[现有代码梳理](http://wiki.intra.xiaojukeji.com/pages/viewpage.action?pageId=427159555)

粗略分为三层代码，存储层，缓存层，manager层。读写锁集中在manager层。有些service起了manager层的作用，如AclService。

==============bean，存储层==============================

org.apache.kylin.metadata.model.TableDesc

主要是描述hive的信息和列的信息

org.apache.kylin.metadata.model.ColumnDesc

列基本信息

org.apache.kylin.metadata.modelTableExtDesc

表的kylin附加属性

org.apache.kylin.metadata.projectprojectInstance{

removeModel{

this.getModels().remove(modelName)

}

}

org.apache.kylin.cube.CubeInstance{

getDescriptor{

CubeDescManager.getInstance(config).getCubeDesc(descName)

}

}

org.apache.kylin.metadata.realization.IRealization{

getAllColumns{

if CubeInstance{

getDescriptor().listAllColumns()

}

}

}

org.apache.kylin.cube.model.cubeDesc{

init{

// note CubeDesc.name == CubeInstance.name

ProjectManager.getInstance(config).findProjects{String}

DataModelManager.getInstance(config).getDataModelDesc{String}

//纯计算

initDimensionColumns{}

initMeasureColumns{}

initMeasureReferenceToColumnFamily{}

initDictionaryDesc{}

amendAllColumns{}

// initialize mandatory cuboids based on mandatoryDimensionSetList

initMandatoryCuboids()

}

}

org.apache.kylin.metadata.realization.RealizationRegistry{

Map<RealizationType, IRealizationProvider> providers = Maps.newConcurrentMap()

getRealization{

providers.get

p.getRealization(name){

CubeManager.getCube

or

HybridManager.getHybridInstance

}

}

}

//表级别黑名单，

org.apache.kylin.metadata.acl.TableACL extends RootPersistentEntity{

//user1 : [DB.TABLE1, DB.TABLE2], means that user1 can not query DB.TABLE1, DB.TABLE2

@JsonProperty()

private TableACLEntry userTableBlackList = new TableACLEntry();

@JsonProperty()

private TableACLEntry groupTableBlackList = new TableACLEntry();

deleteByTbl{

//从map中删除

userTableBlackList.deleteByTbl(table);

groupTableBlackList.deleteByTbl(table);

}

}

//持久化入口

org.apache.kylin.common.persistence.ResourceStore{

private static final ConcurrentMap<KylinConfig, ResourceStore> CACHE = new ConcurrentHashMap<>();

getStore{

//double check

CACHE有config直接get返回

synchronized (ResourceStore.class){

CACHE有config直接get返回

创建并putCache，返回ResourceStore

}

}

getResource{

getResourceWithRetry{}

}

deleteResource{

deleteResourceCheckpoint{}

}

deleteResourceCheckpoint{

beforeChange{}

deleteResourceWithRetry{}

}

beforeChange{

Checkpoint cp = checkpointing.get();

cp.beforeChange{

origResData包含则返回

getResourceWithRetry{}

更新origResData,origResTimestamp的Map

}

}

}

==================缓存层======================

org.apache.kylin.metadata.cachesync.CaseInsensitiveStringCache extends SingleValueCache{

ConcurrentSkipListMap内部数据结构

putLocal 只放入map

put 放入map并广播更新

remove 从map删除并广播

}

org.apache.kylin.metadata.cachesync.CachedCrudAssist extends RootPersistentEntity

SingleValueCache<String, T> cache

save{

//Cube，executable是true

checkCopyOnWrite(default false){

被分享或有cache直接报错

}

check 原时间戳，并更新

cache.put

reload{}

}

reload,reloadAt{

从持久化层读取实体，设置实体被分享，

调用子类的initEntityAfterReload

cache.putLocal

返回实体

}

reloadQuietly{

return reloadAt or null

}

delete{

store.deleteResource{}

cache.remove{}

}

}

================manager层，锁一般在这层==================

//广播

Broadcaster{

//只有CubeService，AclService，CuboidRecommender注册了，关注Cube，CubeDesc，Acl。

registerStaticListener{

doRegisterListener{}

}

//各种manager，ExecutableDao，AssignmentCache,其中HybridManager注册了自己和cube

registerListener{

doRegisterListener{}

}

doRegisterListener{

按照string，list放监控器，所有监听器都配置了自身字符串，SYNC\_ALL，SYNC\_PRJ\_SCHEMA，SYNC\_PRJ\_DATA，SYNC\_PRJ\_ACL。或者说这些事件会通知全部的监听器。

