null;

if (type == -1) {

outTimeVolumes = myStorageVolumeMapper.queryNoticeVolume(date);

} else if (type == 0) {

outTimeVolumes = myStorageVolumeMapper.queryOutTimeVolume(date);

} else if (type == 1) {

outTimeVolumes = myStorageVolumeMapper.queryDestroyVolume(date);

}

ResponseBean<List<OutTimeVolume>> responseBean = new ResponseBean<>();

responseBean.setData(outTimeVolumes);

responseBean.setCodeAndMsg(StorageErrorCode.SUCCESS.getCode(), StorageErrorCode.SUCCESS.getMsg());

return responseBean;

}

}

package com.yingchong.service.yccloud.storage.service;

import com.google.gson.Gson;

import com.yingchong.common.yccloud.commonapi.constants.SourceLimitConstants;

import com.yingchong.common.yccloud.commonapi.mapper.StorageVolumeMapper;

import com.yingchong.common.yccloud.commonapi.mapper.VhdImageMapper;

import com.yingchong.common.yccloud.commonapi.model.VhdImage;

import com.yingchong.common.yccloud.commonapi.util.RedisUtils;

import com.yingchong.common.yccloud.remote.bean.ResponseBean;

import com.yingchong.common.yccloud.remote.service.monitor.LimitRemoteService;

import com.yingchong.service.yccloud.storage.util.StorageErrorCode;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Pointcut;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.context.request.RequestAttributes;

import org.springframework.web.context.request.RequestContextHolder;

import org.springframework.web.context.request.ServletRequestAttributes;

import javax.servlet.http.HttpServletRequest;

@Aspect //定义一个切面

@Configuration

public class StorageAspect {

private static final Logger logger = LoggerFactory.getLogger(StorageAspect.class);

@Autowired

private LimitRemoteService limitRemoteService;

@Autowired

private RedisUtils redisUtils;

@Autowired

private VhdImageMapper vhdImageMapper;

@Autowired

private StorageVolumeMapper storageVolumeMapper;

// 定义切点Pointcut

@Pointcut("execution(\* com.yingchong.service.yccloud.storage.service.StorageService.\*(..))")

public void excudeService() {

}

@Around("excudeService()")

public Object doAround(ProceedingJoinPoint pjp) throws Throwable {

RequestAttributes ra = RequestContextHolder.getRequestAttributes();

ServletRequestAttributes sra = (ServletRequestAttributes) ra;

HttpServletRequest request = sra.getRequest();

String url = request.getRequestURL().toString();

String method = request.getMethod();

String uri = request.getRequestURI();

String queryString = request.getQueryString();

//Map<String, String[]> parameterMap = request.

String sourceLimitSwitch = redisUtils.get(SourceLimitConstants.sourceLimitSwitch);//阈值管理开关,如果是1,说明开启

if("1".equals(sourceLimitSwitch)){

Object[] args = pjp.getArgs();

String methodName = pjp.getSignature().getName();

logger.info("请求开始, 各个参数, url: {}, method: {}, uri: {}, params: {},methodName:{}", url, method, uri, queryString,methodName);

//创建块存储需要走资源阈值管理

if (methodName.equals("createStorageVhdAssetCost")) {//创建块存储

//CreateContainerParam createContainerParam = ((CreateContainerParam)args[0]);

int size = Integer.parseInt(String.valueOf(args[1])); // 块存储大小

String userId = String.valueOf(args[2]);

ResponseBean<Boolean> responseBean = limitRemoteService.checkCanCreateBlock(userId, 1,size);

if (!responseBean.getData()) {//可以创建

logger.error("阈值管理禁止创建,错误原因:{}",responseBean.getRetMsg());

ResponseBean<Double> createVhdImageResponse = new ResponseBean<>();

createVhdImageResponse.setCodeAndMsg(responseBean.getRetCode(),responseBean.getRetMsg());

return createVhdImageResponse;

}else {

Object result = pjp.proceed();

ResponseBean<Double> response = (ResponseBean<Double>) result;

if (response.getRetCode().equals(StorageErrorCode.SUCCESS.getCode())) {

limitRemoteService.updateCreateBlockLimit(userId, 1, size);

}

return result;

}

}

if (methodName.equals("delStorage")) {//删除块存储,恢复资源限制数量

int vhdImageId = Integer.parseInt(String.valueOf(args[0]));

Object result = pjp.proceed();

ResponseBean<Boolean> response = (ResponseBean<Boolean>) result;

if (response.getRetCode().equals(StorageErrorCode.SUCCESS.getCode())) {

VhdImage vhdImage = vhdImageMapper.selectByPrimaryKey(vhdImageId);

//删除成功,修改总量

limitRemoteService.updateCreateBlockLimit(

vhdImage.getUserId(),

-1,

-Integer.parseInt(vhdImage.getSize())/1024);

}

return result;

}

}

// result的值就是被拦截方法的返回值

Object result = pjp.proceed();

Gson gson = new Gson();

logger.info("请求结束，controller的返回值是 " + gson.toJson(result));

return result;

}

}

package com.yingchong.service.yccloud.storage.resource;

import com.alibaba.fastjson.JSONObject;

import com.yingchong.common.yccloud.remote.bean.ResponseBean;

import com.yingchong.common.yccloud.remote.bean.storage.OutTimeStorage;

import com.yingchong.service.yccloud.storage.service.StorageService;

import com.yingchong.util.ResponseUtil;

import com.wordnik.swagger.annotations.Api;

import com.wordnik.swagger.annotations.ApiOperation;

import com.wordnik.swagger.annotations.ApiParam;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.ws.rs.\*;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import java.util.Date;

import java.util.List;

/\*\*

\* =======================================================

\*

\* @Version ：0.0.1

\* @Description ：存储底层服务

\* ========================================================

\*/

@Path("/storage")

@Api(value = "/storage", description = "存储服务", produces = MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

@Component

public class StorageResource {

@Autowired

private StorageService storageService;

@POST

@Path("/storageManage")

@ApiOperation(value = "查询存储列表", httpMethod = "POST", notes = "存储管理")

public Response storageManage(

@ApiParam(name = "tel", value = "用户电话") @QueryParam("tel") String tel,

@ApiParam(name = "userId", value = "用户ID") @QueryParam("userId") String userId,

@ApiParam(name = "dateSta", value = "开始时间") @QueryParam("dateSta") String dateSta,

@ApiParam(name = "dateEnd", value = "结束时间") @QueryParam("dateEnd") String dateEnd,

@ApiParam(name = "status", value = "状态: 0未使用 1已使用 2已删除") @QueryParam("status") Integer status,

@ApiParam(name = "webName", value = "服务器名称或名称") @QueryParam("webName") String webName,

@ApiParam(required = true, name = "page", value = "分页-page") @QueryParam("page") Integer page,

@ApiParam(required = true, name = "pageNum", value = "分页-pageNum") @QueryParam("pageNum") Integer pageNum

) {

return ResponseUtil.getOkResponse(storageService.vhdImageManage(tel, userId, dateSta, dateEnd, status, webName, page, pageNum));

}

@POST

@Path("/deleteImage")

@ApiOperation(value = "删除镜像文件", httpMethod = "POST", notes = "存储管理")

public Response deleteImage(

@ApiParam(required = true, name = "vhdImageId", value = "vhdImageId 镜像di") @QueryParam("vhdImageId") Integer vhdImageId

) {

return ResponseUtil.getOkResponse(storageService.deleteImage(vhdImageId));

}

@POST

@Path("/deleteImageParameter")

@ApiOperation(value = "删除镜像文件回调", httpMethod = "POST", notes = "存储管理")

public Response deleteImageParameter(

@ApiParam(required = true, name = "param", value = "") @RequestBody JSONObject obj

) {

storageService.deleteImage(obj);

return ResponseUtil.getOkResponse("");

