云归档的一些

1.

欢迎使用阿里云归档存储。归档存储作为阿里云数据存储服务体系的一部分，提供了极低成本的数据存储服务。适合各种离线数据，包括但不局限于数字媒体、企业归档资料、科学实验数据、个人备份等。用户可以将数据进行几个月、几年甚至几十年的长时间存储。同时不需要任何基础设施的投入，即可拥有无限扩容、可选择地域的低成本数据归档方案，也免去了后续数据的管理维护成本。需要注意的是，存放于归档存储的数据大部分时间会处于“休眠”状态，而对数据的“唤醒”操作需要消耗一些时间和资源，因此如果您对数据长期稳定存储的需求较高，而对数据的访问和读取速度要求较低的话，归档存储会是一个理想的选择。

2.

@测试部-楼静斐 黄铿负责云归档的API接口整理开发。你发一个一档系统的地址，让他熟悉一下程序怎么用的。

云归档测试环境地址：http://192.168.10.77:9200/

管理员账号：100408 密码：abc123

医生账号：100006 密码：tomtaw001

@hk

3.

svn地址

https://SOURCE-SVN/svn/eWorldCloud/ICIDCAS

4.

用了开源接口管理



5.

曹兵说到，Common的剥离，也想让我做。

说到log问题，没有把参数序列化传入，应该要这样做的。

6.

测试时间的方式：

var swcout = new Stopwatch();

swcout.Start();

\_log.Info("app获取检查详情开始");

var resultdata = \_imageArchiveClient.Invoke(t => t.GetPatientExamInfo(query.ToQuery()));

swcout.Stop();

\_log.Info($"app获取检查详情，耗时{swcout.ElapsedMilliseconds}毫秒");

swcout.Restart();

\_log.Info("app开始地址转换");

7.

DocumentService似乎就是IMCIS。



8.

采集标识：

兵哥 这个CollectionMark 采集状态 这些1000 1144 1111是怎么定义的？

曹兵 2019/2/26 10:21:14

1000是未采集 初始化状态

1144 是检查变更需要重新采集检查

1111是采集完成的

1是变更需要采集（1是采集完成，2才是变更需要重新采集） 0是未采集 4是不存在

第一位是占位符 第二位是检查 第三位是影像 第四位是胶片

自己的扩展：1044，那就是没有影像和胶片，只有检查，还未采集？

DataSource：RIS（放射）、UIS（超声）、EIS（内镜）、XIS（特检）、ECG（心电）、PIS（病理）、Custom（病人自建）

检查类型查找：

/// 检查的为检查类型(见字典ExamType)，如CR、CT、US、ES。检验为检验类型，如血液、体液、生化、免疫、微生物、分子生物学等；测量见测量类型(见字典MeasureType)，如体温、脉搏、呼吸、血压、大便次数、出入量、身高体重等

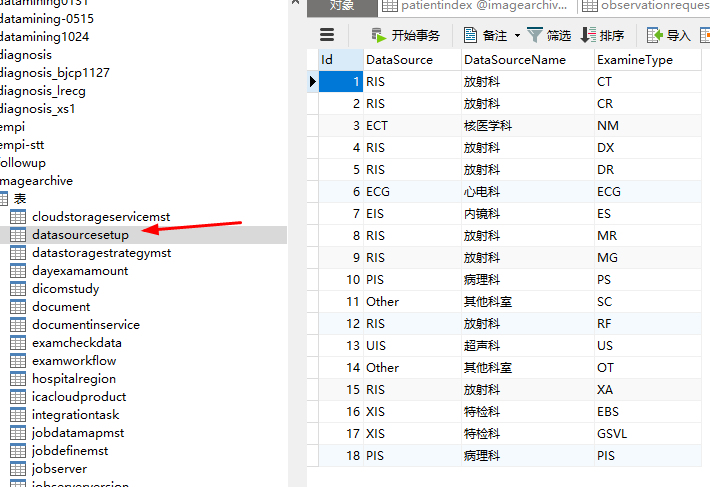
/// </summary>

public string ServiceSectID { get; set; }

字典数据库中：



整体的数据库：

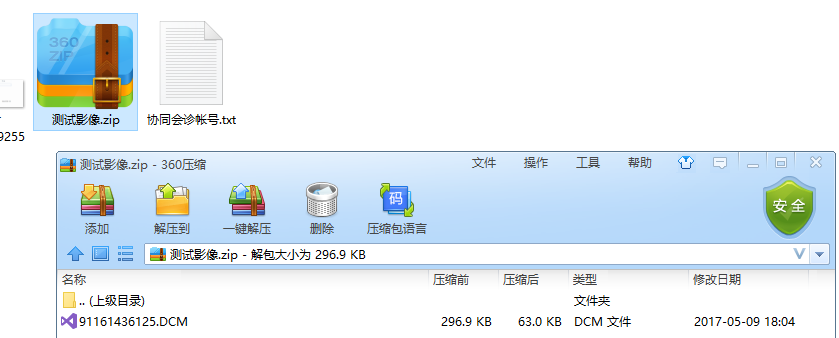


9.

上传影像这里，我发现了，原来协同这边，上传的时候，调用的也是陈赵丹那边的DocumentService的接口。



内部就是dcm文件：



//上传文件

UploadFile: function () {

vm.submitBtnState = true;

if (vm.applyInfo.IsResubmit == 1) {

if (vm.imgaeFile.name != null && vm.fileList.length != 0) {

vm.DeleteFile(vm.fileList[0]);

} else if (vm.imgaeFile.name == null && vm.fileList.length != 0) {

//提交会诊

vm.SubmitApply();

return;

}

} else {

vm.applyInfo.ServiceId = vm.ServiceId;

}

if (vm.imgaeFile.name == null) {

layer.msg('请选择影像文件');

vm.submitBtnState = false;

return;

};

var fileName = vm.imgaeFile.name;

var fileSplitSize = 1 \* 1024 \* 1024;　　//文件分段大小5M

var size = vm.imgaeFile.size;

var index1 = fileName.lastIndexOf(".");

var index2 = fileName.length;

var suffix = fileName.substring(index1, index2);//后缀名

var posturl = vm.IMCISStorageInternetUrl + "/DocumentService/Upload/Document?"

+ "BusinessId=" + vm.applyInfo.ServiceId

+ "&BusinessType=ExamDiagnosis"

+ "&ClassCode=DiagnosisRequest"

+ "&TypeCode=ExamImage"

+ "&FormatCode=DICOMDIR"

+ "&OrganizationID=" + vm.Institution.InstitutionlCode //增加InstitutionlCode，方便文件保存时，按照服务中心所属机构code分类保存

+ "&Token="

+ "&Position=" + vm.fileStart

+ "&FileName=" + fileName

+ "&FileSize=" + size;

var formData = new FormData();

if (vm.IMCISStorageInternetUrl == "") {

vm.submitBtnState = false;

layer.msg("系统参数出错，上传失败！");

} else {

formData.append("file", vm.imgaeFile.slice(vm.fileStart, vm.fileStart + fileSplitSize));

formData.append("name", fileName);

$.ajax({

url: posturl,

type: 'POST',

data: formData,

processData: false,

contentType: false,

success: function (data) {

vm.fileLoad.show = true;

if (vm.fileStart + fileSplitSize >= size) {

vm.fileLoad.text = '上传完成，已上传100%';

vm.fileLoad.width = '100%';

vm.SubmitApply();

} else {

vm.fileStart += fileSplitSize;

var Percentage = Math.round(vm.fileStart / size \* 100);

vm.fileLoad.width = Percentage + '%';

vm.fileLoad.text = '正在上传，已上传' + Percentage + '%';

vm.UploadFile();

}

},

error: function (data) {

vm.fileStart = 0;

vm.fileLoad.show = false;

vm.fileLoad.width = 0;

vm.submitBtnState = false;

if (data.responseText.indexOf("MakeDicomDir失败") >= 0) {

layer.msg('影像上传失败，请确保压缩包里面的文件必须是dcm文件！');

} else {

layer.msg(data.responseText);

}

}

});

}

},

10.

采集整理下：

如果登陆成功，但采集服务没有注册，采集工具上有个按钮，直接跳到云端，去注册（因为要配置采集策略，在Web上去配，TriggerUID对应的key-value对）。

当成功登陆后，采集工作站有一个后台线程一直在跑，会从后台获取云端的控制码，去重启啊，去更新啊。

这里有个字段——ControlCode（控制代码），0：运行1：停止2：重启。JobServer库中，一个采集服务站，对应的ServiceUID是唯一的。

采集工具通过这个字段去判断是否要运行，停止，还是重启。而这个状态值，我们可以在云端（web）上面去修改，这样我们就可以控制采集工作站的运行状况。

登陆，获取到当前管理员对应的采集服务的ServiceUID（服务UID），通过ServiceUID可以去jobtriggermst中获取和JobUID，还有TriggerUID的对应关系，这里是三个UID确定一条数据。其中ServiceUID和另外两个UID是一对多关系。

JobUID的固定的，对应着采集服务（job）中的程序集名字，命名空间名，以及描述，也就是该服务的用途，具体是检查影像上传还是影像，胶片上传，上传到哪，IDCAS还是哪。好像目前的终端都是IDCAS云。

这里的一个ServiceUID会对应多个，比方三个的JobUID，检查采集，影像采集，胶片采集。而这三条数据都会有一个全新的，唯一的TriggerUID（JobUID和ServiceUID会看见重复的）。这个唯一的TriggerUID对应着采集策略（key-value对），也就是配置在web上面的，检查采集，影像采集，胶片采集的具体策略配置。

（

就是我有几个线程任务，几个触发器，几个作业，每个作业的触发参数是什么，比方我这个采集工作站有三个采集任务，检查，影像，胶片，什么时间去执行，执行多少次，按什么频率去执行。平台上（web）上现在是没法配置的，参数都是固定的，在jobtriggermst表中。

这个接口和下面的获取具体配置key-value是分开来的，不然多层嵌套，比方检查，影像，胶片，三个参数，每个里面还有很多的key-value对，或者说序列号后的string（比方影像的dicom参数）。

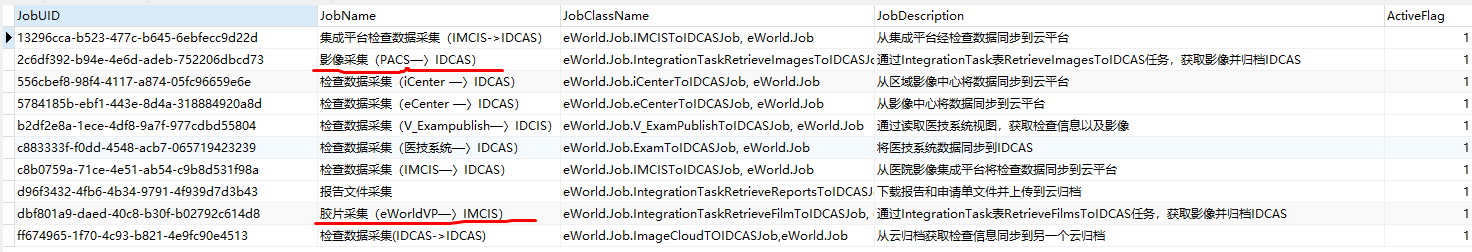
key-value对，其实就是每个任务线程的执行策略（参数）。

其实这些配置，也就是采集工具运行了之后的配置，比方影像，具体传到哪，怎么传，等等，这些参数都是在采集策略的键值对当中。通过Web接口配置到数据库中，下次直接获取。因为http是短连接，无记忆的。便要保存在数据库中。