}

notifyListener{

先调用普通的，再静态的。

根据事件类型来，找到要通知的list，通知

SYNC\_ALL,清理所有manager。

SYNC\_PRJ\_SCHEMA,清理L2Cache，再调用监听器

SYNC\_PRJ\_DATA，清理L2Cache,再调用监听器

SYNMC\_PRJ\_ACL，清理L2Cache,再调用监听器

}

//通知清理缓存，只用于几个工具

notifyClearAll{}

//project,table,model,cubeDesc,Hybrid各种监听器实体缓存改变后调用相关prj

notifyProjectSchemaUpdate{

}

//cube和project实体改变后调用

notifyProjectDataUpdate{}

//TableAclManager和AclService

notifyProjectACLUpdate{}

}

CacheManager{

getCache{

concurrentMap.get

}

}

org.apache.kylin.rest.service.CacheService{

cleanDataCache{

cacheManager.getCache(QueryService.QUERY\_CACHE).removeAll()

}

//监听cube信息

cacheSyncListener{

onEntityChange{

if cube.UPDATE {

Thread.start{

cubeService.updateOnNewSegmentReady{}

}

}

}

onProjectSchemaChange{

cleanDataCache

}

onProjectDataChange{

cleanDataCache

}

onProjectQueryACLChange{

cleanDataCache

}

onClearAll{

cleanAllDataCache

HBaseConnection.clearConnCache();

}

}

}

org.apache.kylin.rest.security.KylinUserManager{

// user ==> ManagedUser

private CaseInsensitiveStringCache<ManagedUser> userMap;

get{

lock.lockForRead{

userMap.get{}

}

}

}

org.apache.kylin.metadata.acl。TableACLManager{

// user ==> TableACL

private CaseInsensitiveStringCache<TableACL> tableACLMap;

private CachedCrudAssist<TableACL> crud;

loadTableACL{

crud.reload{}

null则new一个

}

deleteTableACLByTbl{

lock.lockForWrite{

loadTableACL{}.deleteByTbl{}

crud.save{}

}

}

}

\* This is a second level cache that is built on top of first level cached objects,

\* including Realization, TableDesc, ColumnDesc etc, to speed up query time metadata lookup.

\* <p/>

\* On any object update, the L2 cache simply gets wiped out because it's cheap to rebuild.

//重建代价似乎不高，但读写锁查的等待代价高

org.apache.kylin.metadata.project.ProjectL2Cache{

private Map<String, ProjectCache> projectCaches = new ConcurrentSkipListMap<>(String.CASE\_INSENSITIVE\_ORDER);

ProjectCache{

private Map<String, TableCache> tables = Maps.newHashMap();

private Set<TableDesc> exposedTables = Sets.newHashSet();

private Set<IRealization> realizations = Sets.newHashSet();

private Map<String, ExternalFilterDesc> extFilters = Maps.newHashMap();

}

sanityCheck{

realization.getAllColumns{

}

for(allColumns){

metaMgr.getTableDesc

sanityCheck

}

}

mapTableToRealization{

for(realization.getAllColumns){

prjCache.tables.get

tableCache.realizations.add

}

}

markExposedTablesAndColumns{

for(realization.getAllColumns){

prjCache.tables.get

prjCache.exposedTables.add

tableCache.exposedColumns.add

}

}

loadCache(prjName){

mgr.getProject

for (prj.tables){

metaMgr.getTableDesc

projectCache.tables.put(tableName, cache)

}

for(getExtFilters){

metaMgr.getExtFilterDesc{}

projectCache.extFilters.put(extFilterName, filterDesc)

