表系列

1.

注册患者信息

（1）PatientIndex（病人主索引信息）

（2）Visit（病人就诊记录）

（3）ObservationRequest（观察请求记录）

（4）ObservationResult（观察结果表）

第五种为最后插入数据的表。

（5）LogClinicDataOperation（临床数据操作日志）

2.

这两张表可通过FileUID（文件UID）这个字段进行关联。

业务上先查的Document表，再根据FileUID查的DicomStudy表，所以先定为下面的前后关系。

（1）Document（文档记录）

（2）DicomStudy（影像的检查信息）

（3）Document和DicomStuty表需要去ServiceConfigMst（服务配置表）中获取内外网地址等，去拼凑出来url

其中，ObservationRequest（观察请求记录）表会通过PatientID，PatientMasterID，PIDAssigningAuthority这三个条件又一次关联Document（文档记录）表。

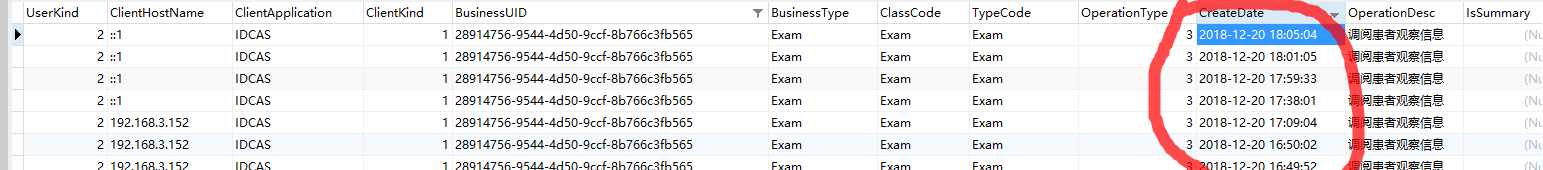
3.

关于LogClinicDataOperation（临床数据操作日志）

发现每一次调阅查看改归档信息，就会往数据库里增加一条记录。

\_logClinicDataOperationRepository.Insert(log);

反应到数据库



4.

ShortUrlAccess（短链接访问表（可放到Redis的哈希表中））

Kind： 1 影像浏览短链接 2 检查报告短链接 3 患者短链接

AccessMode：0 无限制 1 验证码访问 2 有效期访问



5.

PixRuleMst（病人交叉索引规则配置）

PixRuleFieldMst（病人交叉索引规则字段配置）

两张表相当于配置表，不会插入数据，只会获取。

6.

DataSourceSetup（数据来源对应检查类型）

7.

8.

要记忆的：

1.

注册患者信息

因为Visit（病人就诊记录）表中有PatientIndex中的PatientMasterID（病人主索引ID），而ObservationRequest表中有Visit表中的ObservationUID（观察唯一序号），而ObservationResult（观察结果表）中又有ObservationRequest表中的ObservationUID（关联观察请求序列号），所以这四张表的存在关系，就是1234如下排列。

（1）PatientIndex（病人主索引信息）

（2）Visit（病人就诊记录）

（3）ObservationRequest（观察请求记录）

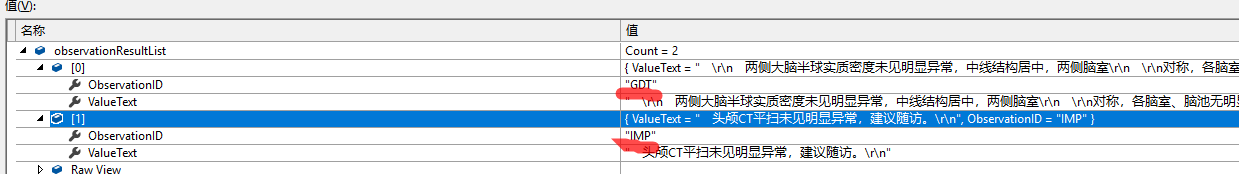
（4）ObservationResult（观察结果表）

第五种为最后插入数据的表。

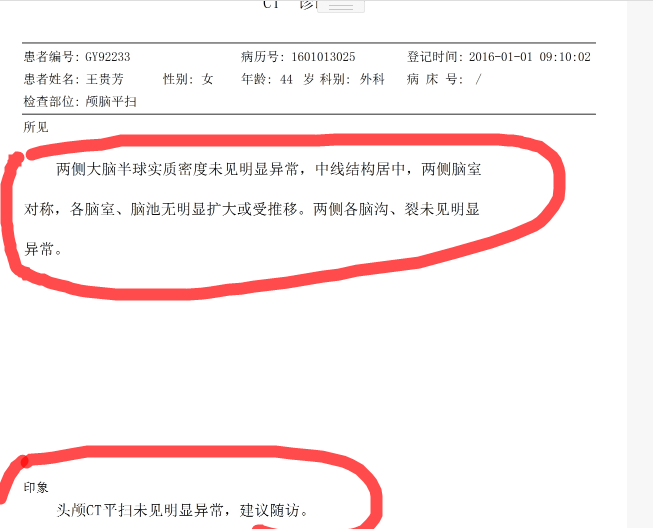
（5）LogClinicDataOperation（临床数据操作日志）

其中ObservationRequest（观察请求记录）和ObservationResult（观察结果表）是一对多，就像下面，同一个观察，会有两个结果。

一个是所见，一个是印象。



反馈到报告单子上：



观察数据库发现成对出现，一格式影像所见，一个是影像诊断。



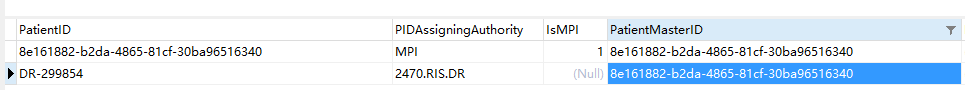
2.