}

@GET

@Path("/findContainerStorage")

@ApiOperation(value = "查询容器的存储", httpMethod = "GET", notes = "查询容器的存储")

public

@ResponseBody

Response findContainerStorage(

@ApiParam(required = true, name = "containerId", value = "containerId") @QueryParam("containerId") Integer containerId

) {

return ResponseUtil.getOkResponse(storageService.findContainerStorage(containerId));

}

@POST

@Path("/updateStorage")

@ApiOperation(value = "修改存储名称", httpMethod = "POST", notes = "存储")

public Response updateStorage(

@ApiParam(required = true, name = "vhdImageId", value = "vhdImageId") @QueryParam("vhdImageId") Integer vhdImageId,

@ApiParam(required = true, name = "storageName", value = "storageName") @QueryParam("storageName") String storageName,

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId

) {

return ResponseUtil.getOkResponse(storageService.updateStorage(vhdImageId, storageName, userId));

}

@POST

@Path("/createStorageVhd")

@ApiOperation(value = "只创建镜像文件", httpMethod = "POST", notes = "存储")

public Response createStorageVhd(

@ApiParam(required = true, name = "storageName", value = "storageName") @QueryParam("storageName") String storageName,

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size,

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

@ApiParam(name = "mountPath", value = "mountPath") @QueryParam("mountPath") String mountPath,

@ApiParam(name = "endTime", value = "endTime") @QueryParam("endTime") Long endTime

) {

return ResponseUtil.getOkResponse(storageService.createStorageVhd(storageName, size, userId, mountPath,endTime));

}

@POST

@Path("/delStorage")

@ApiOperation(value = "删除存储", httpMethod = "POST", notes = "存储")

public Response delStorage(

@ApiParam(required = true, name = "VhdImageId", value = "镜像文件表id") @QueryParam("VhdImageId") Integer VhdImageId

) {

return ResponseUtil.getOkResponse(storageService.delStorage(VhdImageId));

}

@POST

@Path("/updateStorageList")

@ApiOperation(value = "更新存储状态", httpMethod = "POST", notes = "存储")

public Response updateStorageList(

@ApiParam(required = true, name = "vhdImageId", value = "镜像文件表id") @QueryParam("vhdImageId") Integer vhdImageId,

@ApiParam(required = true, name = "type", value = "镜像文件表id") @QueryParam("type") Integer type

) {

return ResponseUtil.getOkResponse(storageService.updateStorageList(vhdImageId, type));

}

@POST

@Path("/formatImage")

@ApiOperation(value = "格式化镜像文件", httpMethod = "POST", notes = "存储")

public Response formatImage(

@ApiParam(required = true, name = "containerId", value = "containerId") @QueryParam("containerId") Integer containerId,

@ApiParam(required = true, name = "vhdImageId", value = "vhdImageId") @QueryParam("vhdImageId") Integer vhdImageId,

@ApiParam(required = true, name = "type", value = "type") @QueryParam("type") Integer type

) {

return ResponseUtil.getOkResponse(storageService.formatBlock(containerId, vhdImageId, type));

}

@POST

@Path("/formatImageParameter")

@ApiOperation(value = "格式化镜像文件回调", httpMethod = "POST", notes = "存储")

public Response formatImageParameter(

@ApiParam(required = true, name = "param", value = "") @RequestBody JSONObject obj

) {

storageService.formatBlock(obj);

return ResponseUtil.getOkResponse("ok");

}

@POST

@Path("/queryNoboundStorageContainer")

@ApiOperation(value = "查询没挂在存储的主机", httpMethod = "POST", notes = "存储")

public Response queryNoboundStorageContainer(

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId

) {

return ResponseUtil.getOkResponse(storageService.queryNoboundStorageContainer(userId));

}

@POST

@Path("/updateMountpath")

@ApiOperation(value = "修改存储挂载点", httpMethod = "POST", notes = "存储")

public Response updateMountpath(

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

@ApiParam(required = true, name = "storageId", value = "storageId") @QueryParam("storageId") Integer storageId,

@ApiParam(required = true, name = "newMountpath", value = "newMountpath") @QueryParam("newMountpath") String newMountpath

) {

return ResponseUtil.getOkResponse(storageService.updateMountpath(storageId, newMountpath, userId));

}

@POST

@Path("/createStorageVhdAssetCost")

@ApiOperation(value = "创建存储并支付接口", httpMethod = "POST", notes = "存储")

public Response createStorageVhdAssetCost(

@ApiParam(required = true, name = "storageName", value = "storageName") @QueryParam("storageName") String storageName,

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size,

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

@ApiParam(name = "mountPath", value = "挂载点") @QueryParam("mountPath") String mountPath,

@ApiParam(required = true, name = "months", value = "购买时间(月)") @QueryParam("months") Integer months

) {

return ResponseUtil.getOkResponse(storageService.createStorageVhdAssetCost(storageName, size, userId, mountPath, months));

}

@POST

@Path("/getCostPurchase")

@ApiOperation(value = "购买存储费用计算接口", httpMethod = "POST", notes = "存储")

public Response getCostPurchase(

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size,

@ApiParam(required = true, name = "months", value = "购买时间(月)") @QueryParam("months") Integer months

) {

return ResponseUtil.getOkResponse(storageService.getCostPurchase(size, months));

}

@POST

@Path("/getRenewalStorageCostPurchase")

@ApiOperation(value = "续费价格计算接口", httpMethod = "POST", notes = "存储")

public Response getRenewalStorageCostPurchase(

@ApiParam(required = true, name = "vhdImageId", value = "vhdImageId") @QueryParam("vhdImageId") Integer vhdImageId,

@ApiParam(required = true, name = "months", value = "months") @QueryParam("months") Integer months

) {

return ResponseUtil.getOkResponse(storageService.getRenewalStorageCostPurchase(vhdImageId, months));

}

@GET

@Path("/findOutTimeStorage")

@ApiOperation(value = "查询过期块存储", httpMethod = "GET", notes = "查询过期块存储")

public ResponseBean<List<OutTimeStorage>> findOutTimeStorage(

@ApiParam(required = true, name = "date", value = "date") @QueryParam("date") Long date,

@ApiParam(required = true, name = "type", value = "type") @QueryParam("type") Integer type

) {

return storageService.findOutTimeStorage(new Date(date),type);

}

}

package com.yingchong.service.yccloud.storage.resource;

import com.alibaba.fastjson.JSONObject;

import com.yingchong.common.yccloud.commonapi.model.StorageVolume;

import com.yingchong.common.yccloud.remote.bean.ResponseBean;

import com.yingchong.common.yccloud.remote.bean.storage.OutTimeVolume;

import com.yingchong.service.yccloud.storage.service.VolumeStorageService;

import com.yingchong.util.ResponseUtil;

import com.wordnik.swagger.annotations.Api;

import com.wordnik.swagger.annotations.ApiOperation;

import com.wordnik.swagger.annotations.ApiParam;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;

import javax.ws.rs.\*;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import java.util.Date;

import java.util.List;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description :

\* ========================================================

\*/

@Path("/storage/volume")

@Api(value = "/storage/volume", description = "共享存储服务", produces = MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

@Component

public class VolumeStorageResource {

@Autowired

VolumeStorageService service;

@POST

@Path("/isHasContact")

@ApiOperation(value = "查询共享存储是否被绑定", httpMethod = "POST", notes = "共享存储管理")

public Response isHasContact(

@Context HttpServletRequest req,

@ApiParam(required = true, name = "volumeId", value = "volume\_Id") @QueryParam("volumeId") Integer volumeId

) {

return ResponseUtil.getOkResponse(service.isHasContact(volumeId));

}

@POST

@Path("/findVolumeStorageByUser")