}

for(pi.getRealizationEntries()){

RealizationRegistry.getRealization(entry.getType(), entry.getRealization()

projectCache.realizations.add(realization)

}

for(projectCache.realizations) {

sanityCheck{}

mapTableToRealization{}

markExposedTablesAndColumns{}

}

}

getCache{

projectCaches.get{}

result == null{

loadCache{}

projectCaches.put

}

}

getRealizationsByTable{

getCache().tables.get(table).tableCache.realizations

}

listExternalFilterDesc{

getCache{}

}

clear{

prj null则全清，否则remove prj

}

}

org.apache.kylin.metadata.project.ProjectManager{

private ProjectL2Cache l2Cache;

private CachedCrudAssist<ProjectInstance> crud;

// project name => ProjectInstance

private CaseInsensitiveStringCache<ProjectInstance> projectMap;

getRealizationsByTable{

l2Cache.getRealizationsByTable(project, table)

}

clearL2Cache{

l2Cache.clear{}

}

save{

crud.save{}

clearL2Cache{}

}

removeProjectLocal{

prjMapLock.lockForWrite{

projectMap.removeLocal

clearL2Cache(proj);

}

}

reloadProjectQuietly{

prjMapLock.lockForWrite{

crud.reloadQuietly

clearL2Cache(project)

}

}

ProjectSyncListener{

只实现onEntityChange{

if drop{

removeProjectLocal{}

} else {

reloadProjectQuietly

broadcaster.notifyProjectSchemaUpdate

broadcaster.notifyProjectDataUpdate

}

}

}

addTableDescToProject{

prjMapLock.lockForWrite{

getProject{}

for（tables）{

TableMetadataManager.getTableDesc{},

projectInstance.addTable(table.getIdentity())

}

//load半天只为了加个表名？

save{}

}

}

findProjectsByModel{

prjMapLock.lockForWrite{

foreach projectMap.values(){

projectInstance.containsModel

projects.add

}

}

}

removeModelFromProjects{

prjMapLock.lockForWrite{

if findProjectsByModel{

projectInstance.removeModel()

save(project)

}

}

}

addModelToProject{

prjMapLock.lockForWrite{

removeModelFromProjects{String}

getProject{}.addmodel(String)

save{prj}

}

}

getProject{

prjMapLock.lockForRead{

projectMap.get

}

}

removeTableDescFromProject{

prjMapLock.lockForWrite{

getProject{}

TableMetadataManager.getTableDesc{}

projectInstance.removeTable{String table}

save{projectInsrtance}

}

}

findProjects(RealizationType, String){

prjMapLock.lockForWrite{

foreach projectMap.values(){

}

//return list prj

}

}

removeRealizationsFromProjects{

prjMapLock.lockForWrite{

foreach findProjects{RealizationType, String}{

projectInstance.removeRealization{}

save{prj}

}

}

}

addRealizationToProject{

getProject{String}

newProject == null{

//This is a project automatically added when adding realization

createProject{}

}

newProject.addRealizationEntry(type, realizationName)

save{prj}

}

moveRealizationToProject{

prjMapLock.lockForWrite{

removeRealizationsFromProjects{}

addRealizationToProject{}

}

}

}

org.apache.kylin.metadata.TableMetadataManager{

private CachedCrudAssist<TableDesc> srcTableCrud

// name => SourceTableExt

private CaseInsensitiveStringCache<TableExtDesc> srcExtMap;

// table name ==> SourceTable

private CaseInsensitiveStringCache<TableDesc> srcTableMap;

private CachedCrudAssist<TableDesc> srcTableCrud;

private CachedCrudAssist<TableExtDesc> srcExtCrud;

getProjectSpecificTableDesc{

//Make sure the returned table desc is project-specific

//All locks on srcTableMapLock are WRITE LOCKS because of this method!!