）

无论是哪种采集策略，也就是ServiceUID对应的JobUID，无论是哪种ServiceUID，他必然对应着检查采集这个JobUID，因为是必须的。检查采集是必须存在的。所以这个Job任务应该是默认就跑着，只要管理员登陆了采集工具，它就默认跑起来（还可以通过ControlCode控制），开始采集检查数据，然后如果需要采集影像和胶片的话，才生成新的，影像或是胶片的，采集集成任务。而影像或是胶片的任务线程，在一开始，ServiceUID对应JobUID的时候，如果一对多是3条，那三个服务就自动跑着了。影像和胶片线程就一直在integrationtask库中寻找着各自的任务，一旦有（也就是集成任务注册进来了），立马开采，传到云端。

这里本以为检查采集是公用的，没想到每个不同的采集方式，比方从区域影像，或者影像中心，各自的检查采集所用的类啊，命名空间啊，是不一样的，反而影像采集和胶片采集是公用的。



检查记录注册采集后，根据是否需要采集影像和胶片，生成对应条数的采集集成任务，然后去执行。采影像和采胶片的线程就获取到这两条集成任务去执行采集，采集成功后要修改采集状态，以及是否拥有影像的状态，胶片的数量。

这里影像和胶片的上传，调用的是赵丹那边封装的一个dll。和调用主动注册里面的文档上传，道理是一样的。

赵丹那边封装了一个上传到S3的，上传到对象存储，或者上传到虚拟目录，把这个封装成了一个dll文件。让我们采集服务去调。这样我们这边就不用配置这个地址了。

有一个获取DocumentService配置信息的接口，没有放进来。

这个动态库里面是没有做用户验证的，只要拿到我们这边的账号（对象存储的账号信息），就可以去上传了。拥有一切读写的权限。

调这个dll时候，要把对象存储的地址，账号，密码，发给赵丹。来成功上传。

如果来调我们的接口，就要通过我们的服务器，那我们的服务器，就成为我们的瓶颈了。本来大家都可以来操作这个对象存储，就飞快。

之前的干法：

如果上传的是一张一张的dicom，我们服务端要生成dicomdir，再传上去。

之前要求把dcm和DicomDir打包成一个压缩包。

现在可以这样：

把影像采集到前置机，前置机生成DICOMDIR，然后把图片一个个，包括DICOMDIR，，一张张上传到对象存储。

这样干的话，就和服务端没有关系了。

关键点：这个动态库能不能保证安全。

缺失的地方，后台没有去处理已经完成的采集任务，导致这个库越来越大，要去物理删除掉。在亮哥这边，加后台服务去处理。定期把成功的，或者失败的，给删除掉。

20天之前的可以删掉。20天内的还需要查询的。

采集服务在注册完检查记录后，对于还需要采集的影像、电子胶片等，生成采集任务保存到平台的任务表，由采集服务的后台线程获取任务并执行。

采集服务从平台上查询保存的采集任务，并通过后台线程去执行。

更新集成任务的执行状态：

采集服务后台现场对于执行成功或失败的任务，调用该接口更新任务状态。

小总结，采集服务中检查注册：

获取视图，读取各种配置，包括采集策略（数据库地址这些都在里面），配置查询视图的sql语句，把这些配置中的检查时间，时间戳，机构代码等配置到where条件中，sql语句执行查询，获取结果，for循环遍历DataTable，每一条数据都要校验后（上传状态到校验表），然后拆分成我们的病人，就诊，观察申请，观察结果这样的格式，再通过注册接口注册到云端，再把检查注册状态更新到云端的校验表中，接着从视图下载申请单和报告单，先下载到本地，再通过陈赵丹的dll上传到DocumentService中，拿到返回结果，写入对应ObservationUID的Document表，以及Media表中(现在报告校验放在这里面，其实其他的也都可以放在这里面，但是分散到检查注册，改变影像状态和胶片数量上去了，成功传入即为10)。更新到并计算本次检查消耗的磁盘空间，保存到ObservationRequest表中。并把上传报告状态（10）传到云端的校验表中。

接下来生成影像采集任务和胶片采集任务。先通过ServiceUID和JobUID，未删除，激活状态为true去查询jobtriggermst表，如果策略中点击选中了影像采集，那么这里肯定有一条对应的TriggerUID的值。这里是三个UID确定一条数据。如果确定影像采集或是胶片采集是必要的，则去集成任务表中查询是否存在当下BusinessUID的影像或是胶片采集任务。

当存在这么个影像或是胶片的采集任务，通过当前的BusinessUID=ObservationUID去获取当前存在的影像或是胶片的采集集成任务，只要状态不是正在处理，就删除，然后重新生成一条状态为0，也就是等待处理的集成任务。这样采集影像和胶片的线程就能获取到，并去采集了。

生成新的影像和胶片采集任务后，更新时间戳，并将新的时间戳保存到采集策略的key-value对中。

最后注册心跳。

具体：

其中的检查采集，将数据库中的供采集服务获取检查数据的视图名字配置在app.config中。这样部署在生产环境后，只要建立了视图，再把视图名字配置在config文件中就可以了。而数据库的ip，密码这些，是配置在采集策略的key-value对中的。以及组织机构啊，名字啊，具体时间戳，都是配置在web页面的策略key-value对中，job服务拿到这些参数，才开始拼接sql语句，去执行（比方检查时间大于value中的检查时间，时间戳大于value中的时间戳，只要就是拼接where条件）。再调用我这边的接口，通过SaveExamInfo方法传到云端，也就是我这边接口连接的数据库地址，现在是77或者202，但生成环境，类似宁波那边，就是一个地址，然后六七八家宁波地区的检查，影像，胶片资料都往这个中心云上传。而如果，要把宁波，台州，丽水的各个云都往杭州的中心云上传，是不是就是云到云传输了？

<add key="ExamDBViewerName" value="V\_Exam" />

检查采集的前提：

基于Sql Server或者ORACLE的检查日志视图或表，将医技系统的检查数据同步到集成平台，要求医技系统建立相关的表或视图。

其中时间戳，比方测试环境配置的是：





string timestamp = StringKeyDataMap.GetString(triggerMaplist, "Timestamp");//时间戳（时间控件选择的值）

执行了sql语句后，就获取了要采集，比方row是100条的DataTable，然后Foreach去遍历，再一条条传到云端。

var returnmmodel = ExamCollect(OrganizationID, exam, service, triggerMaplist, media, setting);

在上传前，会校验参数，但是这个接口，之前开会去除了。到时候也不知道怎么处理。

发现这个接口虽然调用，但没有实质的用途（上面虽然调用了接口，但是没有接收返回结果，然后去判断。反而又自行去判断所有的参数，又做了一边参数校验。）但细一想，用途就是把每天记录的，检查注册，报告，影像，胶片的上传结果都记录在了云端。

WebApiServiceRequest.ExamDataCheck(new ExamDataCheckResultRequestQuery()

{

DataSource = dataSource,

DataSourceName = dataSourceName,

ExamIndex = drobs.GetColumnValue("ExamineID"),

ExamineType = drobs.GetColumnValue("ExamineType"),

HospitalCode = institutionCode,

HospitalName = institutionName,

PerformTime = drobs.GetColumnValueDateTime("PerformTime"),

CollectionTime = DateTime.Now,

ReportTime = drobs.GetColumnValueDateTime("ReportTime"),

RegistFlag = 0,这四个状态

UploadFilmFlag = 0, 这四个状态

UploadImageFlag = 0, 这四个状态

UploadReportFlag = 0这四个状态

});

内部又判断所有的参数，又做了一边参数校验。

#region 数据校验

if (string.IsNullOrWhiteSpace(drobs.GetColumnValue("ExamineID")))

{

dataCheck = false;

errorInfo += "检查记录号ExamineID不能为空,";

}

if (string.IsNullOrWhiteSpace(drobs.GetColumnValue("IdNumber")))

{

dataCheck = false;

errorInfo += "身份证号IdNumber不能为空,";

}

……等等校验

校验完成后，将从视图采集的数据按我们的格式，分开来放进病人主索引类，就诊信息类，观察申请，观察结果表的各个字段中，其中观察结果，发现只采集到了影像所见和影像诊断，后来提出的检验结果，还没有采集进来。估计后续要增加。

接下来就是调用我这边的接口，把检查信息注册上来。（http://ip:port /api/cloud\_archive/job/register\_examinfo）

上传结束后，又调用了一次校验检查信息的接口。

WebApiServiceRequest.ExamDataCheck(new ExamDataCheckResultRequestQuery()

{

DataSource = dataSource,

DataSourceName = dataSourceName,

ExamIndex = drobs.GetColumnValue("ExamineID"),

ExamineType = drobs.GetColumnValue("ExamineType"),

HospitalCode = institutionCode,

HospitalName = institutionName,

PerformTime = drobs.GetColumnValueDateTime("PerformTime"),

CollectionTime = DateTime.Now,

RegistFlag = 10

});

这里的RegistFlag：

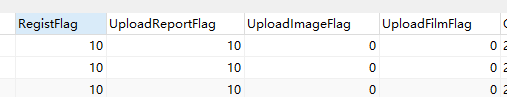
/// <summary>

/// 上传检查标示(-3 已删除 -2 校验失败 -1 上传失败 0 未上传 10 已上传 20 校验成功)

/// </summary>

public Nullable<int> RegistFlag { get; set; }

感觉校验表的意义，在于可以看见检查注册，报告，影像，胶片这些的上传状态。



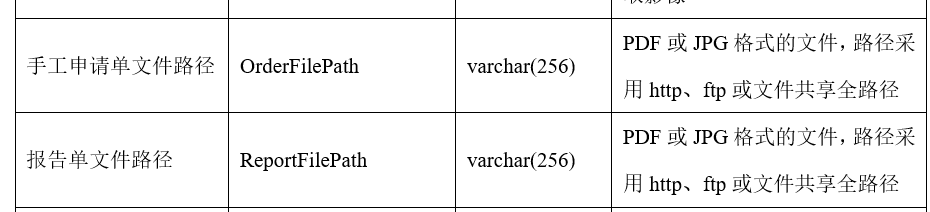
RegistFlag: -3 已删除 -2 校验失败 -1 上传失败 0 未上传 10 已上传 20 校验成功

UploadReportFlag: -2 报告不完整 -1 上传失败 0 未上传 5 无报告 10 已上传 20 校验成功

UploadImageFlag: -2 影像不完整 -1 上传失败 0 未上传 5 无影像 10 已上传 20 校验成功

UploadFilmFlag: -2 胶片不完整 -1 上传失败 0 未上传 5 无胶片 10 已上传 20 校验成功

检查注册之后，就是申请单和报告单的上传，对应着视图中的字段：



将申请单和报告单下载到本地，然后分别将文件和报告单上传到我们自己的云端。

wc.DownloadFile(fileDownloadUrl, zipFileName);

申请单的上传：

UploadFileResponse uplodDirctoryRepose = DocumentClient.UploadFile(wc.DownloadData(requestFileURL)在这里下载申请单, service, docMetadata, true);