@ApiOperation(value = "分页查询共享存储", httpMethod = "POST", notes = "共享存储管理")

public Response findVolumeStorageByUser(

@ApiParam(name = "tel", value = "用户电话") @QueryParam("tel") String tel,

@ApiParam(name = "userId", value = "user\_ID") @QueryParam("userId") String userId,

@ApiParam(required = true, name = "page", value = "page") @QueryParam("page") Integer page,

@ApiParam(required = true, name = "pageNum", value = "pageNum") @QueryParam("pageNum") Integer pageNum,

@ApiParam(name = "dateSta", value = "开始时间") @QueryParam("dateSta") String dateSta,

@ApiParam(name = "dateEnd", value = "结束时间") @QueryParam("dateEnd") String dateEnd,

@ApiParam(name = "status", value = "状态0:未使用，1:已使用，２:已删除") @QueryParam("status") Integer status,

@ApiParam(name = "keyword", value = "关键字") @QueryParam("keyword") String keyword

) {

return ResponseUtil.getOkResponse(service.findVolumeStorageByUser(tel, userId, page, pageNum, dateSta, dateEnd, status, keyword));

}

@POST

@Path("/createVolumeStorage")

@ApiOperation(value = "创建共享存储", httpMethod = "POST", notes = "共享存储管理")

public Response createVolumeStorage(

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

@ApiParam(required = true, name = "volName", value = "volName") @QueryParam("volName") String volName,

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size,

@ApiParam(required = true, name = "mountPath", value = "mountPath") @QueryParam("mountPath") String mountPath,

@ApiParam(required = true, name = "endTime", value = "endTime") @QueryParam("endTime") Long endTime

) {

return ResponseUtil.getOkResponse(service.createVolumeStorage(userId, volName, size, mountPath,endTime));

}

@POST

@Path("/deleteVolume")

@ApiOperation(value = "删除共享存储", httpMethod = "POST", notes = "共享存储管理")

public Response deleteVolume(

@ApiParam(required = true, name = "volumeId", value = "volumeId") @QueryParam("volumeId") Integer volumeId,

@ApiParam(required = true, name = "PhysDeletion", value = "1：数据库删除（预删除） , 0:为真实是删除（彻底删除）") @QueryParam("PhysDeletion") int PhysDeletion

) {

Boolean b = (PhysDeletion != 0);

return ResponseUtil.getOkResponse(service.deleteVolume(volumeId, b));

}

@POST

@Path("/updateVolume")

@ApiOperation(value = "修改存储大小", httpMethod = "POST", notes = "共享存储管理")

public Response updateVolume(

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

@ApiParam(required = true, name = "volumeId", value = "volumeId") @QueryParam("volumeId") Integer volumeId,

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size

) {

return ResponseUtil.getOkResponse(service.updateVolume(userId,volumeId, size));

}

@POST

@Path("/updateVolumeParameter")

@ApiOperation(value = "修改存储大小回调", httpMethod = "POST", notes = "共享存储管理")

public Response updateVolumeParameter(

@ApiParam(required = true, name = "param", value = "") @RequestBody JSONObject obj

) {

service.updateVolumeCallback(obj);

return ResponseUtil.getOkResponse("");

}

@POST

@Path("/updateVolumeStatus")

@ApiOperation(value = "更新共享存储状态 type:0 恢复 10 彻底删除", httpMethod = "POST", notes = "共享存储管理")

public Response updateVolumeStatus(

@ApiParam(required = true, name = "volumeId", value = "volumeId") @QueryParam("volumeId") Integer volumeId,

@ApiParam(required = true, name = "type", value = "type") @QueryParam("type") Integer type

) {

return ResponseUtil.getOkResponse(service.updateVolumeStatus(volumeId, type));

}

@GET

@Path("/queryContainerVolume")

@ApiOperation(value = "查询这个容器的共享存储信息", httpMethod = "GET", notes = "查询这个容器的共享存储信息")

public

@ResponseBody

List<StorageVolume> queryContainerVolume(

@ApiParam(required = true, name = "containerId", value = "containerId") @QueryParam("containerId") Integer containerId

) {

return service.queryContainerVolume(containerId);

}

@POST

@Path("/findVolumeStorageByUserAll")

@ApiOperation(value = "查询当前用户共享卷不分页", httpMethod = "POST", notes = "共享存储管理")

public Response findVolumeStorageByUserAll(

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId

) {

return ResponseUtil.getOkResponse(service.findVolumeStorageByUserAll(userId));

}

@POST

@Path("/finVolumeStorageByContainer")

@ApiOperation(value = "查询当主机下挂在的共享存储", httpMethod = "POST", notes = "共享存储管理")

public Response finVolumeStorageByContainer(

@ApiParam(required = true, name = "containerId", value = "containerId") @QueryParam("containerId") Integer containerId

) {

return ResponseUtil.getOkResponse(service.finVolumeStorageByContainer(containerId));

}

@POST

@Path("/updateVolumeStorageName")

@ApiOperation(value = "修改存储名称", httpMethod = "POST", notes = "共享存储管理")

public Response updateVolumeStorageName(

@ApiParam(required = true, name = "volumeStorageId", value = "volumeStorageId") @QueryParam("volumeStorageId") Integer volumeStorageId,

@ApiParam(required = true, name = "name", value = "name") @QueryParam("name") String name,

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId

) {

return ResponseUtil.getOkResponse(service.updateVolumeStorageName(volumeStorageId, name, userId));

}

@POST

@Path("/deleteVolumeStorage")

@ApiOperation(value = "删除存储回调", httpMethod = "POST", notes = "共享存储管理")

public Response deleteVolumeStorage(

@ApiParam(required = true, name = "param", value = "") @RequestBody JSONObject obj

) {

return ResponseUtil.getOkResponse(service.deleteVolumeStorage(obj));

}

@POST

@Path("/updateMountpath")

@ApiOperation(value = "修改共享卷挂载点", httpMethod = "POST", notes = "存储")

public Response updateMountpath(

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

@ApiParam(required = true, name = "volumeId", value = "volumeId") @QueryParam("volumeId") Integer volumeId,

@ApiParam(required = true, name = "newMountpath", value = "newMountpath") @QueryParam("newMountpath") String newMountpath

) {

return ResponseUtil.getOkResponse(service.updateMountpath(volumeId, newMountpath, userId));

}

@POST

@Path("/queryVolumeStaorageById")

@ApiOperation(value = "查询当前共享卷信息", httpMethod = "POST", notes = "存储")

public Response queryVolumeStaorageById(

@ApiParam(required = true, name = "volumeId", value = "volumeId") @QueryParam("volumeId") Integer volumeId

) {

return ResponseUtil.getOkResponse(service.queryVolumeStaorageById(volumeId));

}

@POST

@Path("/getStorageVolumenBindContainer")

@ApiOperation(value = "分页查询当前共享存储绑定的主机信息", httpMethod = "POST", notes = "共享存储管理")

public Response getStorageVolumenBindContainer(

@ApiParam(required = true, name = "page", value = "page") @QueryParam("page") Integer page,

@ApiParam(required = true, name = "pageNum", value = "pageNum") @QueryParam("pageNum") Integer pageNum,

@ApiParam(name = "storageVolumenId", value = "共享卷ID") @QueryParam("storageVolumenId") Integer storageVolumenId

) {

return ResponseUtil.getOkResponse(service.getStorageVolumenBindContainer(page, pageNum, storageVolumenId));

}

@POST

@Path("/createStorageVolumenAssetCost")

@ApiOperation(value = "创建存储收费业务", httpMethod = "POST", notes = "共享存储管理")

public Response createStorageVolumenAssetCost(

@ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

@ApiParam(required = true, name = "volName", value = "volName") @QueryParam("volName") String volName,

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size,

@ApiParam(name = "mountPath", value = "mountPath") @QueryParam("mountPath") String mountPath,

@ApiParam(required = true, name = "months", value = "months") @QueryParam("months") Integer months