srcTableMapLock.lockForWrite{

srcTable如果有归属项目的表，返回。如果有无项目属性的表，深拷贝一份有项目属性的，putLocal，返回。

由于读，写非无项目属性表，所以写锁。

目的是为了不同项目之间共享hive表的元数据。

？表的元数据多大，不共享行吗？都深拷贝了，为什么不持有多份？

}

}

resetProjectSpecificTableDesc{

ProjectManager.getInstance(config).getProject(prj)

srcTableMapLock.lockForWrite{

srcTableMap.get

if 是no prj中拷贝来的

srcTableMap.removeLocal//为了以后重新加载

else

srcTableCrud.reload

}

}

getAllTablesMap{

ProjectManager.getInstance(config).getProject{}

srcTableMapLock.lockForWrite{

prj == null{

foreach srcTableMap.values

return globalTables.put

}

getProjectSpecificTableDesc{}

}

}

getTableExt(TAbleDesc){

srcExtMapLock.lockForRead{

srcExtMap.get 项目

//TODO: notice the table ext is not project-specific, seems not necessary at all

null则srcExtMap.get null

null则返回新对象

}

}

getTableExt(tableName, prj){

getTableDesc{}

getTableExt(TAbleDesc){}

}

saveTableExt{

srcExtMapLock.lockForWrite{

getStore{}

ResourceStore.getResource{}

resource存在，则ResourceStore.deleteResource{}

srcExtCrud.save{}

}

}

getStore{

ResourceStore.getStore{}

}

getTableDesc{

srcTableMapLock.lockForWrite{

getProjectSpecificTableDesc{

}

}

}

saveSourceTable{

srcTableMapLock.lockForWrite{

srcTableCrud.save

}

}

removeTableExt{

srcExtMapLock.lockForWrite{

getTableExt{}

srcExtCrud.delete{}

}

}

removeSourceTable{

srcTableMapLock.lockForWrite{

getTableDesc{}

srcTableCrud.delete{}

}

}

//普通load表的

SrcTableSyncListener{

onEntityChange{

srcTableMapLock.lockForWrite{

if drop{

srcTableMap.removeLocal

} else {

srcTableCrud.reloadQuietly

}

}

TableDesc.parseResourcePath

if (prj == null) {

for (ProjectManager.getInstance(config).findProjectsByTable) {

notifyProjectSchemaUpdate

}

} else {

notifyProjectSchemaUpdate

}

}

}

//作者，更新时间，location这种数据

SrcTableExtSyncListener{

onEntityChange{

srcExtMapLock.lockForWrite{

if drop{

srcExtMap.removeLocal

} else {

srcExtCrud.reloadQuietly

}

}

}

}

ExtFilterSyncListener{

onEntityChange{

extFilterMapLock.lockForWrite{

if drop{

extFilterMap.removeLocal

} else {

extFilterCrud.reloadQuietly

}

}

}

}

}

org.apache.kylin.rest.service.ModelService{

isTableInModel{

DataModelManager.getModelsUsingTable{}

}

}

org.apache.kylin.metadata.model.DataModelManager{

// name => DataModelDesc

private CaseInsensitiveStringCache<DataModelDesc> dataModelDescMap

getDataModelDesc{

modelMapLock.lockForRead{

dataModelDescMap.get,

}

}

getModels{

modelMapLock.lockForRead{

ProjectManager.getProject{}

遍历，getDataModelDesc{}

}

}

getModelsUsingTable{

modelMapLock.lockForRead{

getModels{}

返回model name list

}

}

getAllTablesMap{

TableMetadataManager.getInstance(config).getAllTablesMap{}

}

createDataModelDesc{

modelMapLock.lockForWrite{

dataModelDescMap.containsKey

prjMgr.getProject{}

prj.containsModel{}

saveDataModelDesc{}

prjMgr.addModelToProject{}

}

}

saveDataModelDesc{

dataModelDesc.init{getAllTablesMap{}}

crud.save{}

}

dropModel{

modelMapLock.lockForWrite{

crud.delete

ProjectManager.getInstance(config).removeModelFromProjects{}

}

}

DataModelSyncListener{

onEntityChange{

modelMapLock.lockForWrite{

drop{

dataModelDescMap.removeLocal

} else {

crud.reloadQuietly

}

}

for findProjectsByModel(){

notifyProjectSchemaUpdate{}

}

}

onProjectSchemaChange{

TableMetadataManager.getInstance(config).resetProjectSpecificTableDesc{

}

modelMapLock.lockForWrite{

for ProjectManager.getInstance(config).getProject(project).getModels{

crud.reloadQuietly

}

}

}

}

}

org.apache.kylin.cube.CubeManager{

// cube name ==> CubeInstance

private CaseInsensitiveStringCache<CubeInstance> cubeMap;