这里用的上传，应该就是兵哥讲的，陈赵丹那边封装的dll。

// Assembly location: E:\some\work\ICIDCAS\trunk\eWorld.JobServer\packages\TomTaw.IMCIS.DocumentSDK.1.0.0.73211\lib\net40\TomTaw.IMCIS.DocumentSDK.dll

接着保存文档存储信息，开始操作Document表，DocumentInService表，和Mediamst表了。这里主要就是将文件上传到DocumentService那边后，返回的参数分割，传到云归档这边的表中，这样下次根据UID就可以获取到所有的参数，重新给拼接起来，再加上DocumentService的头，就可以浏览该文件了。

string webApiUrl = AppConfig.WebApiUrl() + "/Job/SaveDocumentStorageList";

报告单的上传流程和申请单也是一样的，先下载到本地，然后上传给DocumentService的dll获取返回结果，再把结果往Document和Media表里传。

接着又调用一次数据校验接口，把UploadReportFlag = 10这个状态给保存到数据库中的校验表中。

WebApiServiceRequest.ExamDataCheck(new ExamDataCheckResultRequestQuery()

{

DataSource = dataSource,

DataSourceName = dataSourceName,

ExamIndex = drobs.GetColumnValue("ExamineID"),

ExamineType = drobs.GetColumnValue("ExamineType"),

HospitalCode = institutionCode,

HospitalName = institutionName,

UploadReportFlag = 10

});

最后更新检查存储空间，这里是将本次检查采集后的申请表，报告表上传的消耗空间计算出来，单位是mb，然后记录到ObservationRequest观察申请表，该ObservationUID对应的记录中。

WebApiServiceRequest.UpdateExamDiskSpace(new ExamDiskSpaceInfo()

{

ObservationUID = observationUID,

//DiskSpace = fileSizes \* 1024//返回的是KB

});

接下来生成影像采集任务和胶片采集任务。

ImageCollect(serviceUID, OrganizationID, returnmmodel.Item2, exam);

影像采集内部：

显示调用——根据条件查询Job触发器及Job定义列表信息

这里根据ServiceUID和JobUID，未删除，激活状态未true去查询，如果策略中点击选中了影像采集，那么这里肯定有一条对应的TriggerUID的值。

JobTriggerQuery JobTriggerQuery = new JobTriggerQuery

{

ServiceUID = Guid.Parse(ServiceUID),

JobUID = Guid.Parse(JobUID.RetrieveImagesToIDCASJob),

DeleteFlag = false,

ActiveFlag = true

};

var jobTriggerWithJobDefine = WebApiServiceRequest.GetJobTriggerWithJobDefineList(JobTriggerQuery).FirstOrDefault();

string webApiUrl = AppConfig.WebApiUrl() + "/Job/GetJobTriggerWithJobDefineList";

开始配置影像采集集成任务

//配置影像采集作业

#region 创建采集影像任务

IntegrationTaskQuery queryInteTask = new IntegrationTaskQuery

{

Initiatemodule = jobTriggerWithJobDefine.JobClassName,

InitiateUser = jobTriggerWithJobDefine.TriggerUID.Value.ToString(),

InteractionType = "RetrieveImagesToIDCAS",

BusinessKeyUID = observationUID,

OrganizationID = organizationID

};

var imageTaskList = WebApiServiceRequest.GetIntegrationTaskList(queryInteTask);

如果该影像采集任务已经存在，只要采集状态（TaskStatus）（任务状态，0：待处理，1：处理中，2：处理成功，3:处理失败。）不是正在处理中，就将该集成任务删除，然后生成新的影像采集任务。

if (imageTaskList != null && imageTaskList.Count() > 0)//是否存在采集影像的任务

{

DeleteIntegrationTaskQuery query = new DeleteIntegrationTaskQuery();

if (!imageTaskList.Any(i => i.TaskStatus == (byte)IntegrationTaskStatusEnum.Processing))

{

query.TaskList = imageTaskList.ToList();

}

else

{

imageTaskList = imageTaskList.Where(i => i.TaskStatus == (byte)IntegrationTaskStatusEnum.Fail || i.TaskStatus == (byte)IntegrationTaskStatusEnum.Success ||

i.TaskStatus == (byte)IntegrationTaskStatusEnum.Waiting);

query.TaskList = imageTaskList.ToList();

}

WebApiServiceRequest.DeleteIntegrationTask(query);

}

var task = new IntegrationTaskInfo();

task.InitiateHost = CommonUtil.GetIP(); ;

task.InitiateModule = jobTriggerWithJobDefine.JobClassName;

task.InteractionType = "RetrieveImagesToIDCAS";

task.BusinessKeyUID = observationUID;

task.BusinessKey = observationUID.ToString();

task.OrganizationID = organizationID;

task.BusinessKeyExt1 = drobs.GetColumnValue("PatientID");

task.BusinessKeyExt2 = drobs.GetColumnValue("AccessionNumber");

task.BusinessKeyExt3 = drobs.GetColumnValue("ExamineType");

task.CommunicationType = "Send";

task.TaskStatus = (byte)IntegrationTaskStatusEnum.Waiting;

task.RetryCount = 0;

task.InitiateUser = jobTriggerWithJobDefine.TriggerUID.Value.ToString();

task.PerformTime = drobs.GetColumnValueDateTime("PerformTime");

WebApiServiceRequest.SaveIntegrationTask(task);

#endregion

生成胶片采集任务，流程和上面的影像采集是一样的。

FilmCollect(serviceUID, returnmmodel.Item2, OrganizationID, exam);

接着更新时间戳：

var rowTimestampNew = string.Empty;

nExamTimespanType：根据医技视图时间戳字段类型, 0: Int型数值类的值；1: 非数值类的值。默认为0

if (nExamTimespanType == 0)

{

rowTimestampNew = exam.GetColumnValue("RowTimestampBigint");

}

else

{

rowTimestampNew = exam.GetColumnValue("RowTimestamp");

}

再把新的时间戳更新到策略的key-value对表中。

JobTriggerDataMapInfo model = new JobTriggerDataMapInfo()

{

TriggerUID = triggerUID,

EntryKey = "TimestampKey",

EntryValue = rowTimestampNew

};

WebApiServiceRequest.SaveJobTriggerDataMap(model);

最后注册心跳。

采集影像的步骤：

先是获取采集任务：

tsk = context.JobDetail.JobDataMap["IntegrationTask"] as IntegrationTaskInfo;

if (tsk == null)

{

throw new Exception("IntegrationTask不能为空");

}

taskList.Add(tsk);

再是获取任务运行所需参数，也就是获取采集策略中，影像的key-value值，key是HosipitalList。

this.GetJobParams(context);



开始逐个处理任务：

// 开始逐个处理任务

foreach (var task in taskList)

{

logger.Info(LogType.CollectLog, $"{JobDesc}开始执行任务, TaskID: {task.TaskId}");

将状态置为1，正在处理，更新到数据库。

task.TaskStatus = 1;

task.ProcessHost = CommonUtil.GetIP();

task.ProcessTime = DateTime.Now;

var rsp = WebApiServiceRequest.SaveIntegrationTask(task);

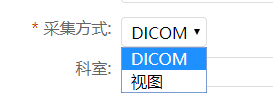
获取检查详细信息：

examInfo = this.GetExamInfo(task);

string webApiUrl = AppConfig.WebApiUrl() + "/v2/public/get\_exam\_info";

如果采集策略中，上传影像的配置，key——hospitalList是配置了的，那么走新的上传方式，否则走老的方式：

新的方式：



如果采集方式是Dicom：

studyInstanceUIDList = this.RetrieveWithFODicomMode(task, ref dcmInfo, examInfo);

如果retrieveImageMethod是EWorldCloud

else if (string.Compare(retrieveImageMethod, "EWorldCloud", true) == 0)

this.RetrieveWithEWorldCloudMode(examInfo, task, imageModel);

生成一个影像下载的临时目录：

string strTempPath = TempFile.GetImageTempPath(this.observationUID);

老的方式：

（未完）

云到云的步骤：

解析UID：

// 解析UID

if (!this.GetJobUIDs(context))

{

throw new StopMomentException();

}

this.GetJobUIDs(context)：方法内部获取三个UID：

this.serviceUID = Guid.Parse(context.JobDetail.JobDataMap.GetString("ServiceUID"));

this.triggerUID = Guid.Parse(context.Trigger.Key.Name);

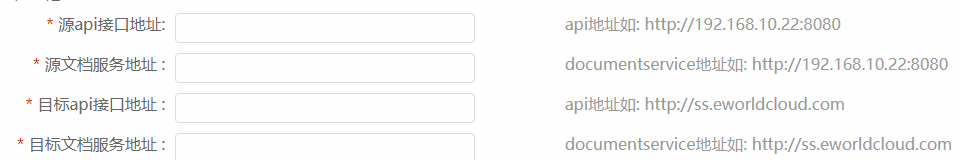
this.jobUID = Guid.Parse(context.JobDetail.Key.Name);

获取任务执行参数，就是采集策略里面的key-value对：

// 获取任务执行参数

this.GetJobParams(context);

涉及源和目标api地址，以及两个DocumentService地址



获取任务配置：

// 获取任务配置

this.settings = WebApiServiceRequest.GetSetting().Result;

判断触发器是否有效：

// 判断触发器是否有效

if (!this.IsValidTrigger())

{

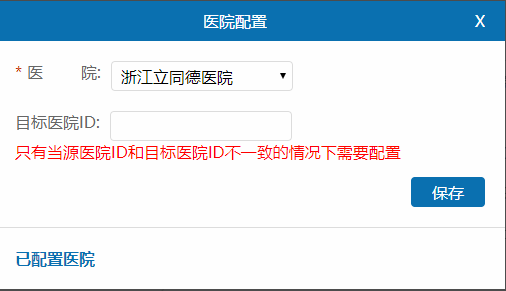
throw new StopMomentException();

}

获取可采集的目标机构代码，两个云上的医院ID如果是一样，就不需要配置了。比方目标是三个医院，那获取了三个医院的机构ID，对应的，从源IDCAS中，同样这三个医院中，把所有的数据采集上来。

// 获取可采集的目标机构代码

this.GetCollectOrganizationID();



获取云到云采集列表：

var result = WebApiServiceRequest.GetPatientExamList(this.sourceImageCloudApiUrl, query);

string webApiUrl = apiurl + "/Job/GetPatientExamList";

for循环遍历所有查询到的检查列表：

foreach (var item in result)

{

获取目标机构ID

string strTargetOrganizationID = this.GetMapTargetOrganizationID(item.observationRequest.OrganizationID);

获取文档服务。每条记录的存储服务可能不一样

this.GetStorageService(strTargetOrganizationID);

this.media = destWebRequestor.GetMediaList(new MediaQuery()

{

OrganizationID = organizationID

}).FirstOrDefault();

string webApiUrl = this.webApiUrl + "/Job/GetMediaList";

获取采集状态：

int nIdcasState = this.GetIdcasState(item.observationRequest);

将检查采集，影像采集，胶片采集的默认状态都置为true。再通过下面的获取采集状态的函数，分析CollectionMark具体三个采集值是什么。

bool bNeedCollectExam = true;

bool bNeedCollectImage = true;

bool bNeedCollectFilm = true;