) {

return ResponseUtil.getOkResponse(service.createStorageVolumenAssetCost(volName, size, userId, mountPath, months));

}

@POST

@Path("/getCostPurchase")

@ApiOperation(value = "计算存储费用", httpMethod = "POST", notes = "共享存储管理")

public Response getCostPurchase(

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size,

@ApiParam(required = true, name = "months", value = "months") @QueryParam("months") Integer months

) {

return ResponseUtil.getOkResponse(service.getCostPurchase(size, months));

}

@POST

@Path("/getStorageVolumenAssetRenewPrice")

@ApiOperation(value = "计算共享存储续费价格", httpMethod = "POST", notes = "共享存储管理")

public Response getStorageVolumenAssetRenewPrice(

@ApiParam(required = true, name = "vhdImageId", value = "vhdImageId") @QueryParam("vhdImageId") Integer vhdImageId,

@ApiParam(required = true, name = "months", value = "months") @QueryParam("months") Integer months

) {

return ResponseUtil.getOkResponse(service.getStorageVolumenAssetRenewPrice(vhdImageId, months));

}

@POST

@Path("/getExpansionCostPurchase")

@ApiOperation(value = "计算存储扩容价格", httpMethod = "POST", notes = "共享存储管理")

public Response getExpansionCostPurchase(

@ApiParam(required = true, name = "id", value = "id") @QueryParam("id") Integer id,

@ApiParam(required = true, name = "size", value = "size") @QueryParam("size") Integer size

) {

return ResponseUtil.getOkResponse(service.getExpansionCostPurchase(id, size));

}

@GET

@Path("/findOutTimeVolume")

@ApiOperation(value = "查询过期共享存储", httpMethod = "GET", notes = "查询过期共享存储")

public ResponseBean<List<OutTimeVolume>> findOutTimeVolume(

@ApiParam(required = true, name = "date", value = "date") @QueryParam("date") Long date,

@ApiParam(required = true, name = "type", value = "type") @QueryParam("type") Integer type

) {

return service.findOutTimeVolume(new Date(date),type);

}

}

package com.yingchong.service.yccloud.storage.mapper;

import com.yingchong.common.yccloud.commonapi.model.StorageVolume;

import com.yingchong.common.yccloud.remote.bean.storage.OutTimeVolume;

import org.apache.ibatis.annotations.Result;

import org.apache.ibatis.annotations.Results;

import org.apache.ibatis.annotations.SelectProvider;

import org.apache.ibatis.type.JdbcType;

import java.util.Date;

import java.util.List;

import java.util.Map;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description :

\* ========================================================

\*/

public interface MyStorageVolumeMapper {

@SelectProvider(type = MyStorageVolumeProvider.class, method = "queryVolumeStorageList")

List<Map<String, Object>> findStorageVolume(Map<String, Object> param);

@SelectProvider(type = MyStorageVolumeProvider.class, method = "queryContainerVolume")

@Results({

@Result(column = "id", property = "id", jdbcType = JdbcType.INTEGER),

@Result(column = "userId", property = "userId", jdbcType = JdbcType.VARCHAR),

@Result(column = "volumeName", property = "volumeName", jdbcType = JdbcType.VARCHAR),

@Result(column = "volumeNameText", property = "volumeNameText", jdbcType = JdbcType.VARCHAR),

@Result(column = "volumePath", property = "volumePath", jdbcType = JdbcType.VARCHAR),

@Result(column = "mountPath", property = "mountPath", jdbcType = JdbcType.VARCHAR),

@Result(column = "volumeId", property = "volumeId", jdbcType = JdbcType.INTEGER),

@Result(column = "volumeSize", property = "volumeSize", jdbcType = JdbcType.INTEGER),

@Result(column = "status", property = "status", jdbcType = JdbcType.INTEGER),

@Result(column = "containerId", property = "containerId", jdbcType = JdbcType.INTEGER),

@Result(column = "createTime", property = "createTime", jdbcType = JdbcType.DATE),

@Result(column = "updateTime", property = "updateTime", jdbcType = JdbcType.DATE)

})

List<StorageVolume> queryContainerVolume(Integer containerId);

@SelectProvider(type = MyStorageVolumeProvider.class, method = "queryStorageVolumenBindContainer")

List<Map<String, Object>> queryStorageVolumenBindContainer(Integer id);

//过期且未删除

@SelectProvider(type=MyStorageVolumeProvider.class, method="queryNoticeVolume")

List<OutTimeVolume> queryNoticeVolume(Date date);

//过期且绑定

@SelectProvider(type=MyStorageVolumeProvider.class, method="queryOutTimeVolume")

List<OutTimeVolume> queryOutTimeVolume(Date date);

//过期未绑定

@SelectProvider(type=MyStorageVolumeProvider.class, method="queryDestroyVolume")

List<OutTimeVolume> queryDestroyVolume(Date date);

}

package com.yingchong.service.yccloud.storage.mapper;

import java.util.Date;

import java.util.Map;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description :

\* ========================================================

\*/

public class MyStorageVolumeProvider {

public String queryVolumeStorageList(Map<String, Object> param) {

StringBuilder sql = new StringBuilder("\tSELECT \n" +

" aa.id, aa.volume\_name as volumeName ,bb.webName ,aa.status,aa.volume\_size as volumeSize, aa.create\_time as createTime, ui.mobile ,aa.mount\_path default\_mount\_path ,aa.end\_time as endTime , datediff(aa.end\_time, now()) as end\_time\_day " +

"FROM\n" +

" storage\_volume aa\n" +

" LEFT JOIN\n" +

" (SELECT \n" +

" b.id,\n" +

"\t\t GROUP\_CONCAT(concat(c.webname,':', a.mount\_path)) as webName \n" +

" FROM\n" +

" cloud\_volume\_container\_contact a\n" +

" LEFT JOIN storage\_volume b ON a.volume\_id = b.id\n" +

" LEFT JOIN cloud\_container\_assets c ON a.container\_id = c.id\n" +

" GROUP BY b.id) bb ON aa.id = bb.id \n" +

" LEFT JOIN user\_info ui ON aa.user\_id = ui.user\_id \n" +

"WHERE aa.status!=10");

if (param.get("tel") != null){

sql.append(" and ui.mobile like concat('%',#{tel},'%') ");

}

if (param.get("userId") != null) {

sql.append(" and aa.user\_id=#{userId}");

}

if (param.get("dateSta") != null) {

sql.append(" and DATE\_FORMAT(aa.create\_time, '%Y-%m-%d %H:%I:%S') >= DATE\_FORMAT(#{dateSta}, '%Y-%m-%d 00:00:00') ");

}

if (param.get("dateEnd") != null) {

sql.append(" and DATE\_FORMAT(aa.create\_time, '%Y-%m-%d %H:%I:%S') <= DATE\_FORMAT(#{dateEnd}, '%Y-%m-%d 23:59:59') ");

}

if (param.get("status") != null) {

sql.append(" and aa.status = #{status} ");

}else {

sql.append("and aa.status != 2 ");

}

if (param.get("keyword") != null) {

sql.append(" and ( aa.volume\_name like concat('%',#{keyword},'%') or bb.webName like concat('%',#{keyword},'%') or aa.id like concat('%',#{keyword},'%') )");

}

return sql.toString();

}

public String queryContainerVolume(Integer containerId) {

String sql = "SELECT sv.id,sv.user\_id userId,sv.volume\_name volumeName,sv.volume\_name\_text volumeNameText, sv.volume\_id volumeId, sv.volume\_size volumeSize,sv.volume\_path volumePath,sv.container\_id containerId,sv.`status`,sv.create\_time createTime, sv.update\_time updateTime, cvcc.mount\_path mountPath from storage\_volume sv " +

"left join cloud\_volume\_container\_contact cvcc on cvcc.volume\_id = sv.id " +

"where cvcc.container\_id = #{containerId}";

return sql;

