温岭开发

是

1.

PatientQuery中多出来的参数：

/// <summary>

/// 语言

/// </summary>

public string Language { get; set; }

这个暂时没有加

/// <summary>

/// 是否为病人主索引。1:是，0:否。

/// </summary>

public string IsMPI { get; set; }

逻辑还是存在就update，不存在就insert

2.

（1）温岭新增的3个接口

（2）修复web页面按检查时间搜索

（3）修复task的更新

3.

查询（要删除）做了标记的。修改的地方。

4.

//查看匹配当前存在的主索引guid，当前注册的用户是否存在

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

{

//当前用户是存在的，就是update

patientIndex = \_patientIndexRepository.Table.Where(a => a.PatientMasterID == patientmasterid && a.PatientID == query.PatientID && a.PIDAssigningAuthority == query.PIDAssigningAuthority).FirstOrDefault();

patientMatch = patientMatch.ForMember(t => t.PatientID, dto => dto.Ignore());

patientMatch = patientMatch.ForMember(t => t.PatientMasterID, dto => dto.Ignore());

patientMatch = patientMatch.ForMember(t => t.PIDAssigningAuthority, dto => dto.Ignore());

patientIndex = Mapper.Map(query, patientIndex);

isAdd = false;

}

其实这种映射发生覆盖原数据的原因，就是采集服务采到数据在推送的时候，有些字段为null，造成原有数据的丢失。

按我自己的思路，推送的时候，只要字段不为null，以为着就要映射，就要覆盖。

可如果就是要覆盖，就是要update的需求呢？而我其他的，数据库中的内容都保持原样，比方一个flag字段，今天传了一个1，十年都没传，就一直存在着，而不会被覆盖了。

其实还是主要看需求，如果是支持覆盖，这种方法是没有错的。所以苗哥说让我不哟啊动，他那边去操作，因为他可以控制，必须参数都保证了，才往我的接口这里传。

就像我调融创的接口前，我也是可以去控制我的数据库参数的。这样就保证了只有是标准的数据，才可以往上传。

5.

患者信息注册：

（1）参数校验

（2）Patient特殊字符处理

（3）入参有主索引并且该主索引是正确，存在的

a.能匹配，更新（update）

b.不能匹配，该主索引插入

（4）入参没有主索引，通过病人信息无所不用其极地去匹配，

a.匹配成功，返回这个主索引的Guid

b.匹配失败，生成一个主索引的信息，并把主索引Guid给返回

上面的第四条第二种说法（之前的方式）:

（4）入参无论有没有主索引，通过病人信息无所不用其极地去匹配，去获取主索引Guid

a.匹配成功，返回这个主索引的Guid

b.匹配失败，生成一个主索引的信息，并把主索引Guid给返回

这里如果主索引入参是有的，就采用这个guid，因为如果guid是正确的，那再第（3）步肯定完成了插入或者更新，因而这里的第一步肯定是能匹配到一个主索引guid的，但这里没有，说明是第一次进来，但采集机构，第三方那边又想命名这个主索引Guid

弊端就是如果主索引是正确的，再第三步已经更新了一遍，但这里又获取一次，又更新了一次。更新了两遍。

（5）通过第四步的匹配，已经拿到了主索引表中必然存在的主索引Guid，将入参的病人ID和分配结构ID与之一起去表中匹配

a.匹配成功，存在该记录，Update

b.匹配失败，不存在该记录，Insert

（5）将操作写入临床数据操作日志（logclinicdataoperation）（是否需要呢，因为BusinessUID来自ObservationRequest表）