// 获取采集状态

this.GetCollectState(nIdcasState, ref bNeedCollectExam, ref bNeedCollectImage, ref bNeedCollectFilm);

再一次对这三个状态进行判断，如果为true，就对相应的，检查，影像，胶片进行采集。

比方检查采集，先进行数据校验，获取校验结果。再根据true或者false调用

// 数据校验

bool bIsValid = this.IMCISExamVerify(item, ref model);

但是并没有根据校验结果去判定是否要同步到另一个云，而是无条件把检查注册到另一个云。

var result = destWebRequestor.RegistExamInfo(examInfo);

根据校验返回的真假去编辑数据校验

if (bIsValid)

{

this.ExamDataCheck(model, 10, null, 11);

}

else

{

this.ExamDataCheck(model, -2, null, 11);

}

string webApiUrl = this.webApiUrl + "/Verify/ExamDataCheck";

开始采集文档，依旧是先下载，再上传。

// 采集文档

this.CollectDocuments(model);

// 文档下载

this.DownloadDocuments(docList, model);

// 上传文档

this.UploadDocuments(model);

根据返回结果去修改采集状态值。最后更新到源数据库的ObservationRequest表中该条数据中。

// 修改采集状态

var strStateTemp = model.IdcasState.ToString();

if (bCollectSuccess)

{

strStateTemp = $"{strStateTemp[0]}1{strStateTemp[2]}{strStateTemp[3]}";

}

else

{

strStateTemp = $"{strStateTemp[0]}3{strStateTemp[2]}{strStateTemp[3]}";

}

model.IdcasState = int.Parse(strStateTemp);

this.UpdateCollectState(model);

string webApiUrl = apiurl + "/Job/UpdateAcquisitionStatus";

接下来就是类似的根据是否需要影像和胶片采集进行对应的采集操作：

if (bCollectSuccess)

{

if (bNeedCollectImage)

{

this.CollectImage(model, item.observationRequest);

}

if (bNeedCollectFilm)

{

this.CollectFilm(model, item.observationRequest);

}

}

11.视图的查找：

从用户角度来看，一个视图是从一个特定的角度来查看数据库中的数据。从数据库系统内部来看，一个视图是由SELECT语句组成的查询定义的虚拟表。从数据库系统内部来看，视图是由一张或多张表中的数据组成的，从数据库系统外部来看，视图就如同一张表一样，对表能够进行的一般操作都可以应用于视图，例如查询，插入，修改，删除操作等。

视图是存储在数据库中的查询的SQL 语句，它主要出于两种原因：安全原因， 视图可以隐藏一些数据，如：社会保险基金表，可以用视图只显示姓名，地址，而不显示社会保险号和工资数等，另一原因是可使复杂的查询易于理解和使用。

视图：查看图形或文档的方式。

视图一经定义便存储在数据库中，与其相对应的数据并没有像表那样又在数据库中再存储一份，通过视图看到的数据只是存放在基本表中的数据。对视图的操作与对表的操作一样，可以对其进行查询、修改(有一定的限制)、删除。

当对通过视图看到的数据进行修改时，相应的基本表的数据也要发生变化，同时，若基本表的数据发生变化，则这种变化也可以自动地反映到视图中。

12.

影像采集中的脱敏配置：

查找数据脱敏：

数据脱敏是指对某些敏感信息通过脱敏规则进行数据的变形，实现敏感隐私数据的可靠保护。在涉及客户安全数据或者一些商业性敏感数据的情况下，在不违反系统规则条件下，对真实数据进行改造并提供测试使用，如身份证号、手机号、卡号、客户号等个人信息都需要进行数据脱敏。数据库安全技术之一，数据库安全技术主要包括：数据库漏扫、数据库加密、数据库防火墙、数据脱敏、数据库安全审计系统。数据库安全风险包括：拖库、刷库、撞库。

这样就可以在开发、测试和其它非生产环境以及外包环境中安全地使用脱敏后的真实数据集。

13.

采集状态有了。

/// <summary>

/// 集成平台采集状态

/// </summary>

public enum AcquisitionStatus

{

/// <summary>

/// 未采集（检查记录，影像，胶片）

/// </summary>

AllNotCollected = 1000,

/// <summary>

/// 检查记录已采集（无变更），影像未采集，胶片未采集

/// </summary>

ExamCollectedImageAndFilmNo = 1100,

/// <summary>

/// 检查记录已采集（无变更），影像未采集，胶片已采集（无变更）

/// </summary>

ExamAndFilmCollectedImageNo = 1101,

/// <summary>

/// 检查记录已采集（无变更），影像未采集，胶片已采集（有变更）

/// </summary>

ExamCollectedImageNoAndFilmChange = 1102,

/// <summary>

/// 检查记录已采集（无变更），影像已采集（无变更），胶片未采集

/// </summary>

ExamAndImageCollectedFilmNo = 1110,

/// <summary>

/// 检查记录已采集（无变更），影像已采集（无变更），胶片已采集（无变更）

/// </summary>

AllCollected = 1111,

/// <summary>

/// 检查记录已采集（无变更），影像已采集（无变更），胶片已采集（有变更）

/// </summary>

ExamAndImageCollectedFilmChange = 1112,

/// <summary>

/// 检查记录已采集（无变更），影像已采集（有变更），胶片未采集

/// </summary>

ExamCollectedImageChangeFilmNo = 1120,

/// <summary>

/// 检查记录已采集（无变更），影像已采集（有变更），胶片已采集（无变更）

/// </summary>

ExamAndFilmCollectedImageChange = 1121,

/// <summary>

///检查记录已采集（无变更），影像已采集（有变更），胶片已采集（有变更）

/// </summary>

ExamCollectedImageAndFilmChange = 1122,

/// <summary>

/// 检查记录已采集（有变更），影像未采集，胶片未采集

/// </summary>

ExamChangeImageAndFilmNo = 1200,

/// <summary>

/// 检查记录已采集（有变更），影像未采集，胶片已采集

/// </summary>

ExamChangeFilmCollectedImageNo = 1201,

/// <summary>

/// 检查记录已采集（有变更），影像未采集，胶片已采集（有变更）

/// </summary>

ExamChangeFilmChangeImageNo = 1202,

/// <summary>

/// 检查记录已采集（有变更），影像已采集（无变更），胶片未采集

/// </summary>

ExamChangeImageCollectedFilmNo = 1210,

/// <summary>

/// 检查记录已采集（有变更），影像已采集（无变更），胶片已采集（无变更）

/// </summary>

ExamChangeImageAndFilmCollected = 1211,

/// <summary>

/// 检查记录已采集（有变更），影像已采集（无变更），胶片已采集（有变更）

/// </summary>

ExamAndFilmChangeImageCollected = 1212,

/// <summary>

/// 检查记录已采集（有变更），影像已采集（有变更），胶片未采集

/// </summary>

ExamAndImageChangeFilmNo = 1220,

/// <summary>

/// 检查记录已采集（有变更），影像已采集（有变更），胶片已采集（无变更）

/// </summary>

ExamAndImageChangeFilmCollected = 1221,

/// <summary>

/// 检查记录已采集（有变更），影像已采集（有变更），胶片已采集（有变更）

/// </summary>

Change = 1222

}

3应该是采集失败，接口没有返回成功：

if (bCollectSuccess)

{

strStateTemp = $"{strStateTemp[0]}1{strStateTemp[2]}{strStateTemp[3]}";

}

else

{

strStateTemp = $"{strStateTemp[0]}3{strStateTemp[2]}{strStateTemp[3]}";

}

现在云到云取任务的接口有点问题， 1114状态也返回了， 导致每次都取到相同的100条1114的记录

hk 2019/3/19 17:50:33

我看了下，现在的筛选条件是：where a.DeleteFlag == false && ((a.CollectionMark.Value != 1111 && a.CollectionMark != 1144) || !a.CollectionMark.HasValue)

hk 2019/3/19 17:50:53

那我再加一个条件 a.CollectionMark.Value != 1114?

陈定苗 2019/3/19 17:51:10

应该是 CollectionMark中存在0的就返回

hk 2019/3/19 17:57:16

苗哥 这给0是未采集 1是采集完成 2是变更 4是没有 对不对

hk 2019/3/19 17:57:17

那3呢

陈定苗 2019/3/19 17:57:34

3一般标识失败

hk 2019/3/19 17:58:37

那不应该存在0 就返回 2变更了 或是3 失败了 是不是也要返回 重新采集？

陈定苗 2019/3/19 17:58:53

云到云是没有2的（因为我在想，如果原始数据修改了，那就是时间戳发生变化，那就重新采集，从集成平台到云A，A中的数据被更改，自动将第二位置为0，那云到云的线程也就重新把这条数据同步到云B）

陈定苗 2019/3/19 17:59:04

失败是不需要返回的

hk 2019/3/19 18:00:48

好 那我判断条件 第一位不用考虑 就234位 一旦出现0， 或是整体没有数据 为 null 我就在这些数据里筛选一定数量返回给你？

hk 2019/3/19 18:00:52

是不是这个逻辑

陈定苗 2019/3/19 18:00:57

是的

hk 2019/3/19 18:01:04

好的 那我把接口改掉

我的修改：

var observationRequestSql = from a in \_observationRequestRepository.Table

where a.DeleteFlag == false && (a.CollectionMark.Value % 10 == 0 || (a.CollectionMark.Value / 10) % 10 == 0 || (a.CollectionMark.Value / 100) % 10 == 0 || !a.CollectionMark.HasValue)

select a;

但发现这样的表达式，ef不支持。所以用原先笨的，不等于的方法，因为苗哥说了，没有2，3的可能性，所以最后两位，只有1和4的可能性，所以就四种可能。

var observationRequestSql = from a in \_observationRequestRepository.Table

where a.DeleteFlag == false && ((a.CollectionMark.Value != 1111 && a.CollectionMark != 1144 && a.CollectionMark != 1114 && a.CollectionMark != 1141) || !a.CollectionMark.HasValue)

select a;

问了喻大神，可以用数组，然后contains，再把不要的不等于，就可以了。

这样，如果有新的不需要采集的标识，只要往数组里加就可以了。

var mark = new short[] { 1111, 1144, 1141, 1114 }; // 不需要返回采集列表的采集标识

var observationRequestSql = from a in \_observationRequestRepository.Table

where a.DeleteFlag == false && (!mark.Contains(a.CollectionMark.Value) || !a.CollectionMark.HasValue)

select a;

14.

兵哥和军哥在说重新采集的事情，比方患者有影像错误了，重开了。我们这边就要重新采集。采集的时候是先删除旧的，再把新的都采集上传上来。

15.

云到云的检查列表优化：

原来的：

observationRequestSql = observationRequestSql.Where(p => p.OrganizationID == query.OrganizationID);

优化后（很疑惑，为什么改成Contains比相等优化的多）：

observationRequestSql = observationRequestSql.Where(p => p.OrganizationID.Contains(query.OrganizationID));

自己重新map，发现速率是一样的啊。

又改回原来的了：

// 之前

var list = observationRequestSql.Project().To<PatientObservationRequestDTO>().ToList();

原来的调用（又学到一招）：

public interface IProjectionExpression

{

/// <summary>

/// Projects the source type to the destination type given the mapping configuration

///

/// </summary>

/// <typeparam name="TResult">Destination type to map to</typeparam>

/// <returns>

/// Queryable result, use queryable extension methods to project and execute result

/// </returns>

IQueryable<TResult> To<TResult>();

}

我这个是手动映射：

//var list =

// Mapper.Map<List<ObservationRequest>, List<PatientObservationRequestDTO>>(

// observationRequestSql.ToList());

16.