// protects concurrent operations around the cached map, to avoid for example

// writing an entity in the middle of reloading it (dirty read)

private AutoReadWriteLock descMapLock = new AutoReadWriteLock();

listAllCubes{

cubeMapLock.lockForRead{

cubeMap.values

}

}

listAllDesc{

descMapLock.lockForRead{

return new ArrayList<CubeDesc>(cubeDescMap.values());

}

}

setCubeMember{

//纯计算

}

//TODO考虑优化

findById{

schedulerCuboidCache.get

if null {

//似乎并发会有问题，为了不锁也是ok

new map and put(String, map)

}

dropCube{

cubeMapLock.lockForWrite{

getCube{}

// remove cube and update cache

crud.delete{}

Cuboid.clearCache(cube){

CuboidManager.getInstance(config).clearCache(cubeInstance)

}

if (deleteDesc && cube.getDescriptor() {

CubeDescManager.getInstance(config).removeCubeDesc{}

}

ProjectManager.getInstance(config).removeRealizationsFromProjects(RealizationType.CUBE, cubeName)

}

}

updateCubeWithRetry{

setCubeMember{}

try{

crud.save{}

} catch{

crud.reload{}

updateCubeWithRetry{}

}

遍历cube.ToRemoveSegments, getStore().deleteResource{}

//this is a duplicate call to take care of scenarios where REST cache service unavailable

ProjectManager.getInstance(cube.getConfig()).clearL2Cache

}

createCube{

cubeMapLock.lockForWrite{

updateCubeWithRetry{}

ProjectManager.getInstance(config).moveRealizationToProject{}

}

}

reloadCubeQuietly{

cubeMapLock.lockForWrite{

crud.reloadQuietly

Cuboid.clearCache

}

}

getCube{

cubeMapLock.lockForRead{

cubeMap.get

}

}

getCubesByDesc{

cubeMapLock.lockForRead{

listAllCubes{}

for cubes{

if name equal

add then return

}

}

}

removeCubeLocal{

cubeMapLock.lockForWrite{

cubeMap.get

!= null{

cubeMap.removeLocal

for segments{

usedStorageLocation.remove

}

Cuboid.clearCache

}

}

}

CubeSyncListener{

onEntityChange{

if drop{

removeCubeLocal()

} else {

reloadCubeQuietly{}

}

}

onProjectSchemaChange{

for projectManager.listAllRealizations{

reloadCubeQuietly

}

projectManager.reloadProjectL2Cache

}

}

}

org.apache.kylin.storage.hybrid.HybridManager{

getHybridInstancesByChild{

lock.lockForRead{

for hybridMap.values{

for hybridInstance.getRealizationEntries{

find

}

}

}

}

getHybridInstance{

lock.lockForRead{

hybridMap.get

}

}

reloadHybridInstance{

lock.lockForWrite{

crud.reload

}

}

HybridSyncListener{

onEntityChange{

if hybrid{

lock.lockForWrite{

if drop{

hybridMap.removeLocal

}else {

crud.reloadQuietly

}

}

for ProjectManager.getInstance(config).findProjects(RealizationType.HYBRID,

hybridName){

notifyProjectSchemaUpdate

}

}

else if cube{

lock.lockForWrite{

for getHybridInstancesByChild(RealizationType.CUBE, cubeName){

crud.reloadQuietly

}

}

}

}

}

}

org.apache.kylin.cube.CubeDescManager{

// name ==> CubeDesc

private CaseInsensitiveStringCache<CubeDesc> cubeDescMap;

private CachedCrudAssist<CubeDesc> crud;

createCubeDesc{

descMapLock.lockForWrite{

cubeDescMap.containsKey(String)

cubeDesc.init

// save resource

crud.save(cubeDesc);