而且如果要写，这两个参数是需要调用接口方给我的，而如果我这里是给采集用的，那肯定是pc端和医生。

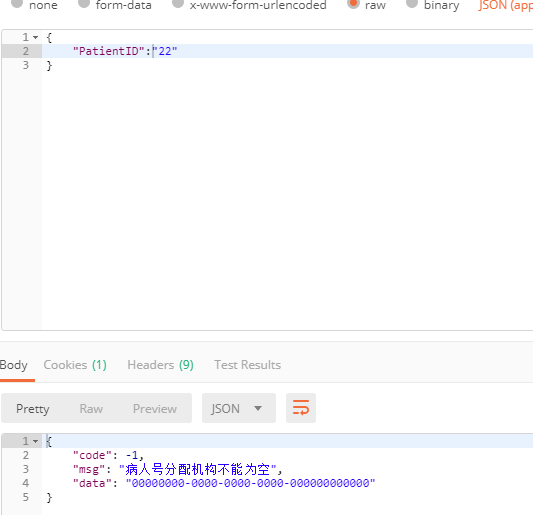
UserKind：1 公众用户 2 医生

ClientKind: 1 pc端 2 手机端

else最终去掉，因为如果带着主索引Guid过来，所有资料又匹配不上，那就是按对方的主索引guid进行创建主索引。可能是对方另外系统中的guid，两边进行关联。

6.

开始调试：



入参必须问题，ok。

7.

感觉更新功能不需要这么较真，因为请求这个接口只是为了获取病人主索引号？

8.

存在主索引，update，不存在，insert或者update？

9.

唯一性的问题，之前是patientID+分配机构+主索引，不对了，感觉。

10.

11.

第二个：

1.2.1.2.1. 患者检查申请信息上传，发现visit表的入参也是在这里面

1.

（1）按照ObservationRequestUID来查，如果有

存在，update

不存在，insert

（2）按照所有的必传参数，去库里查，获取ObservationUID

存在：update

不存在：insert，并且new一个VisitUID的guid

（3）把就诊信息注册到Visit表中。

经过上面，必然存在了，也拿到了ObservationRequest表中的VisitUID，这样，带着VisitUID去查询Visit表，

（1）数据存在，update

（2）不存在，insert

先创建或更新ObservationRequest表，这时候，必然存在了ObservationRequest记录，其中也包含了VisitUID，这样就可以根据ID去创建或者更新下面的Visit表信息了。

2.

是否要将Visit以包含的性质存在于接口中

3.

4.

5.

问题：

1.

针对映射，同样的，被判断为同一个用户的信息，到底什么标准去映射，因为映射意味着覆盖。

我可以选择忽略某些字段而不去映射的，但是具体是哪些，我不知道。

2.

两个类都放一起，造成更新很麻烦，会覆盖。丢失参数的话，你需要去保护。ignore许多

3.

{

"PatientID":"02200040",

"PIDAssigningAuthority":"2402.PS",

"PatientMasterID":"835fae70-9d0b-4cd2-bec0-e0e84262d2d6",

"PatientClass":"门诊",

"OrganizationID":"2402",

"OrganizationName":"余杭区第一人民医院",

"AccessionNumber":"2415565",

"ServiceID":"CGX334833",

"ProcedureName":"左踝关节正侧位",

"ServiceSectID":"DR",

"RegTime":"2019-01-02 09:36:33",

"ObservationDate":"2019-01-02 09:36:33"

}

{

"code": 0,

"msg": "成功",

"data": {

"ObservationUID": "a5a827d9-1d87-4e3a-a19e-c896a7e33fc9",

"VisitUID": "889d302b-5286-4023-92d7-8f22779220e6"

}

}

多了点参数，居然就过变了：

{

"PatientID":"02200040",

"PIDAssigningAuthority":"2402.PS",

"PatientMasterID":"835fae70-9d0b-4cd2-bec0-e0e84262d2d6",

"PatientClass":"门诊",

"OrganizationID":"2402",

"OrganizationName":"余杭区第一人民医院",

"AccessionNumber":"2415565",

"ServiceID":"CGX334833",

"ProcedureName":"左踝关节正侧位",

"ServiceSectID":"DR",

"RegTime":"2019-01-02 09:36:33",

"ObservationDate":"2019-01-02 09:36:33",

"MedRecNO":"33101",

"Age":"8",

"AgeUnit":"岁"

}

{

"code": 0,

"msg": "成功",

"data": {

"ObservationUID": "df412852-9e80-4fc2-ae89-d4c07df53631",

"VisitUID": "14a9a321-a72b-4bef-adb8-624b19088b67"

}

}

4.