云归档。数据采集那边，还有协同这边把业务走完，都要把所有记录归档到云归档这里。

PatientIndex（病人主索引信息）的信息必然是有的，注册的时候必然是有插入或者更改（所以必然存在PatientMasterID，IsMPI=1，这样的初始数据）。而接下来的ObservationRequest（观察请求记录）也必然是有的。Visit表不一定有。

同样的PatientMasterID（病人主索引ID），可以有多条数据，因为针对同一个病人，归档来源可以是多家不同的机构，不同的检查。

如：



关于病人主索引的逻辑，自己整理：

（1）是否存在PatientMasterID（病人主索引ID）：

a.

如果没有，建立一条信息，new一个新的PatientMasterID，并标记为主索引，IsMPI 为true（IsMPI=1）。再把所有病人信息往里面填写。

b.

如果有，找到该PatientMasterID对应的数据，因为PatientID（病人号），PIDAssigningAuthority（分配机构ID）的不同，可能存在多条，所以要根据注册的搜索条件，要匹配全，去找到这条记录。

亮哥这边写的，以前的代码，没有dal层，也就没有了重用。要建立新的Dal层，再模仿协同的GetPatientIndex(SearchParam param)把查询方法给定义出来，再把当前所有的搜索条件传入，这样就可以拿到满足当前条件的entity，如果存在，就是插入，如果不是，就是update。就方便得多了。

这里又涉及两种：

i.

有病人主索引，但是根据PatientID（病人号），PIDAssigningAuthority（分配机构ID）没有找到数据，那就是insert。

主索引为当前的主索引PatientMasterID，数据为新的。

ii.

有病人主索引，而且根据当前PatientID（病人号），PIDAssigningAuthority（分配机构ID）也找到了数据，那就是update。

总结：三种情况。目前自己整理是这样的。

（2）原始：

是否存在PatientMasterID（病人主索引ID）：

a.

如果没有，建立一条信息，new一个新的PatientMasterID，并标记为主索引，IsMPI 为true（IsMPI=1）。再把所有病人信息往里面填写。

b.

如果有，找到该PatientMasterID对应的数据，因为PatientID（病人号），PIDAssigningAuthority（分配机构ID）的不同，可能存在多条，所以要根据注册的搜索条件，要匹配全，去找到这条记录。

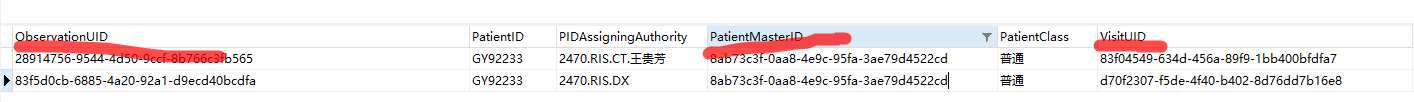
3.

同一个主索引ID，可能有多条记录。可通过病人号，病人号分配机构去进一步精确到具体的一条记录。

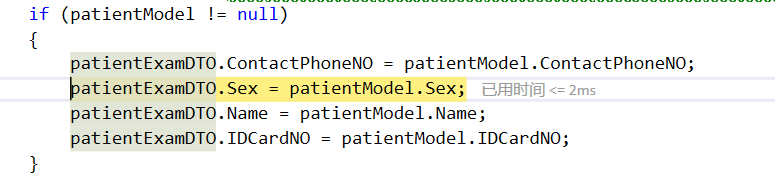


一条主索引表记录，必然有一条观察请求表。通过主索引ID，病人ID，分配机构ID去关联。

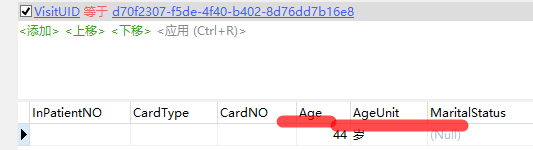
var patientModel = patientIndexSql.Where(a => a.PatientMasterID == observationRequestModel.PatientMasterID && a.PatientID == observationRequestModel.PatientID && a.PIDAssigningAuthority == observationRequestModel.PIDAssigningAuthority).FirstOrDefault();



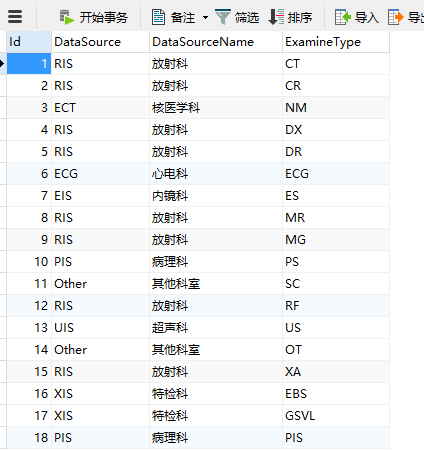
通过病人主索引表去获取到电话号码，性别，姓名，身份证号码。



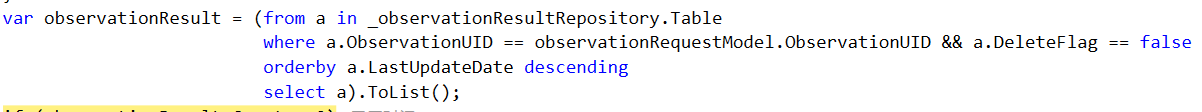
观察请求表中如果visitID不为null，就可以查到就诊visit表中的就诊信息。主要是年龄以及年龄的单位。



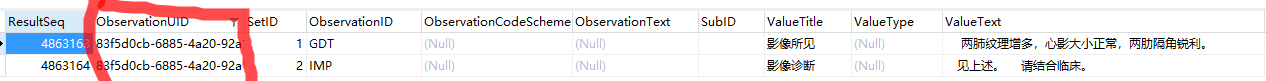
DataSourceSetup（数据来源对应检查类型）通过DataSource（数据来源标识）和ExamineType（对应检查类型）获取当前观察请求记录对应的DataSourceName（数据来源名称）



通过观察UID，以及保证未删除状态，查询到观察结果。



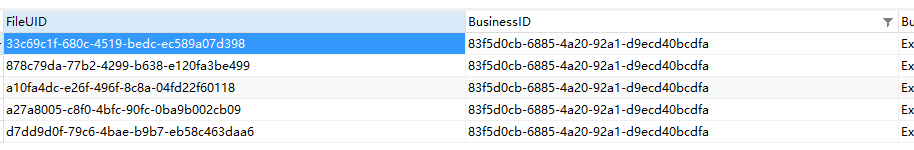
一条观察请求表，同一个observationUID会对应两张观察结果表。因为涉及到影像所见，和医生的诊断。