}

}

getCubeDesc{

descMapLock.lockForRead{

cubeDescMap.get

}

}

clearCuboidCache{

// avoid calling CubeDesc.getInitialCuboidScheduler() for late initializing CuboidScheduler

CuboidManager.getInstance(config).clearCache(descName)

}

removeCubeDesc{

descMapLock.lockForWrite{

crud.delete

clearCuboidCache(cubeDesc.getName());

}

}

reloadCubeDescLocal{

descMapLock.lockForWrite{

crud.reload

clearCuboidCache

}

}

reloadCubeDescQuietly{

descMapLock.lockForWrite{

reloadCubeDescLocal{}

}

}

removeLocalCubeDesc{

descMapLock.lockForWrite{

cubeDescMap.removeLocal

clearCuboidCache

}

}

CubeDescSyncListener{

onEntityChange{

getCubeDesc{}

if drop{

removeLocalCubeDesc

} else {

reloadCubeDescQuietly

}

for ProjectManager.getInstance(config).findProjectsByModel{

notifyProjectSchemaUpdate{}

}

}

onProjectSchemaChange{

for ProjectManager.getInstance(config).listAllRealizations{

reloadCubeDescQuietly

}

}

}

}

org.apache.kylin.source.SourceManager{

private final Cache<String, ISource> sourceMap;

this.sourceMap = CacheBuilder.newBuilder().expireAfterWrite(1, TimeUnit.DAYS)

getCachedSource{

createSourceCacheKey{}//build string

sourceMap.getIfPresent != null{

return Isource;

} else {

synchronized(this) {

createSource();

sourceMap.put()

}

}

}

}

org.apache.kylin.job.dao.ExecutableDao{

JobSyncListener{

onEntityChange{

executableDigestMapLock.lockForWrite{

if drop{

executableDigestMap.removeLocal

} else {

executableDigestCrud.reloadQuietly

}

}

}

}

JobOutputSyncListener{

onEntityChange{

executableOutputDigestMapLock.lockForWrite{

if !isTaskExecutableOutput{

if drop{

executableOutputDigestMap.removeLocal

} else {

executableOutputDigestCrud.reloadQuietly

}

}

}

}

}

}

org.apache.kylin.cube.cuboid.CuboidManager{

final private Map<String, Map<Long, Cuboid>> schedulerCuboidCache = Maps.newConcurrentMap();

findById{

schedulerCuboidCache.get

if null {

//似乎并发会有问题，为了不锁也是ok

new map and put(String, map)

}

cubeCAche.get

if (null) {

//找到最匹配的cuboid

cuboidScheduler.findBestMatchCuboid

new Cuboid

cubeCache.put

}

}

}

org.apache.kylin.job.execution.ExecutableManager{

updateJobOutput{

executableDao.getJobOutput

check status

executableDao.updateJobOutput{

writeJobOutputResource{}

if !isTaskExecutableOutput {

executableOutputDigestMap.put

}

}

}

discardJob{

if status is not final{

if (defaultChainedExecutable) {

for tasks{

updateJobOutput discarded

}

}

updateJobOutput discarded

}

}

}

=================业务逻辑层===========================

目前业务方主要的业务流程

1.load hive表（可以指定多个表）

TableController.loadHiveTables

dpp接口验证表权限，超管和数易账户不验证。

->TableService.loadHiveTablesToProject{

extractHiveTableMeta{

ProjectManager.getProject{}

遍历从hive读表meta，

}

loadTablesToProject{

遍历每个表meta，检查是否允许load

TableSchemaUpdateChecker.allowReload{

TableMetadataManager.getTableDesc{}

未加载过直接返回。

findModelByTable{

DataModelManager.getModelsUsingTable{}

}

...