铿哥，单接口提供：注册、更新（含删除）的功能。温岭市一的接口需要的话请你加班处理一下好吧。@hk   
  
hk 2019/1/11 17:36:47

5.

请求表中数据加进去后，到第二个映射的时候，用的还是入参，而不是

加了这个不映射之后就ok了。

// 默认不需要映射两个uid

mapExpObs = mapExpObs.ForMember(t => t.ObservationUID, dto => dto.Ignore());

mapExpObs = mapExpObs.ForMember(t => t.VisitUID, dto => dto.Ignore());

6.

检查申请

入参保留：

{

"PatientID":"4C15FF6DBD512AD2ECE62AEE373EBA12",

"PIDAssigningAuthority":"2402.PS",

"PatientMasterID":"3d2d5c4d-97f5-40dd-b988-ea4481c93c37",

"PatientClass":"门诊",

"OrganizationID":"D9D1F58C6B410FC9",

"OrganizationName":"余杭区第一人民医院",

"AccessionNumber":"B4E8B249841F56C5",

"ServiceID":"CGX334833",

"ProcedureName":"左踝关节正侧位",

"ServiceSectID":"DR",

"RegTime":"2019-01-11 16:36:33",

"ObservationDate":"2019-01-11 16:36:33",

"MedRecNO":"B8198B514327A3B7",

"Age":"8",

"AgeUnit":"岁"

}

第三方的：

{

"ObservationUID": "AE1810A63DDE9C656066FA922BC9BD32",

"PatientID": "1B78B91CB05BCD13EE1DBA4EFF70273A",

"PIDAssigningAuthority": "74580626.RIS",

"PatientMasterID": "1B78B91CB05BCD13EE1DBA4EFF70273A",

"PatientClass": "体检",

"OrganizationID": "57D087064EAB3E73D3C8541030E7385C",

"OrganizationName": "温岭第一人民医院",

"AccessionNumber": "22A2D0F4A42E26071C8BC1167AC213A1",

"ServiceID": "22A2D0F4A42E26071C8BC1167AC213A1",

"ProcedureName": "体检",

"ServiceSectID": "CT",

"RegTime": "2019/1/12 10:23:12",

"ObservationDate": "2019/1/12 11:06:02",

"MedRecNO": "1B78B91CB05BCD13EE1DBA4EFF70273A",

"Age": "28Y",

"AgeUnit": "岁"

}

7.

8.

第三个：

（1）约束了每次同步必须传的是同一个ObservationUID下的结果集。

1.主键没有传入

2.后续还需要上传化验这些结果，也是用的这个结果注册接口：



3.

修改逻辑，结果上来，不按这个逻辑，按之前的。

因为如果第一次50个记录，第二次40个上来，我覆盖了40个，剩下了10个。必须不剩。所以，先删除，后新增。

不按原来的逻辑。

因为逻辑其实是一份报告，我们自己把他拆成了四张表，三个接口。然而医院那边是一次完整的业务流程，已经有结果了。最终反馈到我们这边。

按新需求来吧。

4.