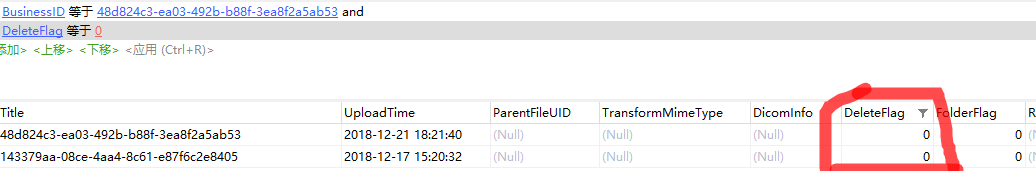
而Document表，通过BusinessID与观察请求表关联起来。一个观察请求表的ObservationUID可以对应多张Document的记录。

注意：需要考虑到DeleteFlag == false的情况。





虽然当然businessID对应Document表有很多张，但是逻辑删除一关联，就只剩两张了。



查询条件摘录（在获取检查详情中）：

var querydocument = new DocumentQuery

{

BusinessType = patientExamDTO.ClinicInfoType,

BusinessID = businessid,

OrganizationID = patientExamDTO.OrganizationID,

是否返回EWorldViewer地址

WithEWorldViewerUrl = true,

是否返回DICOM原始文件地址

WithDicomDownload = false,

是否返回DICOMDIR地址

WithDicomdirPath = false

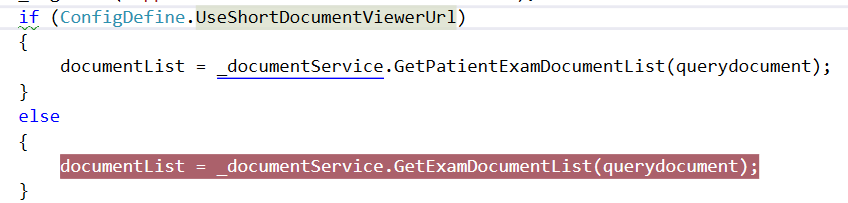
};

观察请求表中的OrganizationID（机构ID）

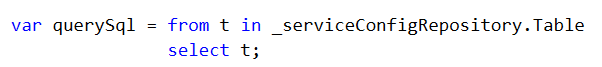
通过ServiceConfigMst（服务配置表）获取内外网rul地址。这里的EndPoint