findCubeByTable{

cubeManager.listAllCubes{}

返回在用的cube

}

遍历cube，checkValidationInCube{}

...判断能否重加载,invalidOnIncompatibleSchema{}

}

有不能load的表直接扔运行时异常

遍历表meta{

TableMetadataManager.getTableDesc{}

TableMetadataManager.saveSourceTable{}

extDesc != null{

TableMetadataManager.getTableExt(tableName, prj){}

TableMetadataManager.saveTableExt{}

}

addTableToProject{

TableMetadataManager.addTableDescToProject{}

}

}

}

2.unLoadHiveTables (可以指定多个表)

TableController.unLoadHiveTables

->tableACLService.deleteFromTableACLByTbl{}->TableACLManager.deleteTableACLByTbl{}，

->TAbleService.unloadHiveTable{

TableMetadataManager.getTableDesc{}

true == ModelService.isTableInModel{}{

removeTableFromProject{

ProjectManager.removeTableDescFromProject{}

}

} else {

DataModelManager.getModelsUsingTable{}

}

// it is a project local table, ready to remove since no model is using it within the project

TableMetadataManager.removeTableExt{}

TableMetadataManager..removeSourceTable{}

// remove streaming info

SourceManager.getCachedSource{tableDesc}

source.unloadTable{}

}

3.saveModelDesc

ModelController.saveModelDesc{}

->ModelService.createModelDesc{

DataModelManager.getDataModelDesc{}

TableManager.getTableDesc{}

DataModelManager.createDataModelDesc{}

}

4.deleteModel

ModelController.deleteModel{}

modelService.getDataModelManager().getDataModelDesc

modelService.dropModel{

getCubeDescManager().listAllDesc{}

getDataModelManager().dropModel{}

}

5.saveCubeDesc

CubeController.saveCubeDesc{}

->cubeService.getProjectManager().getProject{},

->cubeService.createCubeAndDesc{

getCubeManager().getCube{}

getCubeDescManager().getCubeDesc{}

getCubeDescManager().createCubeDesc{}

//纯计算

CuboidCLI.simulateCuboidGeneration{}

getCubeManager().createCube{}

}

6.deleteCube

CubeController.deleteCube{}

->cubeService.getCubeManager().getCube(cubeName)

->cubeService.deleteCube{

jobService.listJobsByRealizationName{

getExecutableManager().getAllOutputs{

纯list转map

}

innerSearchCubingJobs{纯计算}

}

this.releaseAllJobs{

jobService.listJobsByRealizationName{

}

for cubingJobs{

if status != succeed && != discarded{

getExecutableManager().

KylinConfig.getInstanceFromEnv().getDistributedLockFactory()

.lockForCurrentThread

if lock(MRHiveDict) {

lock.purgeLocks

}

release other job lock

}

}

}

cube.getProjectInstance{

ProjectManager.getInstance(getConfig()).getProject

}

if hybridRealizationEntries != null{

for hybridRealizationEntries{

getHybridManager().getHybridInstance{}

check needUpdateHybrid{

cubeRealizationEntries.remove

}

if needUpdateHybrid{}

hybridService.updateHybridCubeNoCheck{

aclEvaluate.checkProjectWritePermission{

getProjectInstance{

ProjectManager.getInstance(KylinConfig.getInstanceFromEnv()).getProject

}

}

HybridCubeCLI.main{

hybridManager.getHybridInstance{}

DataModelManager.getDataModelDesc{}

for cubeNames{

cubeManager.getCube{}

}

update{

if checkCubeSize(default true){

checkSegmentOffset{

for realizations{

cubeManager.getCube{}

}

check segment not overlaps, false throw IllegalArgumentException

}

}

store.checkAndPutResource

ProjectManager.getInstance(kylinConfig).moveRealizationToProject{}

hybridManager.reloadHybridInstance{}

}

}

}

}

}

}

cubeNum = getCubeManager().getCubesByDesc{CubeDescManager.getInstance(config).getCubeDesc{}}.size()

getCubeManager().dropCube{name, cubeNum == 1}

cleanSegmentStorage{

if cleanStorageAfterDelOperation(default true, but shuyi use fast delete){

return

}

...