之前的代码保存：

/// <summary>

/// 上传患者观察结果信息

/// </summary>

/// <param name="query">观察结果请求信息</param>

/// <returns></returns>

private BaseResult UploadObservationResultNew(List<UploadObservationResultQuery> query)

{

Stopwatch sw = null;

var result = new BaseResult { code = ResultCodeEnum.Success, msg = "成功" };

try

{

if (\_logDebug)

{

sw = new Stopwatch();

sw.Start();

\_log.Debug($"收到UploadObservationResultNew请求，传入参数：{JsonConvert.SerializeObject(query)}");

}

// 参数校验已经证明必然存在ObservationUID的数据

var observationUID = query[0].ObservationUID;

var obs =

\_observationRequestRepository.Table.FirstOrDefault(o => o.ObservationUID == observationUID && o.DeleteFlag == false)

// 之前的做法，对每一个进行筛选，确定insert，update，还是delete

//foreach (var observationReuslt in query)

//{

// // 处理更新和删除

// if (observationReuslt.ResultSeq.HasValue && observationReuslt.ResultSeq.Value > 0)

// {

// // 处理更新

// var obsEntity =

// \_observationResultRepository.Table.FirstOrDefault(

// o =>

// o.ResultSeq == observationReuslt.ResultSeq &&

// o.ObservationUID == observationReuslt.ObservationUID && o.DeleteFlag == false);

// if (obsEntity != null)

// {

// // 处理删除

// if (observationReuslt.DeleteFlag.HasValue && observationReuslt.DeleteFlag.Value)

// {

// obsEntity.DeleteFlag = true;

// \_observationResultRepository.Update(obsEntity);

// result.Result.Add(obsEntity.ResultSeq);

// result.msg = "删除成功";

// }

// // 处理更新

// else

// {

// var mapExpObsResult = Mapper.CreateMap<UploadObservationResultQuery, ObservationResult>();

// // 忽略映射主键

// mapExpObsResult = mapExpObsResult.ForMember(t => t.ResultSeq, dto => dto.Ignore());

// // 将入参映射给查询到的obsEntity实体

// obsEntity = Mapper.Map(observationReuslt, obsEntity);

// obsEntity.LastUpdateDate = obs.LastUpdateDate;

// obsEntity.DeleteFlag = false;

// if (!obsEntity.ObservationDate.HasValue && obs.ObservationDate.HasValue)

// obsEntity.ObservationDate = obs.ObservationDate;

// if (!obsEntity.PreliminaryDate.HasValue && obs.PreliminaryDate.HasValue)

// obsEntity.PreliminaryDate = obs.PreliminaryDate;

// if (!obsEntity.AuditDate.HasValue && obs.AuditDate.HasValue)

// obsEntity.AuditDate = obs.AuditDate;

// if (obsEntity.ObservationID == "GDT")

// {

// obsEntity.ValueTitle = "影像所见";

// }

// else if (obsEntity.ObservationID == "IMP")

// {

// obsEntity.ValueTitle = "影像诊断";

// }

// if (!string.IsNullOrEmpty(obsEntity.ResultStatusCode))

// {

// obsEntity.ResultStatusCode = obsEntity.ResultStatusCode;

// obsEntity.ResultStatus = ExamStatus.GetExamResultStatus(obsEntity.ResultStatusCode);

// }

// else if (!string.IsNullOrEmpty(obsEntity.ResultStatus))

// {

// obsEntity.ResultStatusCode =

// ExamStatus.GetExamResultStatusCode(obsEntity.ResultStatus);

// obsEntity.ResultStatus = obsEntity.ResultStatus;

// }

// if (string.IsNullOrEmpty(obsEntity.ResultStatus))

// {

// obsEntity.ResultStatus = obs.ResultStatus;

// obsEntity.ResultStatusCode = obs.ResultStatusCode;

// }

// if (!string.IsNullOrWhiteSpace(obsEntity.ResultStatus))

// {

// obsEntity.ResultStatus = obsEntity.ResultStatus.Trim();

// }

// if (!string.IsNullOrWhiteSpace(obsEntity.ResultStatusCode))

// {

// obsEntity.ResultStatusCode = obsEntity.ResultStatusCode.Trim();

// }

// \_observationResultRepository.Update(obsEntity);