具体：

根据是否启用短链接访问来进入不同的方法。

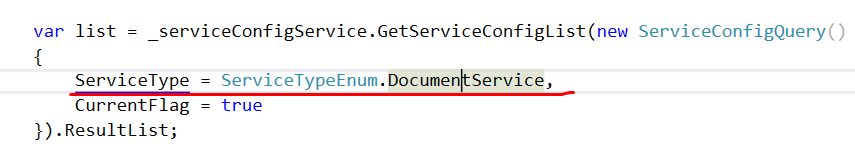


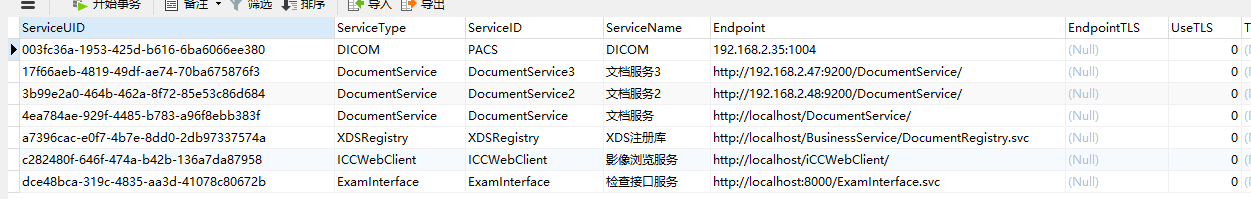
根据条件查询服务配置列表信息——ServiceConfigMst（服务配置表）

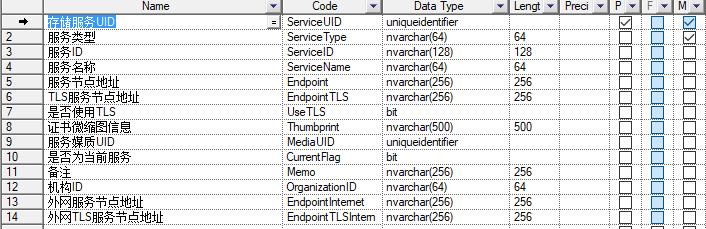


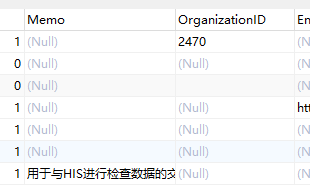
界定了返回的服务配置列表信息未文档服务。

看到数据库中有三个，这里应该是能配置成文档集群的。类似图片，文件的海量上传。

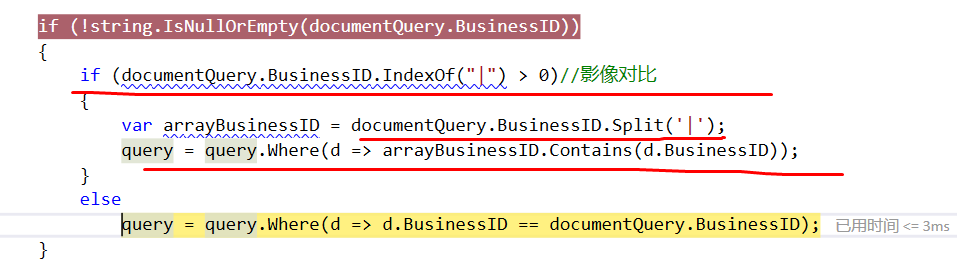




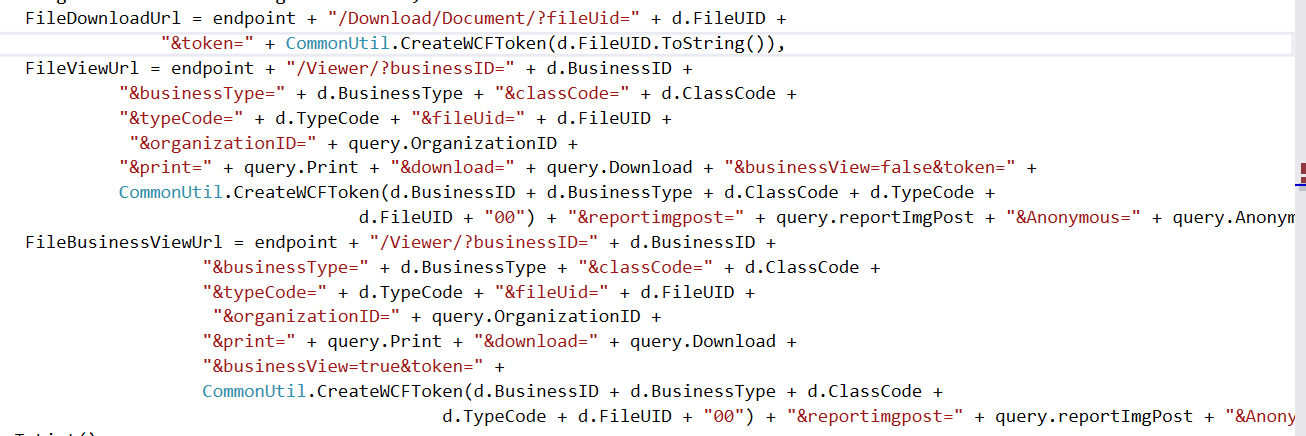




影像对比的选择：

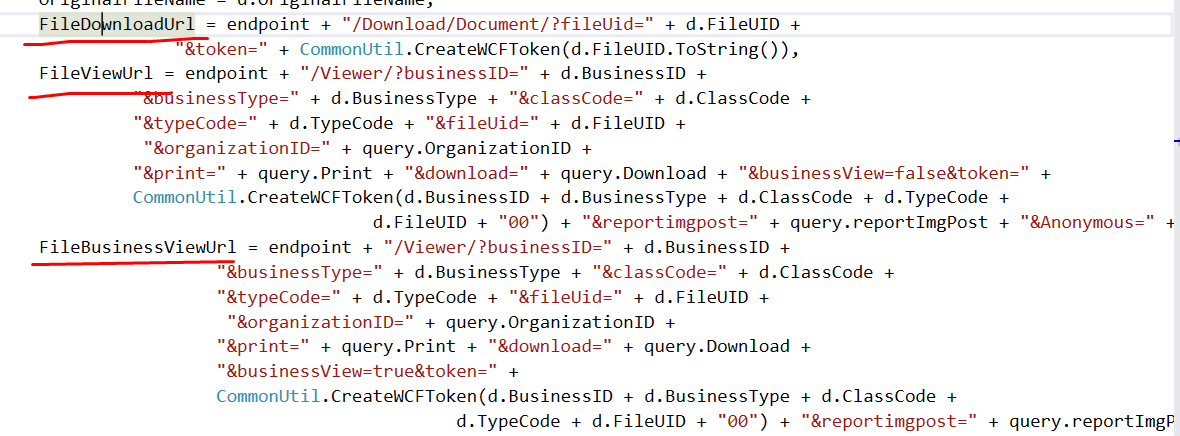


通过Document的查询，然后进行url地址的拼接



如果需要返回Dicom地址，就对dicom的url进行拼接

这里和上面一样，都是根据businessID=observationUID获取的document表的数据。先是循环获取FileDownloadUrl（外网文件下载http地址），FileViewUrl（文件浏览http地址），FileBusinessViewUrl（文件浏览http地址，基于业务的浏览，如果同个业务有多个文件，则可以通过该地址一并浏览）

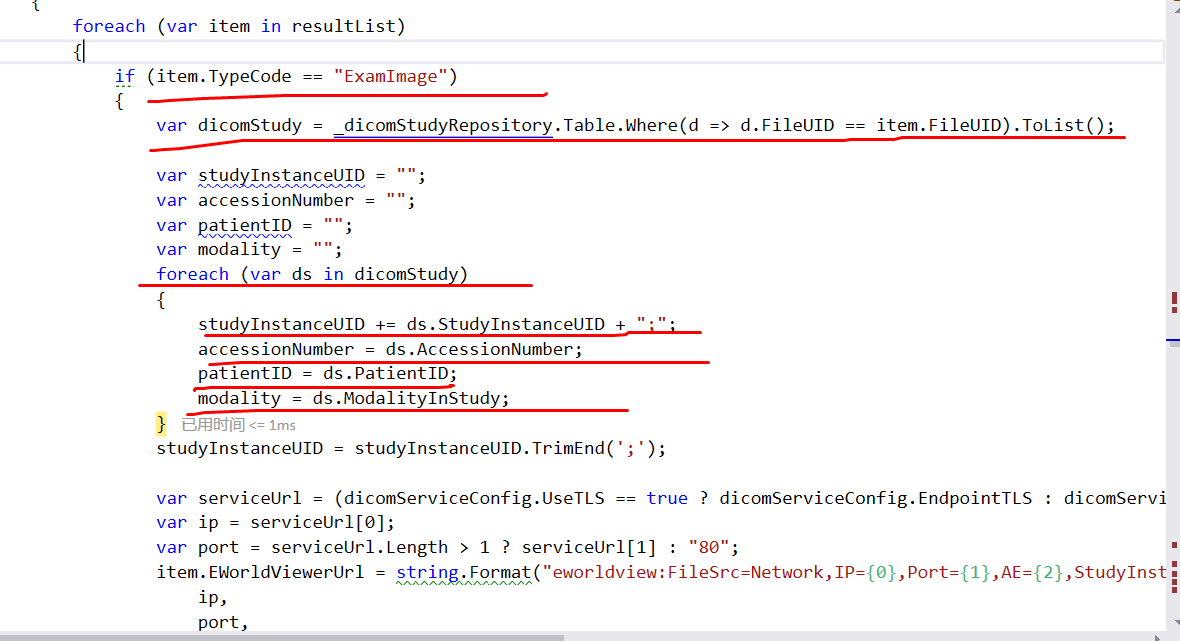


然后如果存在WithDicomDownload（是否返回DICOMDIR地址），WithDicomdirPath（是否返回DICOMDIR地址）WithEWorldViewerUrl（是否返回EWorldViewer地址）这种，根据数据的TypeCode（文档小类代码）选择是去哪个函数处理，也可能访问别的接口获取下载地址。

