記事
Shintaro Kaminaka · 2021年10月18日 14m read

FHIRリポジトリをカスタマイズしよう！パート2（カスタムOperation編）

特報！！

この記事の中でご紹介している、FHIRオペレーションのデモを含め、FHIR PathやFHIRプロファイル対応などの2021.1のFHIR関連新機能をご紹介するウェビナーが、**2021年10月21日** **12:30~13:00** に開催されます！！ご興味ある方はこちらからご登録ください！！

InterSystems IRIS 開発者向けウェビナーシリーズ

開発者の皆さん、こんにちは。
今日は前回の FHIRリポジトリをカスタマイズしよう！パート１ の記事に続き、パート2として、カスタムオペレーションの実装方法をご紹介したいと思います。この記事で紹介している内容のFHIRリポジトリカスタムオペレーションに関するドキュメントマニュアルは こちら になります。
この記事はIRIS for Health 2021.1 をベースに記載しています。バージョンによって実装方法が異なるケースも考えられますので、該当のバージョンのドキュメントをご参照ください。場合によっては新しいバージョンへアップグレードもご検討ください。

FHIRのOperation（オペレーション）とは？

まず、FHIRのOperationについて、簡単にご紹介したいと思います。
HL7 FHIR公式ページの Operationのページ には以下のように記載されています。

> The RESTful API defines a set of common interactions (read, update, search, etc.) performed on a repository of typed resources. These interactions follow the RESTful paradigm of managing state by Create/Read/Update/Delete actions on a set of identified resources. **While this approach solves many use cases, there is some functionality that can be met more efficiently using an RPC-like paradigm, where named operations are performed with inputs and outputs (Execute).**

FHIRにおけるデータのアクセスがRESTのCRUDを基とするのはご存知の通りですが、RPC的なアプローチでより効率的なデータアクセス、データ処理を実現しようというアプローチがFHIRのOperationと言えます。

HL7 FHIR公式ページで規定されているOpeartionは こちら一覧のページ に記載されています。
代表的な所では、Patientリソースで指定できる *$everything* オペレーションや、Observationリソースで指定できる *$lastn* オペレーションがありますので紹介します。

$everything オペレーション

$everything
オペレーションはPatientリソースと組み合わせて使用します。詳細は こちらのページ を参照してください。
IRIS for Health のFHIRリポジトリの場合は、リソースの論理IDまで指定して

GET /Patient/5/$everything

のように指定して実行できます。
この場合、Patientリソースの論理ID=5に紐づいた関連するリソース(Observation, MedicationRequest, Procedure, Encounter など)を自動的に集めてBundle形式でクライアントに返してくれます。
例えばある患者さんの診療情報を一覧して見せたい場合などには便利な機能です。

$lastn オペレーション

$lastn
オペレーションはObservationリソースに組み合わせ使用します。詳細は こちらのページ を参照してください。
IRIS for Health のFHIRリポジトリの場合は以下のように指定できます。categoryおよびpatient(またはsubject)は必須指定パラメータになっています。

```
GET /Observation/$lastn?max=10&category=vital-signs&patient=Patient/123
```

このクエリでは、PatientリソースのID=123の患者に関連した、「vital-signs」カテゴリーに含まれる、最新 **10件** のObservationリソースがBundle形式でクライアントに返されます。直近のデータだけを使ってグラフを表示したい、というようなニーズにはぴったりですね。

カスタムオペレーションを作成てみよう

FHIRのオペレーションとはどのようなものか？具例を2つ挙げてご紹介しました。
カスタムオペレーションはこのようなオペレーションを、プロジェクトのニーズに応じて、自分で定義して実装し利用することができるようになる機能です。定義したカスタムオペレーションはCapability Statementに含めて公開することができます。
以下の手順でカスタムオペレーションを構築することができます。

1. 3つのカスタムクラス(Interactions等)を用意する

まず、最初の手順として、前回の FHIRリポジトリをカスタマイズしよう！パート１ で記載した、3つのカスタムクラスを用意します。詳しい内容はパート1の記事をご覧ください。

2. カスタムオペレーション用のクラスを用意し、Interactionsクラスから参照する

Interactionsクラス等と同様に、カスタムオペレーション用のクラスを継承します。
HS.FHIRServer.Storage.BuiltInOperationsを継承、任意の名称のクラスを作成します。この記事ではCustomFS.MyOperationとします。

（場合によっては、HS.FHIRServer.API.OperationHandlerクラスを継承て作成することもありますが、FHIRリポジトリを利用している場合は、HS.FHIRServer.Storage.BuiltInOperationsを継承、$everythingなどの実装済みのオペレーションが引き続き使用できるようにする必要があります。詳細はドキュメントをご覧ください。）

これでFHIRリポジトリのカスタマイズに関連して作成たクラスは4つになりましたね。

ベースクラス	継承て作成たクラス
HS.FHIRServer.Storage.Json.Interactions	CustomFS.MyInteractions
HS.FHIRServer.Storage.Json.InteractionsStrategy	CustomFS.MyInteractionsStrategy
HS.FHIRServer.Storage.Json.RepoManager	CustomFS.MyRepoManager
(new!) **HS.FHIRServer.Storage.BuiltInOperations**	**CustomFS.MyOperation**