}

if mrHiveDictColumns exist{

final HiveCmdBuilder hiveCmdBuilder = new HiveCmdBuilder();

hiveCmdBuilder.addStatement("drop table if exists " + globalDictDatabase + "." + globalDictTable + "; ");

dictConfig.getCliCommandExecutor().execute(hiveCmdBuilder.build());

}

accessService.clean{

private CaseInsensitiveStringCache<AclRecord> aclMap;

aclService.deleteAcl{

lock.lockForWrite{

递归删子节点findChildren{

lock.lockForRead{

aclMap.values

}

}

crud.delete

}

}

}

}

7.重点，查询

QueryController.query{

QueryService.doQueryWithCache{

KylinUserManager().get{}

//NOW IS FALSE

if isQueryCacheEnabled{

searchQueryInCache{

cacheManager.getCache{}

response.isRunning{

wait 最多一分钟

}

}

}

queryAndUpdateCache{

//default false

if isDummpyResponseEnabled{

cacheManager.getCache(QUERY\_CACHE).put{}

}

if isSelect{

query{

badQueryDetector.queryStart{

runningQueries.put{}

}

queryWithSqlMassage{

QueryConnection.getConnection

if BackdoorToggles.getPrepareOnly{

return getPrepareOnlySqlResponse{}

}

if !isPrepareRequest{

return executeRequest{

conn.createStatement,stat.executeQuery{

OLAPToEnumerableConverter.implement{

RealizationChooser.selectRealization{

attemptSelectRealization{

makeOrderedModelMap{

ProjectManager.getInstance(kylinConfig).getRealizationsByTable

满足条件cube的，计算代价排序，维度数乘1，量度数成10，inner join表数成100。拍脑袋公式

}

QueryRouter.selectRealization{

}

...

}

}

}

...

OLAPEnumerator.moveNext{

queryStorage{

olapContext.getSQLDigest

return ITupleIterator = GTCubeStorageQueryBase.search{

过滤规则，空的segment过滤

每个cubeSeg创建一个scanner

getStorageQueryRequest{

findCuboid{

Cuboid.findCuboid{

//转成long，

toCuboidId{}

findById{

CuboidManager.getInstance(config).findById

}

}

}

}

}

}

}

...

//每个segement一个scanner，每个region一个线程，协处理器聚合region，再聚合了返回

CubeSegmentScanner.iterator{

ScannerWorker.iterator{

CubeHBaseEndpointRPC.getGTScanner{

for getEPKeyRanges{

//static，LoggableCachedThreadPool，共享无限线程，释放时间1分钟

executorService.submit

}

return StorageResponseGTScatter

}.iterator(){

}

}

}

}

}

} else {

getProjectManager().getProject

if getConfig().isQueryPreparedStatementCacheEnable && prepareSqlRequest.isEnableStatementCache{

...

}

}

}

badQueryDetector.queryEnd{

runningQueries.remove{}

notify{

记录log，读写hbase

}

}

QueryConnection.getConnection{}

OLAPContext.clearThreadLocalContexts{}

}

}

}

logQuery{}

recordMetric{}

}

}

//更新缓存

CacheController.wipeCache{}

->CacheService.notifyMetadataChange{}

//执行静态监听器,只有test时不执行,具体

->Broadcaster.notifyListener{...true}

}

8.更新segment

UpdateCubeInfoAfterBuildStep.doWork{

cubeManager.promoteNewlyBuiltSegments{

cubeMapLock.lockForWrite{

segAssist.promoteNewlyBuiltSegments{

getCube{}.latestCopyForWrite{}

...udpatestatus

updateCube{

cubeMapLock.lockForWrite{

updateCubeWithRetry{

(Segments) cube.getSegments().clone()

...add and remove

cube.setSegments(clone)

//this is clone cube

crud.save(cube)//retry 7

for removeSegments{

getStore().deleteResource{}

}

ProjectManager.getInstance(cube.getConfig()).clearL2Cache

}

}

}

}

}

}

}

mergesegment类似，

UpdateCubeInfoAfterMergeStep.doWork{

cubeManager.promoteNewlyBuiltSegments{}

}

org.apache.kylin.cube.cuboid.CuboidManager{

final private Map<String, Map<Long, Cuboid>> schedulerCuboidCache = Maps.newConcurrentMap();