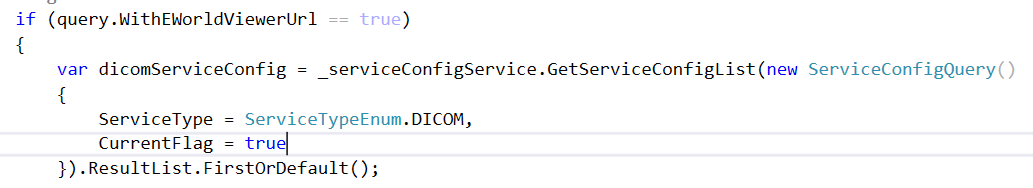


ExamImage的话，就去DicomStudy表中查询数据进行拼接，类似Document表那样各种数据拼接在一起。

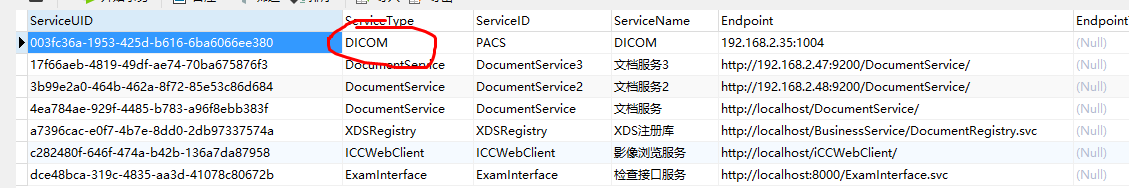
如果同一个FileUID对应多个dicomStuty记录，就都拼接在一起。



如果需要返回EWorldViewer地址，就要去操作DicomStudy（影像的检查信息）表。

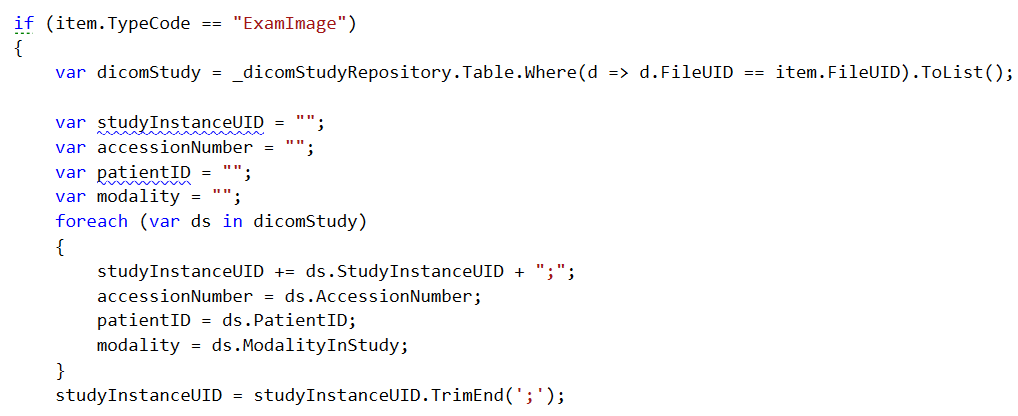


又走到了

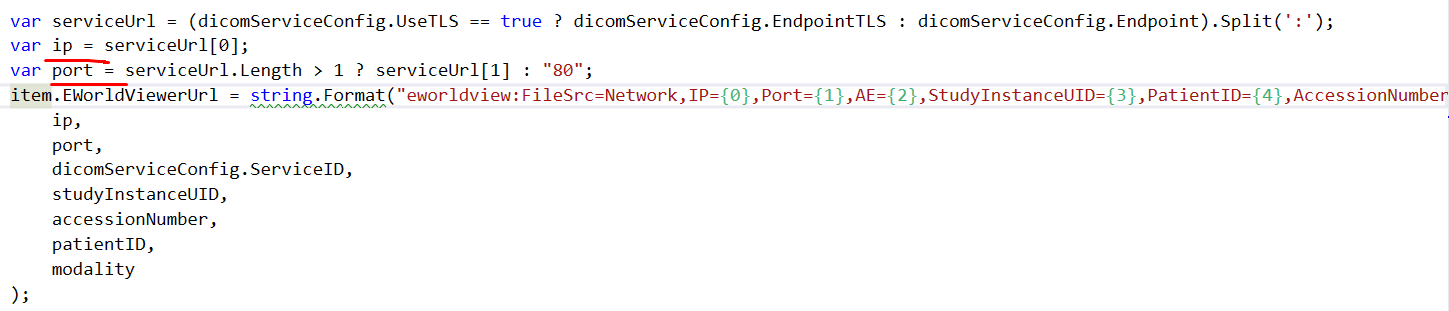


查dicomStuty表

对查询结果进行拼接，整出dicom的url地址

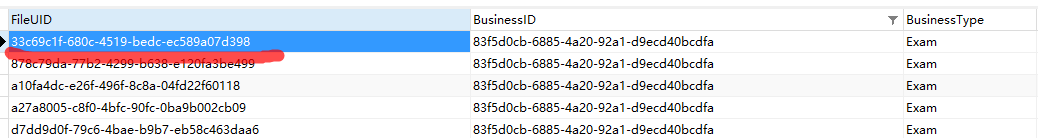


拼接ip，端口（port）

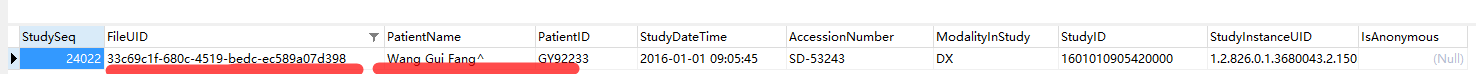


通过FileUID和Document表的FileUID进行关联。

document表：



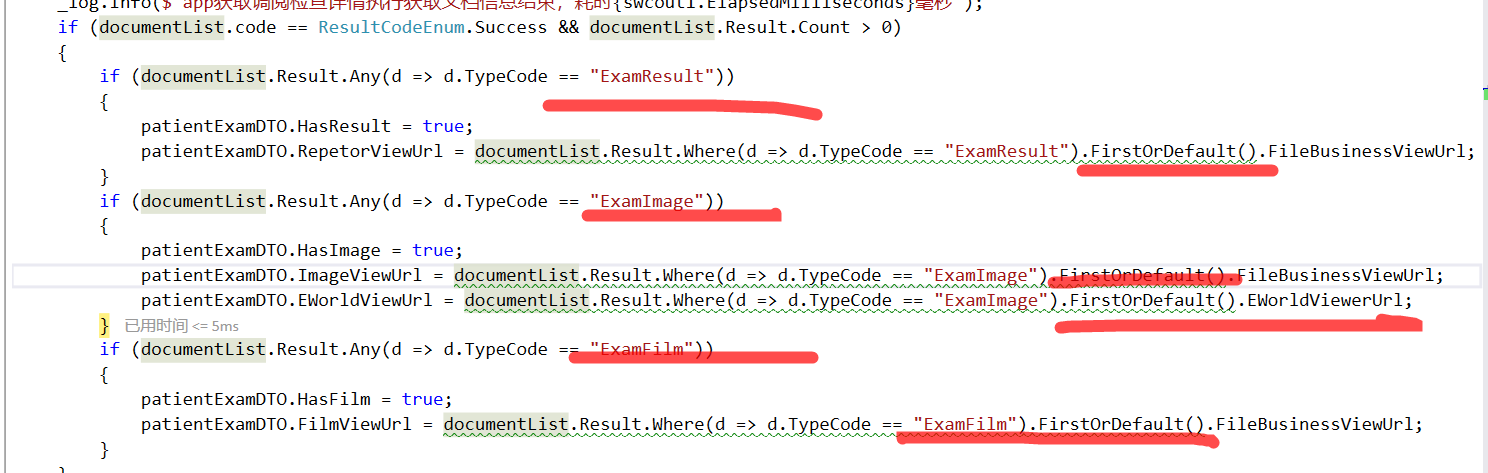
dicomStudy表





最后的结果，又是取ExamResult，ExamImage这种小类的第一个，然后把结果赋值给DTO。

在想，如果是多个，就用list去接受。类似二类科室，就是list接受。



我用了空的url去测试最外层的转换方法：

newHost为空传进来, endpoint是localhost开头的一堆地址



public static string TransLocalhostForClient(string endpoint, string newHost = "")

{

if (string.IsNullOrWhiteSpace(endpoint)) return string.Empty;

var localhost = "://localhost";

if (endpoint.IndexOf(localhost) != -1)

{

跳进来

var host = newHost.IndexOf("http://") != -1 ? newHost.Substring(newHost.IndexOf("http://")+7) : "";

if (string.IsNullOrWhiteSpace(newHost))

{

跳到这里，结果HttpContext.Current为null

if (HttpContext.Current != null)

{

host = HttpContext.Current.Request.ServerVariables["HTTP\_X\_FORWARDED\_HOST"];

if (!string.IsNullOrEmpty(host))

{

//采用代理服务器的情况，得取到客户端访问的host

\_log.Debug("HttpContext.Current.Request.ServerVariables[\"HTTP\_X\_FORWARDED\_HOST\"]：" + host);

host = host.Split(',').FirstOrDefault();

var strVirtualDirectory = string.IsNullOrWhiteSpace(ConfigDefine.VirtualDirectory) ? string.Empty : ("/" + ConfigDefine.VirtualDirectory);