// result.Result.Add(obsEntity.ResultSeq);

// result.msg = "更新成功";

// }

// }

// else

// {

// result.code = ResultCodeEnum.Fail;

// result.msg = "观察序列号对应的数据并不存在";

// return result;

// }

// }

// else // 观察申请结果采集

// {

// #region ObservationResultList

// \_log.Debug("开始处理ObservationResultList信息");

// // 之前的

// //var existResult = \_observationResultRepository.Table.Where(t => t.ObservationUID == obs.ObservationUID && t.DeleteFlag == false).ToList();

// // 应当根据当前的入参去准确地获取这条数据

// //var setID = short.Parse(observationReuslt.SetID);

// 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

// var mapExpObsResult = Mapper.CreateMap<UploadObservationResultQuery, ObservationResult>();

// // 结果表中已经存在结果，update

// if (existResult != null) //修订结果

// {

// // 更新这个数据

// // 忽略映射主键

// mapExpObsResult = mapExpObsResult.ForMember(t => t.ResultSeq, dto => dto.Ignore());

// // 将入参映射给查询到的existResult实体

// existResult = Mapper.Map(observationReuslt, existResult);

// existResult.LastUpdateDate = obs.LastUpdateDate;

// existResult.DeleteFlag = false;

// if (!existResult.ObservationDate.HasValue && obs.ObservationDate.HasValue)

// existResult.ObservationDate = obs.ObservationDate;

// if (!existResult.PreliminaryDate.HasValue && obs.PreliminaryDate.HasValue)

// existResult.PreliminaryDate = obs.PreliminaryDate;

// if (!existResult.AuditDate.HasValue && obs.AuditDate.HasValue)

// existResult.AuditDate = obs.AuditDate;

// if (existResult.ObservationID == "GDT")

// {

// existResult.ValueTitle = "影像所见";

// }

// else if (existResult.ObservationID == "IMP")

// {

// existResult.ValueTitle = "影像诊断";

// }

// if (!string.IsNullOrEmpty(existResult.ResultStatusCode))

// {

// existResult.ResultStatusCode = existResult.ResultStatusCode;

// existResult.ResultStatus = ExamStatus.GetExamResultStatus(existResult.ResultStatusCode);

// }

// else if (!string.IsNullOrEmpty(existResult.ResultStatus))

// {

// existResult.ResultStatusCode =

// ExamStatus.GetExamResultStatusCode(existResult.ResultStatus);

// existResult.ResultStatus = existResult.ResultStatus;

// }

// if (string.IsNullOrEmpty(existResult.ResultStatus))

// {

// existResult.ResultStatus = obs.ResultStatus;

// existResult.ResultStatusCode = obs.ResultStatusCode;

// }

// if (!string.IsNullOrWhiteSpace(existResult.ResultStatus))

// {

// existResult.ResultStatus = existResult.ResultStatus.Trim();

// }

// if (!string.IsNullOrWhiteSpace(existResult.ResultStatusCode))

// {

// existResult.ResultStatusCode = existResult.ResultStatusCode.Trim();

// }

// // 测试

// //var n = existResult.ResultSeq;

// //

// \_observationResultRepository.Update(existResult);

// //测试

// //result.Result.Add(n);

// result.Result.Add(existResult.ResultSeq);

// result.msg = "更新成功";

// }

// // 把当前的这条数据插入

// else

// {

// var obsResult = new ObservationResult();

// obsResult = Mapper.Map(observationReuslt, obsResult);

// if (obsResult.ObservationID == "GDT")

// {

// obsResult.ValueTitle = "影像所见";

// }

// else if (obsResult.ObservationID == "IMP")

// {

// obsResult.ValueTitle = "影像诊断";

// }

// obsResult.ObservationUID = obs.ObservationUID;

// obsResult.DeleteFlag = false;

// obsResult.LastUpdateDate = obs.LastUpdateDate;