原来ObservationResult里面的影像所见和影像诊断是多条的，只要检查变更了，采集服务这边就采一次。

那就是同一个分配机构的同一个病人，可以有多条的ObservationRequest的记录，同时，每一条ObservationRequest记录，ObservationUID可以对应ObservationResult中多条记录。涉及到检查，书写，审核，修订。每一条数据的影像所见和影像诊断会不一样。

17.

校验问的群主：

浙江-新手(1543888360) 9:33:33

大佬们 我现在把平台A（合作方）数据库中的数据全部采集到了平台B 现在对方要我提供数据情况 要让他们能看到两边的数据是否一致 有大佬推荐校验的方式吗

北京-长云(1299232669) 9:35:24

先进a看看，再进b看看看看数据有没有变化

浙江-新手(1543888360) 9:36:06

少说几十万条数据啊 总不能一条一条比对吧

广东-神经病患者(603869636) 9:36:21

查最后一条，查记录总数。

重庆-涛帅(2362775355) 9:36:49

一条一条查询

浙江-新手(1543888360) 9:36:52

问题是备份到B上面的 还有C平台过来的数据 所以查总记录不行

重庆-涛帅(2362775355) 9:36:54

写个工具

重庆-涛帅(2362775355) 9:37:15

一个界面 两张表格 一个过滤条

浙江-新手(1543888360) 9:37:40

好吧 看样子只能一条一条对比了 哈哈

重庆-涛帅(2362775355) 9:38:23

根据过滤条 条件 分别查询两张表

重庆-断点(18668853) 9:38:24

@浙江-新手 回馈机制

广州-Debug(1343539055) 9:38:28

你们 从别的平台过来的数据没标识的吗

重庆-涛帅(2362775355) 9:38:38

多查询几次 就行了

重庆-断点(18668853) 9:38:44

针对数据做SHA1哈希计算，对比哈希就可以了

重庆-断点(18668853) 9:39:02

批量查询后针对ID进行哈希

重庆-断点(18668853) 9:39:06

运算速度极高

重庆-涛帅(2362775355) 9:39:12

嗯 对对对

重庆-断点(18668853) 9:39:24

然后定期将你这边的ID和第三方ID的哈希对比，就知道数据是否一致了

广东-省草(2910882149) 9:39:33

群主把我想说的说出来了

重庆-断点(18668853) 9:40:14

你们要约定一个ID

浙江-新手(1543888360) 9:40:15

听起来极其高端 我查查SHA1哈希看 刚跨行从机械转来it 还啥都不会 哈哈

浙江-新手(1543888360) 9:40:18

谢谢群主

重庆-断点(18668853) 9:40:51

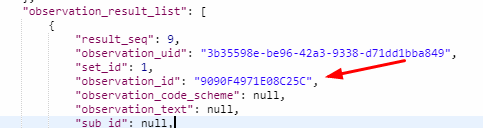
不用谢，你是我群员，我该帮你 。

浙江-新手(1543888360) 9:41:01

嗯嗯

18.

如果把检查结果的加密放在AutoMapStartup中，会造成同用这个实体的云到云检查列表中的申请结果也遭到加密。



所以最终把开辟映射放在控制器的方法中。

// 数据处理

if (ret.data != null && ret.data.Count > 0)

{

result.data = Mapper.Map<List<CloudExamListDTO>, List<CloudExamListModel>>(ret.data);

// 患者检查结果信息DES加密

Mapper.CreateMap<PatientObservationResultModel, PatientObservationResultModel>()

.ForMember(dest => dest.Set\_ID, opt => opt.MapFrom(p => Cryptology.Encrypt(p.Set\_ID))) // DES加密（short报错）

.ForMember(dest => dest.Observation\_ID, opt => opt.MapFrom(p => Cryptology.Encrypt(p.Observation\_ID))); // DES加密

Mapper.CreateMap<CloudExamListModel, CloudExamListModel>();

//数据的DES加密

result.data = Mapper.Map<List<CloudExamListModel>, List<CloudExamListModel>>(result.data);

}

19.

看了苗哥的修改，用的也是军哥这种，[JsonProperty(PropertyName = "job\_uid")]

namespace eWorld.Job.Models.DTO

{

/// <summary>

/// Job触发器信息

/// </summary>

[JsonObject(MemberSerialization.OptIn)]

public class JobTriggerInfo : RequestHandler

{

/// <summary>

/// 任务触发UID

/// </summary>

[JsonProperty(PropertyName = "trigger\_uid")]

public Guid? TriggerUID { get; set; }

/// <summary>

/// 任务名称

/// </summary>

[JsonProperty(PropertyName = "trigger\_name")]

public string TriggerName { get; set; }

/// <summary>

/// 任务定义UID

/// </summary>

[JsonProperty(PropertyName = "job\_uid")]

public Guid? JobUID { get; set; }

/// <summary>

/// 任务组定义UID

/// </summary>

[JsonProperty(PropertyName = "group\_uid")]

public Guid? GroupUID { get; set; }



20.

云到云，在最后的下载中，先是从DocumentService根据TypeCode区分下载申请文件，图文报告，影像，胶片，然后再上传到云B，把申请单，报告单，影像（包括心电波形图，病理切片），胶片统统上传。

/// <summary>

/// 启用云端采集影像

/// </summary>

public bool EnableImageCloudImage { get; set; }

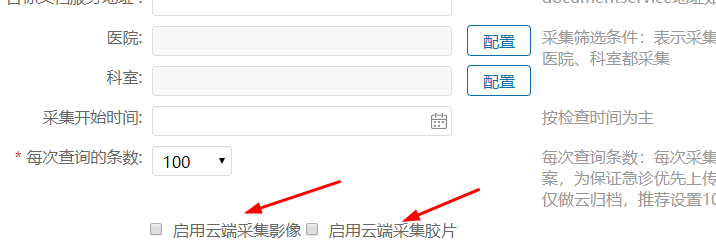
/// <summary>

/// 启用云端采集胶片

/// </summary>

public bool EnableImageCloudFilm { get; set; }

对应采集策略中的：



苗哥表示这里默认是true的，这两个参数，所以在云到云中虽然会走进下面的两个方法，但是内部永远是false，是不执行的。这里暂时用不到，但有种情况就是以后下载走云端，而不是走集成平台，是云端到云端的下载。而不是到原始数据中去下载到B云。而是从A云中下载下来直接上传到B云。（大概是这样）

if (bCollectSuccess)

{

if (bNeedCollectImage)

{

logger.Info(LogType.CollectLog, $"{JobDesc}需要采集影像, ObservationUID: {item.observationRequest.ObservationUID}");

this.CollectImage(model, item.observationRequest);

}

if (bNeedCollectFilm)

{

logger.Info(LogType.CollectLog, $"{JobDesc}需要采集胶片, ObservationUID: {item.observationRequest.ObservationUID}");

this.CollectFilm(model, item.observationRequest);

}

}

方法内部：

这里默认是false，所以不进入函数体。

if (!this.collectImageFromCloud)

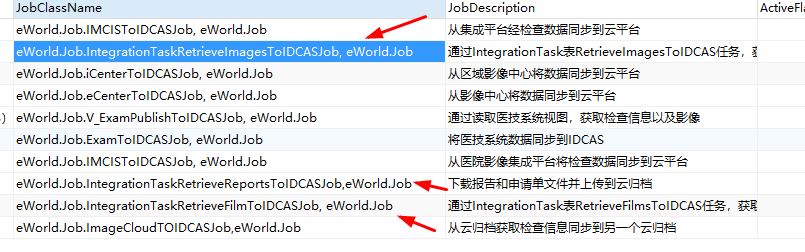
{

// 如果不从云归档采集影像，那就从存档采集

logger.Info(LogType.CollectLog, $"{JobDesc}开始产生获取影像任务，检查唯一号：{model.ObservationUID}");

苗哥对于目前单线程变成多线程的处理，就是在DocumentService中下载申请，报告，影像，胶片这里，如果是下载报告，就生成一个上传报告的集成任务，如果是影像，就生成一个影像的集成任务，胶片就是胶片的集成任务。

就是打箭头的三个：



当这里生成了集成任务，那后面的影像和胶片的集成任务就可以去掉，先注释掉。

这里原本存在的意义是因为有一种情况，后面可能云到云的申请单，报告，从云上采，但影像从存档采（应该还有胶片，也就是不走云端，而是走Dicom的方式）。所以就要生成这样的任务。

因为影像如果不从集成平台采的话，那它的下载方式就不一样了。一种是Dicom方式下载，我们现在的RetrieveImages方式都是从Dicom方式进行下载的。

如果云到云独立一个下载方式，就不是Dicom方式，它是从云端直接下载。就下载方式不一样。但任务名要保持一致，仍旧是生成RetrieveImages这个任务，里面可以加一个字段（参数）来进行区分，它是云到云这种下载方式。

如果是从云上面下载，就在上面的下载文档中生成一个从云下载的任务，而如果是从Dicom中下载，就走下面的生成任务的方式，走Dicom下载方式。

一种就是DocumentService给路径的方式直接下载，还有一种方式是走Dicom下载的方式，走Dicom的访问地址。



21.

云到云的校验

新建一个云到云校验的服务，加载到job列表中，当启用了云到云，那serviceuid默认对应到当前的云到云校验服务，启用。加一个云到云校验的标记，比方1111，仍旧用之前的标记位，完成的状态置为2，后台服务一旦检索到这样的集成任务，开始调用源云的该business\_id对应的observation\_request的数据，再以同样的business\_id获取目标云的该id对应的observation\_request的数据，在服务中对是否存在影像，胶片进行比对，如果匹配，那该集成任务顺理完成，可以删除，否则，将该business\_id对应的集成任务置零，从新开始采集。（或者，更直接的，将源云上该条数据的CollectMark置为1000，那就会继续获取到该条任务，从新开始）（这里会存储原始的CollectMark的值吗，如果失败，直接给恢复成原来的值）

这里应该是只要比对胶片数和是否有影像就可以了。

/// <summary>

/// 胶片数

/// </summary>

public short? FilmCount { get; set; }

/// <summary>

/// 是否需要胶片，0不需要，1需要

/// </summary>

public bool? FilmNeed { get; set; }

/// <summary>

/// 是否有影像

/// </summary>

public bool? HasImage { get; set; }

苗哥表示现在的问题是，集成任务成功之后，是要删除的。以后要这么搞，校验成功的任务才可以删除。

在生成这四条任务（报告，影像下载，影像上传，胶片）（影像下载和影像上传其实同一个）之前，任务表中启用一个字段来表示云到云任务的状态。循环遍历云到云检查结果集中，先会生成一条校验的任务，后才生成报告，影像，胶片的任务，针对的都是当前检查记录的observation\_uid，对应报告这边的business\_id。当报告的任务执行成功，第二位置为1，影像上传成功，第三位置为1，胶片上传成功，第四位置为1。这个时候校验任务就可以启动了，校验任务完成了，就把TaskStatus改为2。一开始都是0，未处理。