}

public String queryStorageVolumenBindContainer (Integer id){

String sql = "SELECT c.id , c.webname, a.mount\_path FROM cloud\_volume\_container\_contact a LEFT JOIN storage\_volume b ON a.volume\_id = b.id LEFT JOIN cloud\_container\_assets c ON a.container\_id = c.id where b.status!=10 and b.id =#{id} ";

return sql;

}

public String queryNoticeVolume(Date date) {

return "select ui.email email, ui.mobile mobile, ui.login\_name loginName, sv.id id,sv.volume\_size volumeSize, sv.`status` status, sv.user\_id userId, sv.volume\_name volumeName,\n" +

"sv.volume\_path path, sv.create\_time createTime, sv.update\_time updateTime, sv.end\_time endTime,cca.id containerId, cca.webname webName\n" +

"from storage\_volume sv \n" +

"left join user\_info ui on sv.user\_id = ui.user\_id\n" +

"left join cloud\_volume\_container\_contact cvcc on cvcc.volume\_id = sv.id\n" +

"left join cloud\_container\_assets cca on cca.id = cvcc.container\_id\n" +

"where #{date} > sv.end\_time and sv.status != 10";

}

public String queryOutTimeVolume(Date date) {

return "select ui.email email, ui.mobile mobile, ui.login\_name loginName, sv.id id,sv.volume\_size volumeSize, sv.`status` status, sv.user\_id userId, sv.volume\_name volumeName,\n" +

"sv.volume\_path path, sv.create\_time createTime, sv.update\_time updateTime, sv.end\_time endTime,cca.id containerId, cca.webname webName\n" +

"from storage\_volume sv \n" +

"left join user\_info ui on sv.user\_id = ui.user\_id\n" +

"left join cloud\_volume\_container\_contact cvcc on cvcc.volume\_id = sv.id\n" +

"left join cloud\_container\_assets cca on cca.id = cvcc.container\_id\n" +

"where #{date} > sv.end\_time and sv.status = 1";

}

public String queryDestroyVolume(Date date) {

return "select ui.email email, ui.mobile mobile, ui.login\_name loginName, sv.id id,sv.volume\_size volumeSize, sv.`status` status, sv.user\_id userId, sv.volume\_name volumeName,\n" +

"sv.volume\_path path, sv.create\_time createTime, sv.update\_time updateTime, sv.end\_time endTime,cca.id containerId, cca.webname webName\n" +

"from storage\_volume sv \n" +

"left join user\_info ui on sv.user\_id = ui.user\_id\n" +

"left join cloud\_volume\_container\_contact cvcc on cvcc.volume\_id = sv.id\n" +

"left join cloud\_container\_assets cca on cca.id = cvcc.container\_id\n" +

"where #{date} > sv.end\_time and sv.status = 0";

}

}

package com.yingchong.service.yccloud.storage.mapper;

import com.yingchong.common.yccloud.remote.bean.storage.ContainerStorageBean;

import com.yingchong.common.yccloud.remote.bean.storage.OutTimeStorage;

import org.apache.ibatis.annotations.Result;

import org.apache.ibatis.annotations.Results;

import org.apache.ibatis.annotations.SelectProvider;

import org.apache.ibatis.type.JdbcType;

import java.util.Date;

import java.util.List;

import java.util.Map;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description :

\* ========================================================

\*/

public interface StorageServiceMapper {

@SelectProvider(type=StorageServiceProvider.class, method="queryStorageList")

List<Map<String, Object>> queryStorageList(Map<String, Object> param);

@SelectProvider(type = StorageServiceProvider.class, method = "containerOfStorageSQL")

List<Map<String, Object>> queryContainerOfStorage(Integer containerId);

@SelectProvider(type = StorageServiceProvider.class, method = "queryStorageInfoSQL")

@Results({

@Result(column = "imageName", property = "imageName", jdbcType = JdbcType.VARCHAR),

@Result(column = "name", property = "name", jdbcType = JdbcType.VARCHAR),

@Result(column = "devName", property = "devName", jdbcType = JdbcType.VARCHAR),

@Result(column = "size", property = "size", jdbcType = JdbcType.INTEGER),

@Result(column = "vhdImageId", property = "vhdImageId", jdbcType = JdbcType.INTEGER),

@Result(column = "path", property = "path", jdbcType = JdbcType.VARCHAR)

})

List<ContainerStorageBean> queryStorageInfo(String vhdImageId);

@SelectProvider(type=StorageServiceProvider.class, method="queryNoboundStorageContainer")

List<Map<String, Object>> queryNoboundStorageContainer(Map<String, Object> param);

@SelectProvider(type=StorageServiceProvider.class, method="queryNoticeStorage")

List<OutTimeStorage> queryNoticeStorage(Date date);

//查找过期的,则是已经绑定的

@SelectProvider(type=StorageServiceProvider.class, method="queryOutTimeStorage")

List<OutTimeStorage> queryOutTimeStorage(Date date);

//需要回收的,过期的且未绑定的

@SelectProvider(type=StorageServiceProvider.class, method="queryDestoryStorage")

List<OutTimeStorage> queryDestoryStorage(Date date);

}

package com.yingchong.service.yccloud.storage.mapper;

import java.util.Date;

import java.util.Map;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description :

\* ========================================================

\*/

public class StorageServiceProvider {

public String queryStorageList(Map<String, Object> param) {

StringBuilder sql = new StringBuilder("SELECT \n" +

" aa.id , c.id containerId , aa.name, c.webname as webName,aa.status,aa.size,aa.mount\_path path,aa.create\_time,ui.mobile,aa.end\_time as endTime ,datediff(aa.end\_time, now()) as end\_time\_day" +

" FROM\n" +

" vhd\_image aa\n" +

" LEFT JOIN\n" +

" (select vhd\_image\_id , server\_assets\_id from tap\_block where `status` !=2) a ON a.vhd\_image\_id = aa.id \n" +

" LEFT JOIN\n" +

" cloud\_container\_assets c ON a.server\_assets\_id = c.id\n" +

" LEFT JOIN \n" +

" user\_info ui ON aa.user\_id = ui.user\_id\n"+

" WHERE aa.status!=10 ");

if (param.get("tel") != null){

sql.append(" and ui.mobile like concat('%',#{tel},'%') ");

}

if (param.get("userId") != null){

sql.append(" and aa.user\_id=#{userId} ");

}

if (param.get("dateSta") != null) {

sql.append(" and DATE\_FORMAT(aa.create\_time, '%Y-%m-%d %H:%I:%S') >= DATE\_FORMAT(#{dateSta}, '%Y-%m-%d 00:00:00') ");

}

if (param.get("dateEnd") != null) {

sql.append(" and DATE\_FORMAT(aa.create\_time, '%Y-%m-%d %H:%I:%S') <= DATE\_FORMAT(#{dateEnd}, '%Y-%m-%d 23:59:59') ");

}

if (param.get("status") != null) {

sql.append(" and aa.status = #{status} ");

} else {

sql.append(" and aa.status!=2 ");

}

if (param.get("webName") != null) {

sql.append(" and ( aa.name like concat('%',#{webName},'%') or c.webname like concat('%',#{webName},'%') or aa.id like concat('%',#{webName},'%') )");

}

return sql.toString();

}

public String containerOfStorageSQL() {

return "select aa.id , aa.`name` ,aa.size from vhd\_image aa LEFT JOIN tap\_block bb ON bb.vhd\_image\_id = aa.id WHERE aa.`status` ='1' and bb.`status` !=2 and bb.server\_assets\_id=#{containerId}";

}

public String queryStorageInfoSQL() {

//status!=2 查询没有删除的

String sql = "select vi.image\_name imageName, vi.`name` name,vi.size size, vi.id vhdImageId,vi.path path, tb.dev\_name devName from vhd\_image vi left join tap\_block tb on vi.id=tb.vhd\_image\_id \n" +