endpoint = endpoint.Replace(host, ConfigDefine.IDCASUrl.Replace(strVirtualDirectory, "").TrimEnd('/'));

\_log.Debug("Endpoint地址为代理模式，将转换为：" + endpoint);

}

else

{

host = HttpContext.Current.Request.Url.Host + ":" + HttpContext.Current.Request.Url.Port;

}

}

else if (WebOperationContext.Current != null)

{

跳到了这里



{Connection: Keep-Alive

Content-Length: 623

Content-Type: text/xml; charset=utf-8

Accept-Encoding: gzip, deflate

Expect: 100-continue

Host: localhost:9201(因为是wcf来访问？)

VsDebuggerCausalityData: uIDPo9XqbiSIx+VNn+yd21QLnzYAAAAAPISOx7t8I0Gv3T1wuk3JmTlJy/nlJSNFkSimlGGkHaMACQAA

SOAPAction: "http://www.tomtaw.com.cn/IImageArchiveService/GetPatientExamInfo"

}

host = WebOperationContext.Current.IncomingRequest.Headers["Host"];

}

else if (OperationContext.Current != null)

{

//获取消息发送的远程终结点IP和端口

var property = OperationContext.Current.IncomingMessageProperties[RemoteEndpointMessageProperty.Name] as RemoteEndpointMessageProperty;

if (endpoint != null)

host = property.Address;

}

}

if (!string.IsNullOrEmpty(host))

{

var strVirtualDirectory = string.IsNullOrWhiteSpace(ConfigDefine.VirtualDirectory) ? string.Empty : ("/" + ConfigDefine.VirtualDirectory);

if (!string.IsNullOrWhiteSpace(strVirtualDirectory))

{

host = host.Replace(strVirtualDirectory, "").TrimEnd('/');

}

endpoint = endpoint.Replace(localhost, "://" + host);

}

}

4.

重新整理了患者信息(patientIndex)的注册+观察申请(ObservationRequest)信息，就诊信息(visit)+观察申请结果(observationResult)信息，三个信息的注册。

（1）病人主索引

a.