三位就可以了，初始为”000”的字符串，第一位是报告，第二位是影像，第三位是胶片。

当为”111”， TaskStatus=0时，校验任务就可以启动了。

校验服务一直检索，比方隔一段时间调用一个获取集成任务的接口，条件就是TaskStatus=0，另外一个字段=”111”，才获取到这条记录，开始执行——调用两个云的接口获取observation的记录，然后本地进行比对。

只有这个字段的234位都是1了，并且TaskStatus为0，这条任务才能被检索到，才能进行校验处理。处理完成，校验成功了，再把这条任务的TaskStatus置为2，表示完成。如果校验失败了，那么这个字段TaskStatus就是3。

校验的准测：

1.检查记录要匹配

2.报告数量相同

3.影像数量相同

4.胶片数量相同

采集服务去两边获取过来，进行比对。

以生成校验任务的方式来处理。

云到云采集校验方案：

1）云到云检查列表获取后，循环遍历将检查信息注册到目标云，再根据当前observation\_uid获取云到云检查文档列表后，在文档下载中将云到云检查文档列表中的报告，影像，胶片的下载拆分成三个集成任务，并在这三个任务之前，先生成一条云到云信息校验的任务，其中TaskStatus=0，InteractionId=”000”。同时，报告，影像，胶片的三个集成任务，InteractionId=”0”，用于区分是原来集成平台到云的Retrieve，还是当前云到云的Retrieve任务。

2）RetrieveReports，RetrieveImages，RetrieveFilm一直执行，处理报告，影像，胶片的下载和上传，当报告上传完毕，如果InteractionId=”0”，证明是云到云的Retrieve任务，则将校验任务中InteractionId的第一位的值置为1，也就是InteractionId=”1xx”，当影像上传完毕，同样InteractionId=”0”，将第二位的值置为1，也就是InteractionId=”x1x”，当胶片上传完毕，InteractionId=”0”，将第三位的值置为1，也就是InteractionId=”xx1”。

3）采集服务新增一个云到云信息校验服务，检索集成任务中的TaskStatus=0，InteractionId=”111”的记录，一旦有这样的记录，获取该BusinessId作为入参，调用源云的校验接口，以及目标云的校验接口，获取两条检查记录进行比对，如果检查记录匹配，报告，影像，胶片的数量相同，则视为校验成功，此时将该条校验集成任务的TaskStatus=2，也就表示校验成功，否则TaskStatus=3，表示校验失败。

4）TaskStatus=2的集成任务，会被定期删除。

22.

确实可能存在报告，影像，或者胶片缺失，那么我这里的校验标记位就可能触发不了，因为始终无法变为111，一种是通过CollectMark，计算出是否有报告，胶片，影像，但是1111，四位，前面是检查，和报告又有区别啊。后两位是影像和胶片。通过这里确定，感觉不对。

还有一种，苗哥是通过当前的observation\_uid检索所有的云到云文档列表，通过循环遍历，立马的TypeCode确定是生成哪种任务，在循环遍历之后，通过查询任务可以知道是否有缺失，如果报告缺失，第一位的置置为”4xx”，如果是影像，则第二位”x4x”，胶片的就是”xx4”，这样最终的校验触发条件就是InteractionId的值转化为int后，大于等于111。这样就触发任务。

23.

云到云采集校验方案1：

1）云到云检查列表获取需要同步的检查列表，循环遍历检查列表，将其注册到目标云，当注册结束，直接通过observation\_uid获取到目标云上的该条检查记录，与当前的检查记录进行对比，如果一致，才进行第二步。否则直接将源云上该条数据的collect\_mark置为”x0xx”，这样就会执行重新采集。

1）云到云检查列表获取后，循环遍历将检查信息注册到目标云，再根据当前observation\_uid获取云到云检查文档列表后，在文档下载中将云到云检查文档列表中的报告，影像，胶片的下载拆分成三个集成任务，并在这三个任务之前，先生成一条云到云信息校验的任务，其中TaskStatus=0，InteractionId=”000”。同时，报告，影像，胶片的三个集成任务，InteractionId=”0”，用于区分是原来集成平台到云的Retrieve，还是当前云到云的Retrieve任务。

2）当循环遍历云到云文档列表结束，查询当前business\_id对应的RetrieveReports，RetrieveImages，或是RetrieveFilm任务，如果不存在，将对应的校验任务的标记位InteractionId打为”4xx”，”x4x”，或是”xx4”，表示该任务不存在。

3）RetrieveReports，RetrieveImages，RetrieveFilm一直执行，处理报告，影像，胶片的下载和上传，当报告上传完毕，如果InteractionId=”0”，证明是云到云的Retrieve任务，则将校验任务中InteractionId的第一位的值置为1，也就是InteractionId=”1xx”，当影像上传完毕，同样InteractionId=”0”，将第二位的值置为1，也就是InteractionId=”x1x”，当胶片上传完毕，InteractionId=”0”，将第三位的值置为1，也就是InteractionId=”xx1”。

4）采集服务新增一个云到云信息校验服务，检索集成任务中的TaskStatus=0，InteractionId的int值>=111的记录，一旦有这样的记录，获取该business\_Id作为入参，调用源云的云到云检查接口，以及目标云的云到云检查接口，获取两条检查记录，只要都存在，重要信息比对一致，就说明检查记录正常。如果不一致，continue，并将源云该条记录的collect\_mark的第二位置为0，即x0xx，这样会通过云到云获取检查列表获取到，重新进行云到云的采集。并且，后续的报告，影像，胶片的集成任务也就不必生成。

如果检查记录是一致的，接着同样以business\_Id作为入参，获取源云和目标云的检查文档列表，通过type\_code可以确定类型是报告，影像，还是胶片。通过比对数量可以知道是否数据一致。如果不一致，将对应的集成任务的task\_status置为0，重新采集，并将校验任务的InteractionId对应位置为0，同时将task\_status置为0，继续丢入等待校验的池子中。

如果完全校验成功，此时将该条校验集成任务的TaskStatus=2，也就表示校验成功，否则TaskStatus=3，表示校验失败。

5）如果需要对校对信息进行统计的话，和校对相关的集成任务就不能删除，或者要放到一张单独的云到云任务校验表中，这样才能进行统计。

6）TaskStatus=2的集成任务，会被定期删除。

这里有个问题，苗哥是通过云到云检查文档列表确定是否需要进行文档下载和上传，但是通过collect\_mark是可以知道有没有影像，胶片的。第二位是检查和报告。

感觉应该是通过collect\_mark的三位就可以确定是否需要生成报告，影像，胶片的任务。

另外，通过云到云的检查文档列表，同样可以知道是否需要报告，影像，胶片的任务。

24.

两个礼拜迭代一个版本，今天云归档任务重，要部署的地方很多。

1.0.7维护，1.0.6的api接口也要维护，给第三方调用，比方我写的患者信息注册那些。

25.

一种是可以校对了的标记位，目前是InteractionId，另外可以加一个标记位，表示检查（苗哥是没提到这个，就后面三个），报告，影像，胶片，具体的校验情况。此时TaskStatus=3。成功了的话，TaskStatus=2，这是肯定的。

但是通过TaskStatus=3去查询，通过nteractionIdExt1这个第二标记位，如果校验成功，则对应位为1，如果校验失败，则为3。比方”113”，就知道失败的原因是因为胶片校对失败。

这样就可以通过查表知晓校验的具体情况了。

比方如果界面上显示了胶片校验失败，有个按钮，一点，重新采集。那就是business\_id对应的RetrieveFilm集成任务，TaskStatus置为0，重新开始采集。而不需要所有的东西都重新采集一遍。

校验没有问题，很快的，哪怕是只是胶片重新采集了，把报告，影像都叫重新校验一遍，也快的。

没有的情况，我提出的，比方报告，影像，胶片没有，缺失，那置为4。苗哥说没有的情况有两种，一种是现在没有，以后会有，一种是以后也没有。

兵哥表示只要采集了，就校验，而不管集成任务里面报告，影像，胶片到底有没有上传完成，只要有变更（collect\_mark有变化），就触发它的重新校验。

报告，影像，胶片任务完成都触发校验任务，那就是一条数据最多校验3次。

云到云的采集，只要数据有变更，就一定要触发校验任务重新进行校验。

就像统计的字段，就是汇总标记—IsSummary这个东西，用来做check（校验）表和检查表的校验。用来保证数据完整性的。有变更就全部置为0。

校验到底要不要走任务。集成任务这张表。这是两种方案。

UploadFlag目前是用于在线监控展示。就是采集质量那里。

5，没有影像，10，上传成功，都要校验。

A云同步到B云，在目标云，也就是B云上，通过这张表去校验。

从目标云上，去examcheckdata表中获取数据，去和源云的数据进行比对。校验的标记打到目标这里。

报告可以用什么ifeich，苗哥说哈希值比对，但影像不可以，因为涉及到压缩这些。

数量也可以走DicomDir查看，可以调赵丹那边DocumentService里面获取数量。

后期补上来，数量会不一样，因为同样的数据过来，但检查号不一样了，源云第一条已经删除了，现在只有一条，但目标云有两条。

楼哥表示，记录是不删除的，删除的只是文档，所以源头的记录也是两条，在目标云通过标记去校验，之前源云上已经校验的数据是不用管的。

只要校验那些新追加的，或者已经变更的源云的数据。所以源云变更的同时，肯定CollectMark变掉，造成云到云重新采集，同时触发check表中的Upload标记位变化，只要就触发了重新校验。又是拿目标云和源云的同一条数据进行比对。

一对一，标记可以打在源云上，但一对多，肯定要把标记打在目标云上。

如果原始数据变化，有变更，必然涉及到源云的CollectMark发生变化，触发重新采集，一旦触发重新采集，势必导致目标云中Check表中UploadFlag值的变化，这样，就触发了校验任务。

以目标为标准，需要以检查时间为准来校验，以源为标准，需要以记录为标准。

我这里，需要增加一个校验时间的标记位么？

我还要开一个接口，获取检查校验信息的接口。

之后，1.0.6上面，云到云也要改。工作量很大啊。也就是我的工作。

26.

兵哥表示有一个服务是处理检查表和ExamCheck表的数据一致的。

军哥这里加了字段，一个检查更新时间，我这边在处理检查，报告，影像，胶片上传标记的时候，要把服务器时间写入。

27.

我加了日志后，更新到服务器，发现日志里面写入了：

[时间]2019-03-23 13:36:29,255[内容]:患者信息注册成功，主索引为：25dfcee7-6edf-45e7-b7d4-58cbe076893b,传入参数：

{"PatientID":"66666666","PIDAssigningAuthority":"74580626.ECG","PatientMasterID":"25dfcee7-6edf-45e7-b7d4-58cbe076893b","Name":"张秀英","NameSpell":null,"MotherName":null,"Sex":"女","BirthDate":"1962-11-14T00:00:00","BirthPlace":null,"Nation":null,"Citizenship":null,"MaritalStatus":null,"IDCardNO":"332623196211141448","HealthCardNO":null,"ContactPhoneNO":"13175391911","HomePhoneNO":null,"BusinessPhoneNO":null,"Email":null,"AddressProvince":null,"AddressCity":null,"AddressDistrict":null,"AddressStreet":null,"AddressRoad":null,"AddressDetail":"","Postalcode":null,"Occupation":null,"WorkUnit":null,"Language":null,"InsuranceType":null,"InsuranceID":null,"CreateOrgnizationID":"74580626","InHospitalFlag":null,"DeleteFlag":null}[异常]:

而且，logclinicdataoperation表也成功写入了信息。

28.