"where vi.id = #{vhdImageId} and status !=2 ";

return sql;

}

public String queryNoboundStorageContainer(Map<String, Object> param) {

StringBuilder sql = new StringBuilder("SELECT\n" +

"\t\*\n" +

"FROM\n" +

"\tcloud\_container\_assets\n" +

"WHERE\n" +

"\tid NOT IN (\n" +

"\t\tSELECT\n" +

"\t\t\tserver\_assets\_id\n" +

"\t\tFROM\n" +

"\t\t\ttap\_block\n" +

"\t)\n" +

"AND `status` = 1 user\_id = #{user\_id}");

return sql.toString();

}

public String queryNoticeStorage(Date date) {

return "select vi.id id, vi.`name` name, vi.user\_id userId, vi.path path, vi.`status` status, vi.create\_time createTime,vi.update\_time upateTime, vi.end\_time endTime,vi.size size,\n" +

"ui.email email, ui.mobile mobile, ui.login\_name loginName ,cca.id containerId, cca.webname webName \n" +

"from vhd\_image vi left join user\_info ui on vi.user\_id = ui.user\_id \n " +

"left join tap\_block tb on tb.vhd\_image\_id = vi.id \n" +

"left join cloud\_container\_assets cca on cca.id = tb.server\_assets\_id \n"+

"where #{date} > vi.end\_time and vi.`status` != 10 "; //10是彻底删除

}

public String queryOutTimeStorage(Date date) {

return "select vi.id id, vi.`name` name, vi.user\_id userId, vi.path path, vi.`status` status, vi.create\_time createTime,vi.update\_time upateTime, vi.end\_time endTime,vi.size size,\n" +

"ui.email email, ui.mobile mobile, ui.login\_name loginName ,cca.id containerId, cca.webname webName \n" +

"from vhd\_image vi left join user\_info ui on vi.user\_id = ui.user\_id \n " +

"left join tap\_block tb on tb.vhd\_image\_id = vi.id \n" +

"left join cloud\_container\_assets cca on cca.id = tb.server\_assets\_id \n"+

"where #{date} > vi.end\_time and tb.`status` = 1 and vi.status = 1" ; //1是绑定状态

}

public String queryDestoryStorage(Date date) {

return "select vi.id id, vi.`name` name, vi.user\_id userId, vi.path path, vi.`status` status, vi.create\_time createTime,vi.update\_time upateTime, vi.end\_time endTime,vi.size size,\n" +

"ui.email email, ui.mobile mobile, ui.login\_name loginName ,cca.id containerId, cca.webname webName \n" +

"from vhd\_image vi left join user\_info ui on vi.user\_id = ui.user\_id \n " +

"left join tap\_block tb on tb.vhd\_image\_id = vi.id \n" +

"left join cloud\_container\_assets cca on cca.id = tb.server\_assets\_id \n"+

"where #{date} > vi.end\_time and vi.`status` = 0 "; //0是未使用状态

}

}

package com.yingchong.service.yccloud.storage.util;

import com.yingchong.service.yccloud.storage.service.StorageService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Service;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.List;

/\*\*

\* =======================================================

\*

\* @Version ：0.0.1

\* @Description ：

\* ========================================================

\*/

@Service

public class StorageDateUtil {

private static final Logger logger = LoggerFactory.getLogger(StorageDateUtil.class);

/\*\*

\* 获取当前时间的下一小时整

\* 计算资源购买价格使用作为价格的开始计算费用时间

\*

\* @return

\*/

public String getStartChargingTime() {

Calendar calendar = Calendar.getInstance();

calendar.set(Calendar.HOUR\_OF\_DAY, calendar.get(Calendar.HOUR\_OF\_DAY) + 1);

SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:00:00");

try {

return df.format(calendar.getTime());

} catch (Exception e) {

logger.debug("时间转换错误 getStartChargingTime" + e);

}

return null;

}

/\*\*

\* 获取当前时间的下后(nonth)个月 的时间

\* 计算资源购买价格使用作为价格的结束计算费用时间

\*

\* @param month

\* @return

\*/

public String getEndChargingTime(int month) {

Calendar calendar = Calendar.getInstance();

calendar.set(Calendar.HOUR\_OF\_DAY, calendar.get(Calendar.HOUR\_OF\_DAY) + 1);

calendar.set(Calendar.MONTH, calendar.get(Calendar.MONTH) + month);

SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:00:00");

try {

return df.format(calendar.getTime());

} catch (Exception e) {

logger.debug("时间转换错误 getEndChargingTime" + e);

}

return null;

}

/\*\*

\* 得到两个时间段相差的分钟数

\*

\* @param startChargingTime

\* @param endChargingTime

\*/

public Long getMinute(String startChargingTime, String endChargingTime) {

try {

SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

java.util.Date now = df.parse(endChargingTime);

java.util.Date date = df.parse(startChargingTime);

long l = now.getTime() - date.getTime();

return ((l / (60 \* 1000)));

} catch (Exception e) {

logger.debug("时间转换错误 getMinute" + e);

e.printStackTrace();

}

return null;

}

}

package com.yingchong.service.yccloud.storage.util;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description : 定义错误码

\* ========================================================

\*/

public enum StorageErrorCode {

SUCCESS("000000", "调用成功"),

SUCCESS\_code("success", "调用成功"),

CODE\_010101("010101","数据库执行异常"),

CODE\_010102("010102","底层发送数据失败"),

CODE\_010103("010103","添加共享存储调用后端服务失败"),

CODE\_010104("010104","删除共享存储调用底层失败"),

CODE\_010105("010105","未找到存储或存储被占用不能执行操作"),

CODE\_010106("010106","共享卷只能扩大,不能缩小"),

CODE\_011001("011001","热加载块设备异常"),

CODE\_011003("011003","解除载块设备异常"),

CODE\_011002("011002","更新数据库异常"),

CODE\_010201("010201","存储名称不正确"),

CODE\_010202("010202","未找到需要修改的数据"),

CODE\_010203("010203","名称不能重复"),

CODE\_010204("010204","获取资源失败"),

CODE\_010205("010205","未找到数据"),

CODE\_010206("010206","资源不足"),

CODE\_010207("010207","被占用的存储不能修改挂载点"),

CODE\_010208("010208","当前存储不能删除"),

CODE\_010209("010209","删除失败请联系管理员");

private String code;

private String msg;

private StorageErrorCode(String code, String msg) {

this.code = code;

this.msg = msg;

}

public String getCode() {

return code;

}

public String getMsg() {

return msg;

}

}

package com.yingchong.service.yccloud.storage;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.netflix.feign.EnableFeignClients;

import com.yingchong.feign.jaxrs.yingchongFeignClientConfig;

import com.wordnik.swagger.config.ConfigFactory;

import com.wordnik.swagger.config.ScannerFactory;

import com.wordnik.swagger.config.SwaggerConfig;

import com.wordnik.swagger.jaxrs.config.DefaultJaxrsScanner;

import com.wordnik.swagger.jersey.JerseyApiReader;

import com.wordnik.swagger.model.ApiInfo;

import com.wordnik.swagger.reader.ClassReaders;

@SpringBootApplication(scanBasePackages = {"com.yingchong"})

//设置basePackages以扫描依赖jar包中的@FeignClient注解，否则依赖jar中的远程接口将无法使用

@EnableFeignClients(defaultConfiguration = {yingchongFeignClientConfig.class}, basePackages = {"com.yingchong"})

@EnableDiscoveryClient

@MapperScan("com.yingchong")

public class StorageApplication implements CommandLineRunner {

public static void main(String[] args) {

SpringApplication.run(StorageApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

SwaggerConfig config = ConfigFactory.config();

config.setBasePath("/v1");

config.setApiVersion("1.0.0");

ApiInfo info = new ApiInfo("存储服务接口", "存储服务接口，服务ID为fs-yccloud-storage", "", "", "", "");

config.setApiInfo(info);