患者信息，默认采集，不传主索引，当传了主索引，默认是更新，当又传了DeleteFlag为True，意味着删除。物理删除。

采集时，主要判断的是PatientID(病人号)，分配机构(2401.RIS这种)，主索引，首先会各种信息（身份证，病历卡号，市民卡号等等）匹配，无论如何找到病人的主索引，如果又有，按照上面的三个参数进行匹配，有，就是update，没有就是insert。

而当没有主索引，建立一个，之后再匹配一次，必然是匹配得到的，因为刚才建立，然后因为病人号（PatientID）用的主索引的GUID，所以用三巨头参数去匹配，必然为null，则insert，插入此条数据。

这样的更新，也就是覆盖匹配机制，必然导致当同一个病人，在不同科室就诊，会有两条注册信息存在数据库中，比方同一个病人，不同的申请，2401.RIS，2470.eCenter，这样，就会两条数据在库中。可确实是这样的逻辑，因为查询的时候，就是通过三巨头在主索引表中获取了数据，ToList()，存在的情况采取ObservationRequest，Visit，ObservationResult表中获取相应数据，然后整合，返回数据结果。

b.

来查询的时候，用的也是三巨头参数，看有没有匹配的病人，然后拿三巨头去接下来的ObservationRequest表中查询匹配的数据。

因为我上传申请信息用的完全匹配，即便同一个科室上来，也就是三巨头一样，但其他的检查项目，单号不一样，也是不一样的检查，这样就避免了要紧数据的覆盖。

所以预想会是，同一个病人主索引(PatientIndex)三巨头的查询，可能在申请表（ObservationRequest）中有多条数据，然后可以根据每条的ObservationUID去结果集中查询结果，结果会包含影像所见，影像诊断，还有一堆检验的数据，我是以ObservationUID+ObservationID去甄别，这样同一个ObservationUID，ObservationID重复了，就是update，否则都是insert进来，保证数据的完整性。

申请：

var obsEntity = \_observationRequestRepository.Table.FirstOrDefault(t => t.PatientID == query.PatientID && t.PIDAssigningAuthority == query.PIDAssigningAuthority && t.PatientMasterID == query.PatientMasterID && t.DeleteFlag == false && t.PatientClass == query.PatientClass && t.OrganizationID == query.OrganizationID && t.AccessionNumber == query.AccessionNumber && t.ServiceSectID == query.ServiceSectID&&t.ServiceID==query.ServiceID);

结果：

var existResult =

\_observationResultRepository.Table

.FirstOrDefault(t => t.ObservationUID == obs.ObservationUID && t.DeleteFlag == false &&

t.ObservationID == observationReuslt.ObservationID

/\*&&t.SetID == setID\*/);//加观察结果顺序准确获取该条记录

（2）申请

也是一样的逻辑，有ObservationUID，是更新，如果DeleteFlag为True，删除。逻辑删除。

采集的时候，用的是上面的多参数匹配，不少采集，也不轻易去覆盖已有数据。

（3）结果

和上面一样，有ObservationUID，是更新，如果DeleteFlag为True，删除。逻辑删除。

采集的时候，用的是上面的多参数匹配，不少采集，也不轻易去覆盖已有数据。

暂时记录：

（1）获取检查详情：http://ip:port/v2/public/get\_exam\_info

入参备注：

accessionNumber（检查号），idCardNO（身份证号），observationUID（检查业务唯一号）不能同时为空

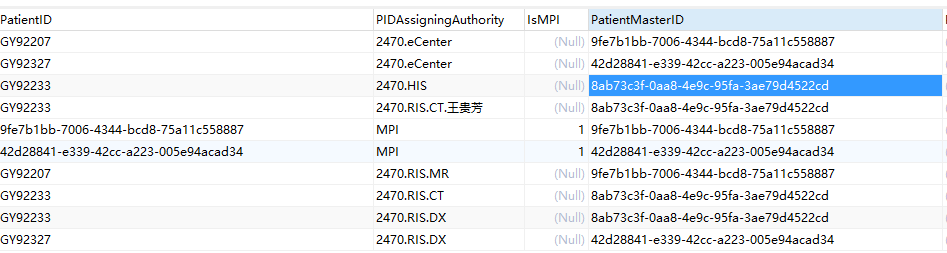
当传的是身份证号，也是通过身份证号先去主索引表中获取所有的不重复的主索引PatientMasterID，然后再去ObservationRequest表中查询。

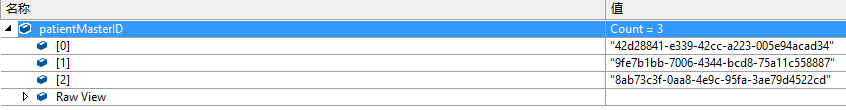
入参，身份证：

"idCardNO":"217A087D889A985FF86AB90F3050283930BC27DA937302B2"