云到云采集校验方案1（走集成任务）：

1）云到云检查列表获取需要同步的检查列表，循环遍历检查列表，将其注册到目标云，当注册结束，直接通过observation\_uid获取到目标云上的该条检查记录，与当前的检查记录进行对比，比对信息如果一致，才进行第二步。否则continue跳过此次云到云采集任务，并将源云该条记录的collect\_mark第二位的值置为0，即x0xx，这样就会通过云到云检查列表重新获取到，重新进行云到云的采集。并且，后续的报告，影像，胶片的集成任务也就不必生成。

2）循环遍历中，如果检查信息校验成功，则根据当前observation\_uid获取云到云检查文档列表后，在文档下载中将云到云检查文档列表中的报告，影像，胶片的下载拆分成三个集成任务，并在这三个任务之前，先生成一条云到云信息校验的任务，其中TaskStatus=0，InteractionId=”000”。同时，报告，影像，胶片的三个集成任务，InteractionId=”0”，用于区分是原来集成平台到云的Retrieve，还是当前云到云的Retrieve任务。

3）当循环遍历云到云文档列表结束，查询当前business\_id对应的RetrieveReports，RetrieveImages，或是RetrieveFilm任务，如果不存在，将对应的校验任务的标记位InteractionId打为”4xx”，”x4x”，或是”xx4”，表示该任务不存在。

4）RetrieveReports，RetrieveImages，RetrieveFilm一直执行，处理报告，影像，胶片的下载和上传，当报告上传完毕，如果InteractionId=”0”，证明是云到云的Retrieve任务，则将校验任务中InteractionId的第一位的值置为1，也就是InteractionId=”1xx”，当影像上传完毕，同样InteractionId=”0”，将第二位的值置为1，也就是InteractionId=”x1x”，当胶片上传完毕，InteractionId=”0”，将第三位的值置为1，也就是InteractionId=”xx1”。

5）采集服务新增一个云到云信息校验服务，检索集成任务中的TaskStatus=0，InteractionId的int值>=111的记录，一旦有这样的记录，获取该business\_Id作为入参，获取源云和目标云的检查文档列表，通过type\_code可以确定类型是报告，影像，还是胶片。通过比对数量可以知道是否数据一致。如果不一致，将对应的集成任务的task\_status置为0，重新采集，并将校验任务的InteractionId对应位置为0，同时将task\_status置为0，继续丢入等待校验的池子中。

如果完全校验成功，此时将该条校验集成任务的TaskStatus=2，也就表示校验成功，否则TaskStatus=3，表示校验失败。并且增加一个字段，比方原有的InteractionIdExt1字段用于标记位，同样用1xx，x1x，xx1来标记报告，影像，胶片的校验结果。如果校验成功，则为1，如果失败，则为3。比方”113”，就知道失败的原因是因为胶片校对失败。这样的话，如果触发重新采集，也只需要将business\_id对应的RetrieveFilm集成任务，TaskStatus置为0，重新开始采集，而不需要所有的东西都重新采集一遍。

6）如果需要对校对信息进行统计的话，和校对相关的集成任务就不能删除，或者要放到一张单独的云到云任务校验表中，这样才能进行统计。

云到云采集校验方案2（不走集成任务，标记位打在目标云，满足一云采集到多云的需求）：

1）云到云检查列表获取需要同步的检查列表，循环遍历检查列表，将其注册到目标云，并将目标云examcheckdata表中当前检查唯一号ExamIndex对应的RegistFlag置为10，表示上传成功，获取云到云检查文档列表后，在文档下载中将云到云检查文档列表中的报告，影像，胶片的下载拆分成三个集成任务，一旦上传成功，目标云中对应的标记位UploadReportFlag（上传报告标识），UploadImageFlag（上传影像标识），UploadFilmFlag（上传胶片标识）置为10，如果没有，则为5。

2）开一个云到云获取检查校验信息的接口，把examcheckdata表的数据返回。后台启用一个校验服务，一直去目标云中获取RegistFlag为10，UploadReportFlag，UploadImageFlag，UploadFilmFlag等于5或是10的数据，获取到对应的检查唯一号ExamIndex，通过云到云获取检查列表和文档列表两个接口，分别调取源云和目标云的数据，比对检查信息，重要信息一致，将目标云的RegistFlag置为20，表示校验成功。如果失败，RegistFlag置为30，并将源云ObservationRequest表中的CollectMark置为x0xx，触发云到云检查信息的重新采集。而报告，影像，胶片的校验，比对的是云到云检查文档结果集中，源云，目标云中报告，影像，胶片的数量是否一致。如果一致，将校验结果置为20，如果不一致，将UploadFlag置为30，同时将源云该条数据的CollectMark置为x0xx，xx0x，xxx0，从而触发云到云检查信息，影像信息，胶片信息的重新采集。

而一旦触发新的采集任务，无论成功还是失败，势必造成目标云examcheckdata表该检查号对应数据的UploadFlag变更，一旦变更，不再是20或者30，变为5或者10，即再次触发校验服务的重新校验。

如果需要对校验进行统计，还需要在examcheckdata表中增加一个校验时间的字段。

5，没有影像，10，上传成功，都要校验。

A云同步到B云，在目标云，也就是B云上，通过这张表去校验。

从目标云上，去examcheckdata表中获取数据，去和源云的数据进行比对。校验的标记打到目标这里。

报告可以用什么ifeich，苗哥说哈希值比对，但影像不可以，因为涉及到压缩这些。

数量也可以走DicomDir查看，可以调赵丹那边DocumentService里面获取数量。

后期补上来，数量会不一样，因为同样的数据过来，但检查号不一样了，源云第一条已经删除了，现在只有一条，但目标云有两条。

楼哥表示，记录是不删除的，删除的只是文档，所以源头的记录也是两条，在目标云通过标记去校验，之前源云上已经校验的数据是不用管的。

只要校验那些新追加的，或者已经变更的源云的数据。所以源云变更的同时，肯定CollectMark变掉，造成云到云重新采集，同时触发check表中的Upload标记位变化，只要就触发了重新校验。又是拿目标云和源云的同一条数据进行比对。

一对一，标记可以打在源云上，但一对多，肯定要把标记打在目标云上。

如果原始数据变化，有变更，必然涉及到源云的CollectMark发生变化，触发重新采集，一旦触发重新采集，势必导致目标云中Check表中UploadFlag值的变化，这样，就触发了校验任务。

以目标为标准，需要以检查时间为准来校验，以源为标准，需要以记录为标准。

我这里，需要增加一个校验时间的标记位么？

我还要开一个接口，获取检查校验信息的接口。

之后，1.0.6上面，云到云也要改。工作量很大啊。也就是我的工作。

29.

数量要一致，状态也要一致（CollectionMark）。

校验失败的话，要进行短信通知。

之前集成平台的校验方式：

检查记录的校验，起了两个任务，第一个任务是统计检查量，就是每一条采上来时间，平台肯定有记录，然后这个任务，比方每天凌晨两点去执行，然后查前一天的检查时间，所有的检查数，包括平台的发布数，然后两边做比较。

对于每一条记录的校验，是在校验表里面。影像有没有上传，报告有没有上传，胶片有没有上传，源影像张数，目标影像张数。

阿刚安排：

我就根据检查表去核对数量，如果数量不对，要求告诉对方哪条记录是没有的，没有的，就要求对方插入到我们的列表中来。如果哪条记录变更了，让他打个变更，如果哪条记录删除了，让他打个删除的标记。

对于这样的记录，我才插到校验表中去。

校验表的目的只是为了我一条一条地去（核对它的数量是否一致，）核对它记录的内容是否一致，报告的内容，影像的内容，胶片的内容。

先是核对数量，只是把不对的，插到校验表任务中去。来处理。促使重采也好，促使比对也好，关键是先确定数量，重点是数量，如果数量对了，再按照记录去核对。

如果它是很早以前的，它发生变更了，也促使我们重采，重采去生成校验表中的某一条记录，

兵哥方案缺失的问题，目标平台是有问题的，比方我要去采审核完成的，但源平台上有书写完成但没有审核完成的，这个确实会缺失的。

它审核完了会重新读上来。

数量的话，源云上比方有2000个，但我采上来的，只有1900个，就知道缺失多少个了。

一旦审核完成，会触发统计校验任务，重新去校验校验表中的数据。

逻辑：首先是检查，它可能存在书写啊，审核啊，多种状态，目标平台这里，它有个采集服务，它有一个统计的任务，按检查时间去统计源平台所有的检查数，然后写到统计校验表中去。

然后目标平台有采集过来的检查数，然后它会再写一个发布的检查数，还有一个源平台可发布的，这三个数量，可发布的和已发布的去校验，如果这两个相等，就证明今天的采集是没有问题的。可能就是当天可发布的和当天所有的检查数会存在误差，可能没有书写，没有审核什么的，等它审核完成之后会自动地过来，采集上来，然后触发统计任务重新统计这一天的。

这个是检查数的校验。

然后校验表里每一条的记录，拿到每条记录，去校验报告影像，源平台影像多少张，目标平台影像多少张，这些都会去校验。

可能存在问题，目标平台我不采胶片，那胶片就是打了个标记4，代表我不去采了，也不用去校验这个胶片，是这样处理的。

这里是两张表，一个是统计校验表，一个是校验表。校验表是详情。

思路是对的，校验数量和实际的校验，这两个是要分开的。

兵哥目前的校验，目前是校验已发布的，也就是采集到了目标平台的检查数，再去比较源平台上满足该发布条件的，也就是可发布数，如果这两个相等的话，就表明当天的数量是一致的。

这样存在的问题，比方校验查的是2000条，其中可发布的是1900条，但剩下来的100条，具体什么情况，是不知道的。

准备做的，不管它是书写完的还是审核完的，全部采集过来，只不过不让它访问，那这样能保证两个平台的数量是一致的（让我想到集成平台到云的采集，一共有58条数据，但实际上来只会有54条，因为其他四条，比方状态是不满足的，所以进不了sql语句，这时候可发布的，确实只有54条，我采集上来也是54条，那就是校验数量是成功的），一模一样。这里就不存在可发布，已发布，或者说总的检查数了，其实就是已发布的，和总的，两个平台这样去校对。那少的这个，就会写进详情里面，是没有采上来的。

整改方案：比方我现在是取已发布的，1900条，查的某个时间范围，就是今天零点之前的四十八个小时，这段时间。

现在把这个列表查出来给对方，让对方一条条给我标，有这条，修改了，或者delete了，或者没有需要insert，再把表返还给我，我拿到后，如果是修改了，那就知道要重采了，加到重采表中去，生成一条新的任务，如果是delete，那我这边就delete掉，如果是add，那我就增加，重新采。

这样就保证两边数据的数量是一致的，然后在校对的内容，校对的线程中，再一条条去对，这样就可以了。

如果是这样的话，就要求对方提供给我们这样的接口，如果是集成平台，那就是集成平台提供给我们这样的接口，如果是云到云，那那边的云也要提供给我们一个这样的接口。如果你是视图，那我就要通过视图来处理了，我们采集工具这里要做处理。

先是从目标，比方查今天的凌晨往前推48个小时，调用这个时间范围，你把这个列表，把这个数据集给我，然后我用这个数据集调源平台的接口，源平台把数据集处理好（校验好数量）后返还给我，我就知道有没有修改，有没有删除，有没有新增的，然后我再做相印的处理。到我的任务表，到我的校验表里去。小明和詹智在客户端做这部分操作就可以了，根据拿到的结果集做相应处理。

我给的数据，他for循环我的结果集，去他的数据库中查，把结果集又返还给我，我再根据返回的结果集做处理，是更改标记位重采，删除该条记录，还是啥的。这样也就不需要打标记位了。

如果是云到云，或者集成平台到云，可以直接修改标记位触发重采，那其他的，比方视图，就不行了。

如果是第三方，不走的采集服务的被动抓取，那就是根据我的结果集，自己去比对，然后再通过我的主动注册的接口（含有增删改功能）把记录给主动注册上来。

如果是被动抓取，那就是采集服务来干这个活儿。

这样的话，两边肯定对的牢，对不牢的话，错误日志会告诉我们原因。因为一校对，错误日志都有了，可以生成了，哪条记录我是修改了，哪条记录我是没有的，那兵哥那边一看原因就知道了。

最终提供一个校验结果，提供一个质量报告。

X到云的校验方案：

1.源云和目标云分别提供一个获取时间段（比方当天凌晨往前推24小时）检查结果集的接口。

提供获取文档列表的接口。

2.