さらに、このFHIRリポジトリのサーバ処理でこのカスタムオペレーション用のクラスが使用されるように、Interactionsクラスの OperationHandlerClass パラメータでこのクラスを指定します。

```
Class CustomFS.MyInteractions Extends HS.FHIRServer.Storage.Json.Interactions
{

 Parameter OperationHandlerClass As %String = "CustomFS.MyOperations";
```

3. カスタムオペレーションの処理を実装する

いよいよ、具体的なカスタムオペレーションの処理内容を実装していきます。2.で作成した、CustomFS.MyOperationsクラス内にメソッドを作成して実装します。

まず作成するオペレーションは影響範囲の応じて3つに分けることができます。
この3つの影響範囲=Scope ＋ オペレーション名により、作成するべきメソッド名が決まってきます。

メソッド名命名ルールについて

メソッド名命名には以下のようなルールがあります。

FHIR*Scope*Op*OperationName*

まずメソッド名の先頭は **FHIR** です。

次に *Scope* には以下の3つのタイプのいずれかが入ります。

Scope	説明
System	"ベース" の FHIR エンドポイントに追加する操作を指定します (例えば、 http://fhirserver.org/fhir)。これらの操作は、サーバ全体に適用されます。
Type	FHIR エンドポイントにリソースタイプ共に追加する操作を指定します (例えば、 http://fhirserver.org/fhir/Patient)。これらの操作は、指定されたリソースタイプのすべてのインスタンスで動作します。
Instance	リソースの特定のインスタンスを指す FHIR エンドポイントに追加する操作を指定します (例えば、 http://fhirserver.org/fhir/Patient/1)。これらの操作は、リソースの特定のインスタンスでのみ動作します。

以下の図も影響範囲の理解の参考になると思います。

そして、"Op" に続き、先頭を大文字にしたオペレーション名が続きます。

例えば、「$deleteall」というカスタムオペレーションをSystemレベルで作成したいなら

FHIRSystemOpDeleteall

というメソッドを作成ます。

「$anonymize」というカスタムオペレーションをInstanceレベルで作成たいなら

FHIRInstanceOpAnonymize

というメソッドを作成ます。

つまり　**"FHIR" _ (System or Type or Instance) _ "Op" _ (先頭大文字にしたオペレーション名)**
という命名ルールですね。

メソッドの実装方法

では、実際に、Instanceレベルの$anonymizeオペレーションを作成ていきましょう。このオペレーションの目的はPatientリソースのnameエレメントの中身を匿名化(*******で埋める)を実装することです。

このメソッドは

GET /Patient/5/$anonymize

のような形で実装されることを想定しています。

上記の例でてきたように、FHIRInstanceOpAnonymizeメソッドを以下のように実装します。サンプルなので詳細なエラーハンドリングまでは実装していません。

```
ClassMethod FHIRInstanceOpAnonymize(pService As HS.FHIRServer.API.Service, pRequest A
s HS.FHIRServer.API.Data.Request, pResponse As HS.FHIRServer.API.Data.Response)
{
    ///RestClient????????FHIR??????????????????????ID?Patient?????????
    Set clientObj = ##class(HS.FHIRServer.RestClient.FHIRService).CreateInstance(pReq
uest.SessionApplication)
    Do clientObj.SetResponseFormat("JSON")
    set clientResponseObj=clientObj.Read(pRequest.RequestMethod,pRequest.Type,pReques
t.Id)

    set pResponse.Json=clientResponseObj.Json
    set pResponse.Status=clientResponseObj.Status
    set pResponse.ETag=clientResponseObj.ETag
    set pResponse.LastModified=clientResponseObj.LastModified
    set pResponse.Location=clientResponseObj.Location
    set pResponse.IsPrettyOut=clientResponseObj.IsPrettyOut


    //??????????
    if pResponse.Status="200" {
        //DynamicObject??name??????Iterator??????????
        set iter=pResponse.Json.name.%GetIterator()
        while iter.%GetNext(.key,.value) {
            do pResponse.Json.name.%Get(key).%Set("text","***********")
            do pResponse.Json.name.%Get(key).%Set("family","***********")
            do pResponse.Json.name.%Get(key).%Set("given","***********")
        }
```

```
    }
}
```

パート1のカスタマイズ処理では、実際に検索(GET)されたデータや、送信(POST/PUT)されたデータに対してカスタム処理を実施しましたが、オペレーションの場合はベースとなる検索を実行する必要があります。もちろん、オペレーションの実装目的によっては、検索をする必要がない場合もあるでしょう。