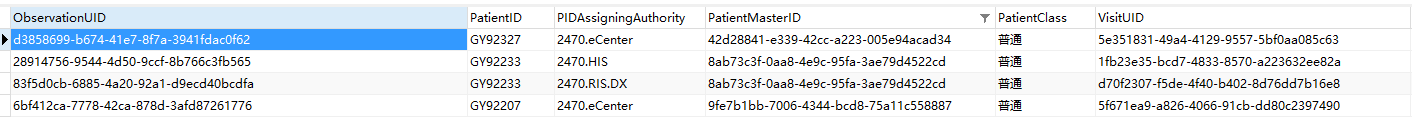
解密后：330724198911020312

对应的主索引，有三个不重复的主索引ID





然后带着这个patientMasterID的list，去ObservationRequest表中查询所有的数据，有四条。



这里却将四条数据，根据请求申请时间反序，获得第一条，也就是最新的那次检查：

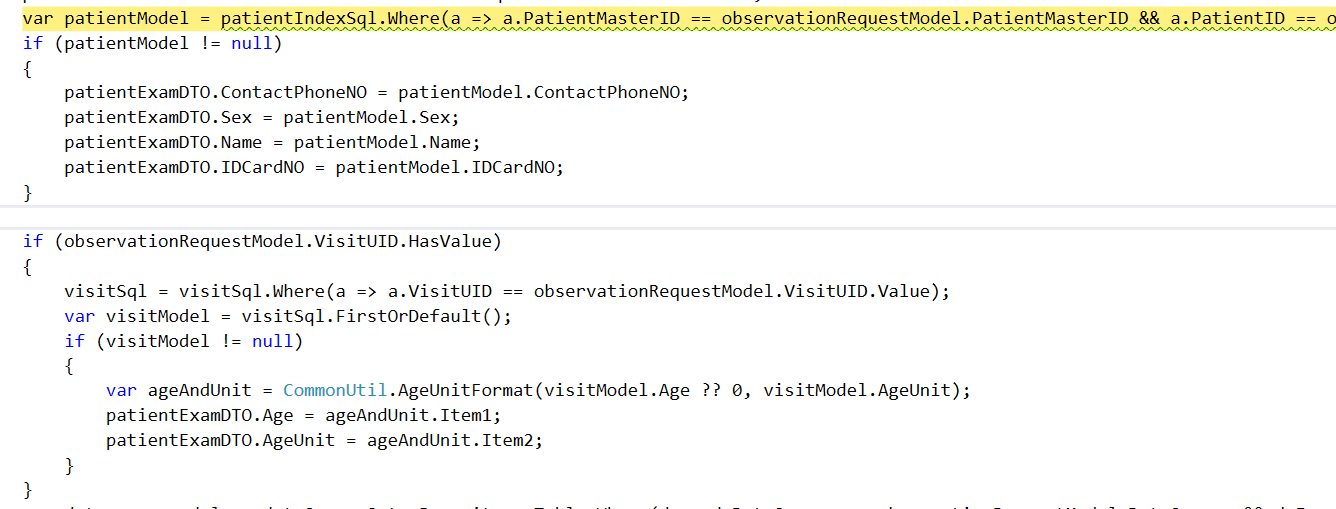
var observationRequestModel = observationRequestSql.OrderByDescending(a => a.ObservationDate).FirstOrDefault();

也就是说，当用身份证查询的时候，获取的是最近一次检查的结果。

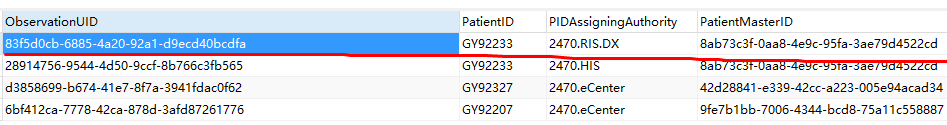
接下里去PatientIndex表，还有visit表中获取对应的数据

其实这里，按三巨头去查，如果是同一个科室，同一个病人，是有问题的，前面用的按时间排序，这里也应该用观察时间，或者说创建时间排序，这样至少是最新的那条记录。

var patientModel = patientIndexSql.Where(a => a.PatientMasterID == observationRequestModel.PatientMasterID && a.PatientID == observationRequestModel.PatientID && a.PIDAssigningAuthority == observationRequestModel.PIDAssigningAuthority).FirstOrDefault();



按上面的数据真实姓继续说，如果按三巨头，因为申请表是按申请时间倒叙，也就是获得了4条数据中的这条。



还好三巨头对应的是一条数据。不对，当患者信息注册的时候，患者这里是不绝对定位的，Any存在匹配的就是三巨头的数据，如果存在，就update。那也就对了。无论后面的检查申请中有多少个ObservationUID，只要三巨头一样，对应的，在PatientIndex表中，只会有一条数据的。因为重复的，在采集的时候都覆盖掉了。



Visit是唯一对应的，没有关系。

接下来去DataSourceSetup（数据来源对应检查类型）获取DataSourceName(数据来源名称)

var datasourcemodel = \_dataSourceSetupRepository.Table.Where(d => d.DataSource == observationRequestModel.DataSource && d.ExamineType == observationRequestModel.ServiceSectID).FirstOrDefault();



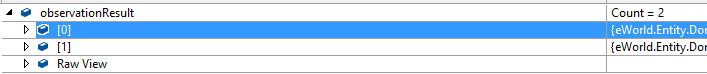
接着到观察结果中获取观察结果

var observationResult = (from a in \_observationResultRepository.Table

where a.ObservationUID == observationRequestModel.ObservationUID && a.DeleteFlag == false

orderby a.LastUpdateDate descending

select a).ToList();



if (observationResult.Count > 0)

{

foreach (var item in observationResult)

{

if (item.ObservationID == "Gross")

{

patientExamDTO.GrossSight = item.ValueText;

}

if (item.ObservationID == "GDT")

{

patientExamDTO.StudyDescription = item.ValueText;

}

if (item.ObservationID == "IMP")

{

patientExamDTO.DiagnosticImpression = item.ValueText;

}

}

}

接下来就通过ObservationUID和Document表中的BusinessID进行关联，进入到了Document的世界：

string businessid = observationRequestModel.ObservationUID.ToString();

var querydocument = new DocumentQuery

{

BusinessType = patientExamDTO.ClinicInfoType,

BusinessID = businessid,

OrganizationID = patientExamDTO.OrganizationID,

WithEWorldViewerUrl = true,

WithDicomDownload = false,

WithDicomdirPath = false

};

var documentList = new ModelResult<List<DocumentViewDTO>>();

if (ConfigDefine.UseShortDocumentViewerUrl)

{

documentList = \_documentService.GetPatientExamDocumentList(querydocument);

}

else

{

documentList = \_documentService.GetExamDocumentList(querydocument);

}

小总结：注册是怎么进来的，查询也是一样的顺序。先患者主索引，再申请表，就诊表，观察结果表。

患者信息，通过三巨头是可以唯一性的。

患者信息和观察申请，是一对多。通过三巨头查询的话。

观察申请和观察结果，也是一对多的关系。

观察申请和visit是一对一，是必然存在的，通过我的接口注册的话。

5.

6.

7.

8.