ScannerFactory.setScanner(new DefaultJaxrsScanner());

ClassReaders.setReader(new JerseyApiReader());

}

}

package com.yingchong.service.yccloud.storage.config;

import com.yingchong.common.yccloud.commonapi.util.CloudProperty;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.bind.RelaxedPropertyResolver;

import org.springframework.context.EnvironmentAware;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.env.Environment;

/\*\*

\* =======================================================

\*

\* @Version :0.0.1

\* @Description :

\* ========================================================

\*/

@Configuration

public class StorageConfig implements EnvironmentAware {

// 解析application.yml

private RelaxedPropertyResolver propResolver;

@Autowired

private CloudProperty property;

private static final String storage = "storage";

@Override

public void setEnvironment(Environment environment) {

propResolver = new RelaxedPropertyResolver(environment, "config.");

}

@Bean(name = "vhdPath")

public String vhdPath() {

//return propResolver.getProperty("vhd\_path");

return property.getProperty(storage, "vhd\_path");

}

@Bean(name = "imgName")

public String imgName() {

//return propResolver.getProperty("imgname");

return property.getProperty(storage, "imgname");

}

@Bean(name = "mkfs")

public String mkfs() {

//return propResolver.getProperty("mkfs");

return property.getProperty(storage, "mkfs");

}

@Bean(name = "userLocation")

public String userLocation() {

//return propResolver.getProperty("user\_location");

return property.getProperty(storage, "user\_location");

}

@Bean(name = "volumePath")

public String volumePath() {

//return propResolver.getProperty("volumePath");

return property.getProperty(storage, "volumePath");

}

// @Bean(name = "consulServPath")

// public String consulServPath() {

// return propResolver.getProperty("consulServPath");

// }

@Bean(name = "zookeeperConnectionString")

public String zookeeperConnectionString(){

//return propResolver.getProperty("zookeeper");

return property.getProperty("pub","zookeeper");

}

}

package com.yingchong.service.yccloud.storage.config;

import java.util.HashMap;

import java.util.Map;

import javax.servlet.Filter;

import org.springframework.boot.bind.RelaxedPropertyResolver;

import org.springframework.context.EnvironmentAware;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.env.Environment;

import org.springframework.web.filter.CharacterEncodingFilter;

/\*\*

\* @author yu.ruibo

\*

\* 通用配置类

\*

\*/

@Configuration

public class WebConfig implements EnvironmentAware {

// 解析application.yml

private RelaxedPropertyResolver propResolver;

/\*\*

\* 解决中文内容编码问题，统一用UTF-8编码

\*

\* @return

\*/

@Bean

public Filter characterEncodingFilter() {

CharacterEncodingFilter characterEncodingFilter = new CharacterEncodingFilter();

characterEncodingFilter.setEncoding("UTF-8");

characterEncodingFilter.setForceEncoding(true);

return characterEncodingFilter;

}

@Override

public void setEnvironment(Environment environment) {

// TODO Auto-generated method stub

propResolver = new RelaxedPropertyResolver(environment);

}

public Map<String,String> staticResource(){

Map<String,String> commonStaticResource = null;

try {

commonStaticResource = new HashMap<String,String>();

commonStaticResource.put("hostUrl", propResolver.getProperty("business.hostUrl"));//医疗特殊分类“家庭医生” 前台判断业态使用

} catch (Exception e) {

e.printStackTrace();

}

return commonStaticResource;

}

}

package com.yingchong.service.yccloud.storage.config;

import javax.annotation.Resource;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.retry.ExponentialBackoffRetry;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class ZookeeperConfig {

@Resource(name="zookeeperConnectionString")

private String connectString;

@Bean

public CuratorFramework zkconfig(){

System.out.println("初始化CuratorFramework 。。。");

System.out.println("connectString:" + connectString);

CuratorFramework zk = CuratorFrameworkFactory.

builder()

.connectString(connectString)

.sessionTimeoutMs(5000)

.connectionTimeoutMs(5000)

.retryPolicy(new ExponentialBackoffRetry(1000, 3)).build();

zk.start();

return zk;

}

}

# 参考 http://www.tuicool.com/articles/veUjQba

# Web容器配置

server:

port: 8084

undertow:

accesslog:

# 开启Undertow访问日志

enabled: true

dir: /data/logs

eureka:

#EurekaInstanceConfigBean

instance:

preferIpAddress: true

instanceId: ${spring.cloud.client.hostname}:${server.port}

# instanceId: ${vcap.application.instance\_id:${spring.application.name}:${spring.application.instance\_id:${server.port}}}

client:

registerWithEureka: true

fetchRegistry: true

healthcheck:

enabled: false

serviceUrl:

defaultZone: http://10.10.32.85:8761/eureka/,http://10.10.32.86:8761/eureka/

hystrix:

command:

# Globally config

default:

execution:

# Disable Hystrix timeout globally (for all services)

timeout.enabled: false

# Set Hystrix timeout to 60s globally

#isolation.thread.timeoutInMilliseconds: 60000

# Per service config

#<yourServiceName>:

#execution:

# Disable Hystrix timeout (per service)

#timeout.enabled: false

# Set Hystrix timeout to 60s (per service)

#isolation.thread.timeoutInMilliseconds: 60000

threadpool:

default:

coreSize: 2000

maxQueueSize: 100

queueSizeRejectionThreshold: 20

spring:

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://10.10.20.46:3306/yingchong\_cloud\_21?useUnicode=true&characterEncoding=UTF-8

username: yingchong

password: yingchong2015

rabbitmq:

host: 10.10.27.42

port: 5672

username: admin

password: admin

redis:

timeout: 10000

host: 10.10.20.112

consul:

consulServPath: 10.10.20.204

mybatis:

config: classpath:mybatis-config.xml

# 批量插入或更新时，每次事务执行的插入或更新数据行数

rows\_per\_transaction: 500

# Log 配置 级别大写 TRACE|DEBUG|INFO|WARN|ERROR

logging:

level:

#root: INFO

com.yingchong: INFO

config:

# 创建存储

vhd\_path: /docker-data/works/

#mgname后缀

imgname: .vhd

#磁盘格式

mkfs: ext4

#容器挂在点

user\_location: /data

#共享存储

volumePath: /docker-data/works/

consulServPath: 10.10.20.204

zookeeper: 10.10.27.1:2181,10.10.27.1:2182,10.10.27.1:2183

spring:

application:

name: fs-yccloud-storage-v

http:

encoding: { charset: UTF-8, enable: true, force: true }

cloud:

config:

name: application

profile: eureka-service,mysql-pool,${spring.application.name}

label: dev

uri: http://config.org.yingchong.com

info:

component: fs-yccloud-storage

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<include resource="org/springframework/boot/logging/logback/base.xml" />

<appender name="console" class="ch.qos.logback.core.ConsoleAppender">

<!-- Log message format -->

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %5level --- [%thread] %logger{36} : %msg%n</pattern>

<charset>UTF-8</charset>

</encoder>

<!-- deny all events with a level below DEBUG, that is TRACE -->

<filter class="ch.qos.logback.classic.filter.ThresholdFilter">

<level>DEBUG</level>

</filter>

</appender>

<appender name="sysDailyRollingFile" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>/data/logs/sys.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- daily rollover -->

<fileNamePattern>/data/logs/sys-%d{yyyyMMdd}.log</fileNamePattern>

<!-- 保留30天的日志 -->

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %5level --- [%thread] %logger{36} : %msg%n</pattern>

<charset>UTF-8</charset>

</encoder>

<filter class="ch.qos.logback.classic.filter.ThresholdFilter">

<level>INFO</level>

</filter>

</appender>

<appender name="infoDailyRollingFile" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>/data/logs/jy-info.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- daily rollover -->