// if (observationReuslt.ObservationDate == null && obs.ObservationDate != null)

// obsResult.ObservationDate = obs.ObservationDate;

// if (observationReuslt.PreliminaryDate == null && obs.PreliminaryDate != null)

// obsResult.PreliminaryDate = obs.PreliminaryDate;

// if (observationReuslt.AuditDate == null && obs.AuditDate != null)

// obsResult.AuditDate = obs.AuditDate;

// if (!string.IsNullOrEmpty(obsResult.ResultStatusCode))

// {

// // obsResult.ResultStatusCode = obsResult.ResultStatusCode;

// obsResult.ResultStatus = ExamStatus.GetExamResultStatus(obsResult.ResultStatusCode);

// }

// else if (!string.IsNullOrEmpty(obsResult.ResultStatus))

// {

// obsResult.ResultStatusCode = ExamStatus.GetExamResultStatusCode(obsResult.ResultStatus);

// obsResult.ResultStatus = obsResult.ResultStatus;

// }

// if (string.IsNullOrEmpty(obsResult.ResultStatus))

// {

// obsResult.ResultStatus = obs.ResultStatus;

// obsResult.ResultStatusCode = obs.ResultStatusCode;

// }

// if (!string.IsNullOrWhiteSpace(obsResult.ResultStatus))

// {

// obsResult.ResultStatus = obsResult.ResultStatus.Trim();

// }

// if (!string.IsNullOrWhiteSpace(obsResult.ResultStatusCode))

// {

// obsResult.ResultStatusCode = obsResult.ResultStatusCode.Trim();

// }

// \_observationResultRepository.Insert(obsResult);

// //测试

// var n = obsResult.ResultSeq;

// result.Result.Add(n);

// result.msg = "数据insert成功";

// //result.Result.Add(existResult.ResultSeq);

// }

// }

//}

5.

6.

7.

8.

存在问题：

1.

采集的时候，比方ObservationRequest，采集的时候，是否更新的准确定位，判别的是PatientID，分配机构，主索引，但是有重复，比方恰好是同一个医院同一个科室同一个病人的第二次检查，那就会覆盖，所以还要根据其他判定条件去精确寻找，在get的方法中再加判断而增加查询query中的参数。

2.

3.

要问的：

（1）按照这个规范开发，那些个原来的拼接，不要了？

（2）每次操作，比方患者注册，是否要将操作写入临床数据操作日志，还待确定

（3）Visit就诊表的上传，似乎没有这样的接口

（4）注册用户的时候允不允许第三方去规定GUID，不行吧，因为可以nvarchar，是可变的。到时候参数不对了。

（5）更新的时候，覆盖了许多内容，使之为null 了。（如果入参是正确的，没有这个问题，但如果，用户就是想去更改，我觉得把，还是单独开一个Update的接口吧）

（6）所有的入参都是string类型，要不要改，改了，肯定造成之前的不兼容

问了简老后的结果：

（1）第三个接口，约束了每次同步必须传的是同一个ObservationUID下的结果集。

（2）第三个接口，以后还会有检验的信息，不仅仅是现在的影像所见，还有影像诊断。

（3）约定接口的调用顺序。

1.更新所存在的Bug问题，以什么条件去筛选实体，现在是根据PatientID，分配机构，主索引，三个确定唯一性，可如果我要改的就是这其中的，比方PatientID，我填错了，现在我想改，能改吗？