1）如果是主动注册，第三方获取目标云该时间段的检查结果集，与自己的数据进行对比，如果是缺失，则调用目标云的注册接口，如果是已修改，则调用更新接口，如果是已删除，直接调用删除接口。

当数据一致后，获取源云的获取文档列表接口，再比对报告，影像，胶片的具体数量。

2）如果是被动抓取，就是采集工具分别调用源云和目标云获取检查结果集，如果缺失，则重新进行该记录的注册，如果是发生了更新，也是重新注册，如果是删除呢，也是注册？

当确保两边的数量一致时，才启动校验线程针对每一条记录去校验，如果是

校验文档的时候，比方没有选中采集胶片，那目标云上CollectionMark对应的位置置为4，相应的，该部分也就不需要进行校验。

校验失败的话，要进行短信通知。

30.

/// <summary>

/// 短链接类型 1 影像浏览短链接 2 检查报告短链接 3 患者短链接（默认2）

/// </summary>

public Nullable<short> Kind { get; set; }

/// <summary>

/// 访问模式 0 无限制 1 验证码访问 2 有效期访问(默认2)

/// </summary>

public short? AccessMode { get; set; }

这里用的是AccessMode=2的情况，也就是有效期访问，而我的二维码，应该无限制的，AccessMode为0。

if (code == null)

{

code = new Random().Next(100000, 1000000).ToString();

var query = new AddShortUrlQuery

{

ObservationUID = examDetailParams.ObservationUID,

AccessMode = 2,

CustomParamContent = code,

ExpireTime = DateTime.Now.AddMinutes(ConfigDefine.DigitalImageOutDate)

};

var result = \_imageArchiveClient.Invoke(i => i.GetShortUrlAccess(query));

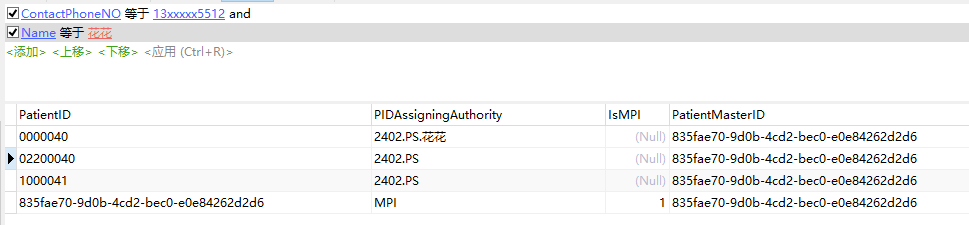
31.

整理主动注册接口：

if (\_patientIndexRepository.Table.Any(a => a.PatientID == query.PatientID && a.PIDAssigningAuthority == query.PIDAssigningAuthority))

{

主干的，在匹配主索引后，只通过病人号和分配机构去查询是否拥有该记录，我分支加了主索引的，但其实，如果匹配上，主索引肯定是唯一的，对的上的，也就可以不加该主索引，问题不大。



而其实，入参当中，和visit表相关的数据，在映射到PatientIndex表后，都是丢失的。

所以我这边其实可以控制不传的，因为传了也是白传。



观察申请表的逻辑，传了ObservationUID就是更新，否则一律是insert，和我1.0.6写的不一样。1.6.6我是根据所有必传入参进行查询，如果存在了，就update，否则就都是insert。

另外，患者注册，观察申请注册，观察申请结果注册的参数验证，不是我1.0.6这种很严格的，因为同一个方法，采集服务的检查注册也是公用这个方法，如果严格了入参，那采集服务从其他平台采集过来的参数也会有约束了。

32.

张小明 2019/4/12 14:30:20

邵逸夫医院离线采集工具

环境：本机电脑上搭建FTP，目录指向本机硬盘的某个文件夹

测试结果：

1、扫描3.5w目录（单纯扫描ftp目录，无下载上传等其他操作），耗时1小时左右；

2、测试185份影像文件，共8.5G数据，从扫描至全部成功上传完成耗时约26分钟；

3、经过Modality映射后部分原本匹配不上的数据匹配上了

测试部-楼静斐 2019/4/12 14:31:57

这样看起来 好像速度没有明显的提升。

张小明 2019/4/12 14:34:14

主要时间是在上传方面，一点没提高么

张小明 2019/4/12 14:35:17

我上传任务时改成多任务异步同时上传了，我之前测试比单任务同步上传要快点。但是上传还是会受限于网络

研发部-简刚 2019/4/12 14:36:05

不用一次性扫描那么多目录吧

张小明 2019/4/12 14:37:05

扫描规则还没改，还是按照之前的，先扫面完成。全部扫描完成之后开启下载、上传、等其他线程

张小明 2019/4/12 14:41:02

之前的性能我也不知道怎么样，按照现在的这个上传速度，大概是8.5\*1024/26/60=5.6兆每秒。

测试部-楼静斐 2019/4/12 14:41:43

之前的速度大概是5M/S

张小明 2019/4/12 14:42:35

网络上限因素呢

测试部-楼静斐 2019/4/12 14:44:13

现在通过FTP方式 是不是没有下载和打包过程了？

瓶颈在上传这边?

张小明 2019/4/12 14:44:30

恩。。。大小错了，总共上传了9.69G，那速度大概在6.4M/s

张小明 2019/4/12 14:44:45

下载也有的，打包没了

研发部-简刚 2019/4/12 14:51:58

扫描的过程要改进

研发部-简刚 2019/4/12 14:52:03

完全是没有必要的1小时

张小明 2019/4/12 14:52:07

多了个从ftp上下载目录中的第一个影像文件的步奏，少了个压缩的步奏

研发部-简刚 2019/4/12 14:52:26

实际读取的时候，还是要扫描的。

研发部-简刚 2019/4/12 14:52:46

FTP的链接是否是长链接？

张小明 2019/4/12 14:54:47

用的FtpWebRequest这个类，是否长链接还真没看过

张小明 2019/4/12 14:55:56

扫描是为了一次性读取所有目录放入数据库的。后面实际读取时从数据库拿目录再去下载的

张小明 2019/4/12 15:00:12

我是先回来吗@研发部-简刚

研发部-简刚 2019/4/12 15:00:52

可以

研发部-简刚 2019/4/12 15:00:58

调试通了就先回吧

研发部-简刚 2019/4/12 15:01:01

再改进

研发部-简刚 2019/4/12 15:01:22

你回来我跟你看看代码，确认逻辑是否要调整。

33.

策略是在源平台，但是生成的任务，还有校验任务都是放在目标平台。

对啊，如果校验任务是生成在目标云，那策略就要配置在目标云上面。

更新起来，源云和目标云都得更新？

34.

采集服务（由詹智主要负责开发维护，曹兵，黄铿，张小明辅助）：

1.服务配置（能力开放平台接口地址，临时文件路径，日志文件路径及日志级别设置，上传任务数设置，DICOM文件数设置，影像查询设置）

2.配置自检

3.QR服务安装及配置

4.采集服务客户端工具（登陆，安装卸载服务，启动停止服务，配置及管理）

5.采集服务（工具登陆账号及配置获取及启动采集作业）

6.采集作业（影像中心检查采集作业，集成平台检查采集作业，第三方医技视图检查采集作业，云到云同步检查采集作业，预警服务作业，校验服务作业，统计服务作业，影像下载作业，影像上传作业，报告下载上传作业，胶片下载上传作业）

7.数据接口实体

8.通用辅助类

9.自动更新服务

新增采集服务辅助工具（由黄铿主要负责开发维护，曹兵辅助）：

1.指定周期数据统计

2.指定检查优先采集

3.影像中心和集成平台采集状态重置

4.ExamCheckData表校验工具（由统一任务管理平台处理，采集服务废弃）

新增：

1.离线影像上传根据（原单独工具，由张小明主要负责开发维护，詹智协助）

2.云端检查导入院内工具（原集成在采集服务，剥离出来后由曹兵负责开发维护）

曹兵 2019/4/16 13:29:01

@研发部-简刚 @测试部-楼静斐

曹兵 2019/4/16 13:29:33

整理出来的采集服务内容 及后续重构的设计

曹兵 2019/4/16 13:29:47

你们看下还有没有漏掉的？

研发部-简刚 2019/4/16 13:54:43

内容看起来基本都有了。

研发部-简刚 2019/4/16 13:55:18

几个新增的工具跟原来的是一起发布吗？用户界面要看看，保证易用性，风格一致。

曹兵 2019/4/16 13:57:23

不是一起发布 采集服务是单独一个 辅助性的独立出来了 根据工程师和运营需要发布 风格上是一致的

测试部-楼静斐 2019/4/16 13:58:42

新增采集服务辅助工具 这个工具单独发布好吧 能合起来吗

测试部-楼静斐 2019/4/16 13:58:52

这个工具单独发布不好吧

曹兵 2019/4/16 14:00:35

这个仅仅是辅助处理问题的 提交测试会一起提交，你也可以放在发布目录 工程师和运营可以根据需要选择 跟采集服务不是放在一起的

测试部-楼静斐 2019/4/16 14:02:13

指定周期数据统计 这个以前采集服务里也有做哦

测试部-楼静斐 2019/4/16 14:02:36

影像中心和集成平台采集状态重置 这个是做什么的

曹兵 2019/4/16 14:03:28

看来很多东西你也不知道

测试部-楼静斐 2019/4/16 14:03:59

解释下咯

曹兵 2019/4/16 14:05:16

这个是采集服务里面设置一段时间，将这段时间影像中心或集成平台已采集的数据状态重置为未采集，重新进行采集。

曹兵 2019/4/16 14:05:50

主要是出现采集失败问题，可以用这个来方便处理，不用打开原平台数据库操作

35.

36.

37.

38.

39.

40.