<fileNamePattern>/data/logs/jy-info-%d{yyyyMMdd}.log</fileNamePattern>

<!-- 保留30天的日志 -->

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %5level --- [%thread] %logger{36} : %msg%n</pattern>

<charset>UTF-8</charset>

</encoder>

<filter class="ch.qos.logback.classic.filter.LevelFilter">

<level>INFO</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

</appender>

<appender name="severeDailyRollingFile" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/jy-severe.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- daily rollover -->

<fileNamePattern>/data/logs/jy-severe-%d{yyyyMMdd}.log</fileNamePattern>

<!-- 保留30天的日志 -->

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %5level --- [%thread] %logger{36} : %msg%n</pattern>

<charset>UTF-8</charset>

</encoder>

<!-- deny all events with a level below WARN, that is TRACE, DEBUG, and INFO -->

<filter class="ch.qos.logback.classic.filter.ThresholdFilter">

<level>WARN</level>

</filter>

</appender>

<logger name="com.yingchong" level="INFO">

<appender-ref ref="infoDailyRollingFile" />

<appender-ref ref="severeDailyRollingFile" />

</logger>

<logger name="dao" level="DEBUG"/>

<!--log4jdbc -->

<logger name="jdbc.sqltiming" level="debug"/>

<logger name="com.ibatis" level="debug" />

<logger name="com.ibatis.common.jdbc.SimpleDataSource" level="debug" />

<logger name="com.ibatis.common.jdbc.ScriptRunner" level="debug" />

<logger name="com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate" level="debug" />

<logger name="java.sql.Connection" level="debug" />

<logger name="java.sql.Statement" level="debug" />

<logger name="java.sql.PreparedStatement" level="debug" />

<logger name="java.sql.ResultSet" level="debug" />

<!-- the root level of logging -->

<root level="INFO">

<appender-ref ref="sysDailyRollingFile" />

</root>

</configuration>

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<settings>

<setting name="logPrefix" value="dao."/>

</settings>

<plugins>

<plugin interceptor="com.github.pagehelper.PageHelper">

<property name="dialect" value="mysql"/>

<!-- 该参数默认为false -->

<!-- 设置为true时，会将RowBounds第一个参数offset当成pageNum页码使用 -->

<!-- 和startPage中的pageNum效果一样-->

<property name="offsetAsPageNum" value="true"/>

<!-- 该参数默认为false -->

<!-- 设置为true时，使用RowBounds分页会进行count查询 -->

<property name="rowBoundsWithCount" value="true"/>

<!-- 设置为true时，如果pageSize=0或者RowBounds.limit = 0就会查询出全部的结果 -->

<!-- （相当于没有执行分页查询，但是返回结果仍然是Page类型）-->

<property name="pageSizeZero" value="true"/>

<!-- 3.3.0版本可用 - 分页参数合理化，默认false禁用 -->

<!-- 启用合理化时，如果pageNum<1会查询第一页，如果pageNum>pages会查询最后一页 -->

<!-- 禁用合理化时，如果pageNum<1或pageNum>pages会返回空数据 -->

<property name="reasonable" value="true"/>

<!-- 3.5.0版本可用 - 为了支持startPage(Object params)方法 -->

<!-- 增加了一个`params`参数来配置参数映射，用于从Map或ServletRequest中取值 -->

<!-- 可以配置pageNum,pageSize,count,pageSizeZero,reasonable,orderBy,不配置映射的用默认值 -->

<!-- 不理解该含义的前提下，不要随便复制该配置 -->

<property name="params" value="pageNum=pageHelperStart;pageSize=pageHelperRows;"/>

<!-- 支持通过Mapper接口参数来传递分页参数 -->

<property name="supportMethodsArguments" value="false"/>

<!-- always总是返回PageInfo类型,check检查返回类型是否为PageInfo,none返回Page -->

<property name="returnPageInfo" value="none"/>

</plugin>

</plugins>

</configuration>

buildscript {

ext {

springBootVersion = "1.3.1.RELEASE"

springCloudVersion = "Brixton.M4"

}

repositories {

maven { url "http://10.10.10.60:8081/nexus/content/groups/public" }

maven { url "http://repo.spring.io/libs-release" }

maven { url "http://repo.spring.io/libs-milestone" }

maven { url "http://repo.spring.io/snapshot" }

jcenter()

mavenCentral()

}

dependencies {

classpath "io.spring.gradle:dependency-management-plugin:0.5.4.RELEASE"

classpath "org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}"

classpath 'org.sonarsource.scanner.gradle:sonarqube-gradle-plugin:2.0'

}

configurations {

compile.exclude module: "spring-boot-starter-tomcat"

}

}

def env = System.getProperty("profile") ?: "dev"

apply plugin: "java"

apply plugin: "eclipse"

apply plugin: "idea"

apply plugin: "maven-publish"

apply plugin: "spring-boot"

apply plugin: "io.spring.dependency-management"

apply plugin: 'org.sonarqube'

jar {

baseName = "fs-yccloud-storage"

version = "0.0.1-SNAPSHOT"

}

// Configuration中通过package扫描注册Provider，需要在此添加对应jar包

springBoot {

requiresUnpack = ['com.wordnik:swagger-jersey2-jaxrs\_2.10', 'com.yingchong:yingchong-commons-ms']

}

sourceCompatibility = 1.8

targetCompatibility = 1.8

repositories {

maven { url "http://10.10.10.60:8081/nexus/content/groups/public" }

maven { url "http://repo.spring.io/libs-release" }

maven { url "http://repo.spring.io/libs-milestone" }

maven { url "http://repo.spring.io/snapshot" }

jcenter()

mavenCentral()

}

dependencyManagement {

imports {

mavenBom "org.springframework.cloud:spring-cloud-starter-parent:${springCloudVersion}"

}

}

dependencies {

//springframework

compile "org.springframework.boot:spring-boot-starter-undertow"

compile "org.springframework.boot:spring-boot-starter-web"

compile "org.springframework.boot:spring-boot-starter-jersey"

compile "org.springframework.boot:spring-boot-starter-data-mongodb"

compile "org.springframework.boot:spring-boot-starter-actuator"

compile "org.springframework.cloud:spring-cloud-starter-eureka"

compile "org.springframework.cloud:spring-cloud-starter-feign"

compile 'org.springframework.cloud:spring-cloud-config-client'

//common

compile "com.yingchong:yingchong-commons-ms:0.0.10-SNAPSHOT"

compile ("com.yingchong:yccloud-common-domain:0.0.1-SNAPSHOT"){

changing = true

}

compile ("com.yingchong:yccloud-common-commonapi:0.0.2\_$env-SNAPSHOT"){

changing = true

}

compile ("com.yingchong:yccloud-common-remoteservice:0.0.2\_$env-SNAPSHOT"){

changing = true

}

compile "org.springframework.boot:spring-boot-starter-amqp"

// Database Config

compile "com.zaxxer:HikariCP"

compile "mysql:mysql-connector-java"

compile "org.mybatis.spring.boot:mybatis-spring-boot-starter:1.0.1"

// pagehelper

compile "com.github.pagehelper:pagehelper:4.1.0"

//swagger

compile("com.wordnik:swagger-jersey2-jaxrs\_2.10:1.3.8") {

exclude(module: 'jsr311-api')

}

// Utilities

compile "com.google.guava:guava:19.0"

compile "com.alibaba:fastjson:1.2.7"

testCompile "org.springframework.boot:spring-boot-starter-test"

}

eclipse {

classpath {

containers.remove("org.eclipse.jdt.launching.JRE\_CONTAINER")

containers "org.eclipse.jdt.launching.JRE\_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-1.8"

}

}

configurations.all {

resolutionStrategy {

cacheDynamicVersionsFor 1, 'seconds'

cacheChangingModulesFor 1, 'seconds'

}

}

task wrapper(type: Wrapper) {

gradleVersion = "2.9"

}