2.唯一性的问题，同一个科室，两个检查，这些PatientID，还有分配机构号，会不会变呢？

3.病人信息表，是物理删除。

返回列表，之前的入参：

{

"organizationID":"47000326-5,2470,42490819400,321084100000",

"resultDateEnd":"/Date(1545407999000+0800)/",

"resultDateStart":"/Date(1540321600000+0800)/",2018/12/18 10:02:42

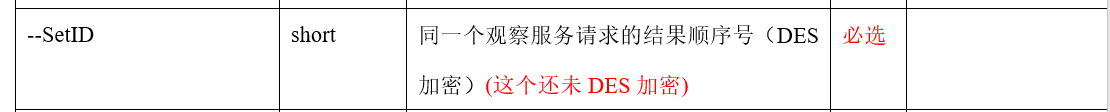
"PageIndex":1,

"PageSize":10

}

private const string DefaultKey = "t0MT@w~1";

t0MT@w~1



逻辑处理：

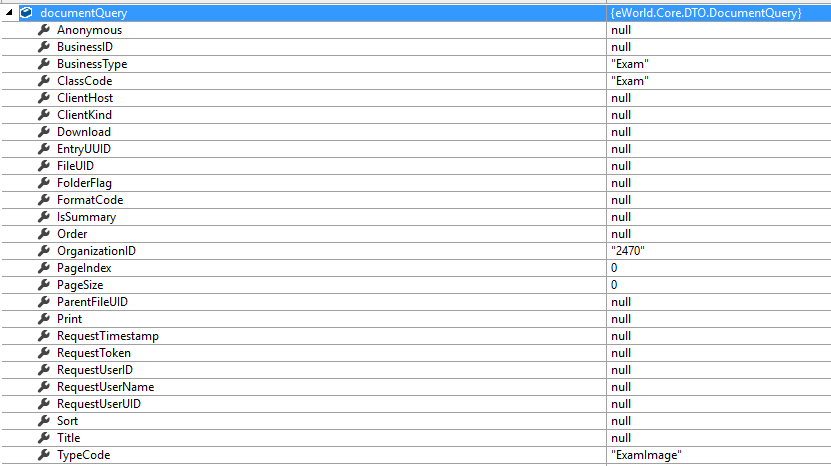
1.

1.3.2.3. 获取影像浏览地址

http://ip:port/v2/public/get\_exam\_imageurl

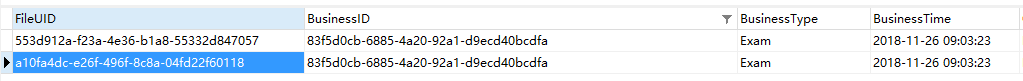
进入到组织查询-文档的方法：

var docmentInfo = QueryDocument(docmentQuery).FirstOrDefault();



最终是按上传时间排序

query = query.OrderBy(d => d.UploadTime);





但前面是取第一个，

var docmentInfo = QueryDocument(docmentQuery).FirstOrDefault();

而且传递参数的时候，

var docmentQuery = new DocumentQuery

{

BusinessID = query.ObservationUID,

OrganizationID = query.OrganizationID,

BusinessType = businessType,

ClassCode = classCode,

TypeCode = typeCode

};

这里没有传递很多的，可以精确到某条记录的查询参数，所以如果数据多条的话，如果只每次传ObservationUID，因为每条数据对应的都是该ObservationUID，每次拿到的都是最新的那条数据，所以数据是错的。

2.

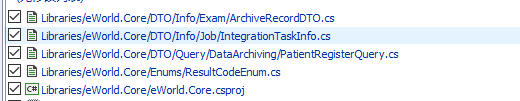
最终的svn提交

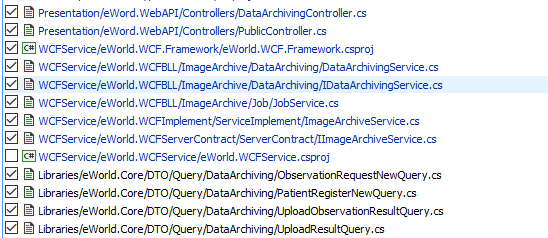
1.修改SaveIntegrationTask bug。

2.温岭市影像云增加患者注册，观察申请上传，观察结果上传接口。

3.增加参数验证类ValidateParaExtend。

采集任务，还有web首页返回列表的查询





验证参数



3.

4.