このメソッドでは、得られた検索結果に対して、DynamicObjectのデータ操作を使用して、データを更新し匿名化を行っています。

4. 作成したカスタムオペレーションの定義をCapability Statementに追加する

次に、ロジックを作成したカスタムオペレーションの定義をCapability Statementに追加します。まず、先ほど同じCustomFS.MyOperationクラスにCapability Statement追加用のAddSupportedOperationsメソッドを記述します。

```
ClassMethod AddSupportedOperations(pMap As %DynamicObject)
{
    Do ##super(pMap)
    Do pMap.%Set("anonymize","http://myfhirserver/fhir/OperationDefinition/patient-
anonymize")
}
```

メソッド内の、pMap.%Setの最初の引数は"オペレーション名"、2番目の引数はそのオペレーションの定義を表すURIを記載します。
2番目の引数は例えば、先述のPatient/[id]/$everything オペレーションであれば、 こちらのページ に記載されているように

http://hl7.org/fhir/OperationDefinition/Patient-everything

となります。例えば、あるデータ連携プロジェクトに基いて実装した場合は、そのプロジェクトの実装ガイド内で規定されたURIになるでしょうし、自プロジェクト内で利便性のために構築したいうことであれば、任意のURIを指定することもできます。

次にコマンドラインユーティリティを起動して、この変更を反映します。
以下はCUSTOMFSネームスペース内で実行した例になります。

```
USER>zn "customfs"

CUSTOMFS>d ##Class(HS.FHIRServer.ConsoleSetup).Setup()
Query returns no results
HS.FHIRServer.Installer:InstallNamespace Created FHIR web application
HS.FHIRServer.Installer:InstallNamespace Created FHIR API web application
What do you want to do?
  0)  Quit
  1)  Create a FHIRServer Endpoint
  2)  Add a profile package to an endpoint
  3)  Display a FHIRServer Endpoint Configuration
  4)  Configure a FHIRServer Endpoint
  5)  Decommission a FHIRServer Endpoint
```

```
  6)  Delete a FHIRServer Endpoint
  7)  Update the CapabilityStatement Resource
  8)  Index new SearchParameters for an Endpoint
  9)  Upload a FHIR metadata package
  10) Delete a FHIR metadata package
Choose your Option[1] (0-10): 7

For which Endpoint do you want to update the CapabilityStatement?
  1) /csp/healthshare/customfs/fhir/r4 [enabled] (for Strategy 'CustomFS' and Metadat
a Set 'hl7.fhir.r4.core@4.0.1')
Choose the Endpoint[1] (1-1): 1

Update the /csp/healthshare/customfs/fhir/r4 service CapabilityStatement to reflect t
he endpoint strategy. Proceed? (y/n): yes
```

以上の手順を実行すると、FHIRリポジトリのCapability Statementにカスタムオペレーションのエントリが追加されます！
Capability StatementはFHIRリポジトリに以下のリクエストをすると取得できます。anonymizeオペレーションがちゃんと追加されていますね。

```
 GET http://fhirserver/csp/healthshare/customfs/fhir/r4/metadata
```

```
"operation": [
{
    "name": "everything",
    "definition": "http://hl7.org/fhir/OperationDefinition/Patient-everything"
},
(?)
{
    "name": "anonymize",
    "definition": "http://myfhirserver/fhir/OperationDefinition/patient-anonymize"
},
```

5. 実行して動作を確認してみよう

以上で、カスタムオペレーション $anoymize の実装は完了です。早速動作確認してみましょう！

まずカスタムオペレーションなしで普通にPatientリソースのデータをリクエストしてみます。

```
 GET http://localhost:52785/csp/healthshare/customfs/fhir/r4/Patient/2
```

私のデモ環境では、以下のようなnameエレメントを持つデータが格納されていました。

```
{
    "resourceType": "Patient",
    "id": "2",
    (?)
    "name": [
        {
            "extension": [
```

```
                {
                    "url": "http://hl7.org/fhir/StructureDefinition/iso21090-EN-
representation",
                    "valueCode": "IDE"
                }
            ],
            "use": "official",
            "text": "?? ??",
            "family": "??",
            "given": [
                "??"
            ]
        },
        {
            "extension": [
                {
                    "url": "http://hl7.org/fhir/StructureDefinition/iso21090-EN-
representation",
                    "valueCode": "SYL"
                }
            ],
            "use": "official",
            "text": "??? ???",
            "family": "???",
            "given": [
                "???"
            ]
        }
```

次に、実装した $anonymizeを追加して実行してみます!

```
 GET http://localhost:52785/csp/healthshare/customfs/fhir/r4/Patient/2/$anonymize
```

以下のように匿名化されたデータが取得できました!

```
{
    "resourceType": "Patient",
    "id": "2",
    (?)
    "name": [
        {
            "extension": [
                {
                    "url": "http://hl7.org/fhir/StructureDefinition/iso21090-EN-
representation",
                    "valueCode": "IDE"
                }
            ],
            "use": "official",
            "text": "***********",
            "family": "***********",
            "given": "***********"
        },
        {
            "extension": [
```

```
                {
                    "url": "http://hl7.org/fhir/StructureDefinition/iso21090-EN-
representation",
                    "valueCode": "SYL"
                }
            ],
            "use": "official",
            "text": "***********",
            "family": "***********",
            "given": "***********"
        }
    ],
```

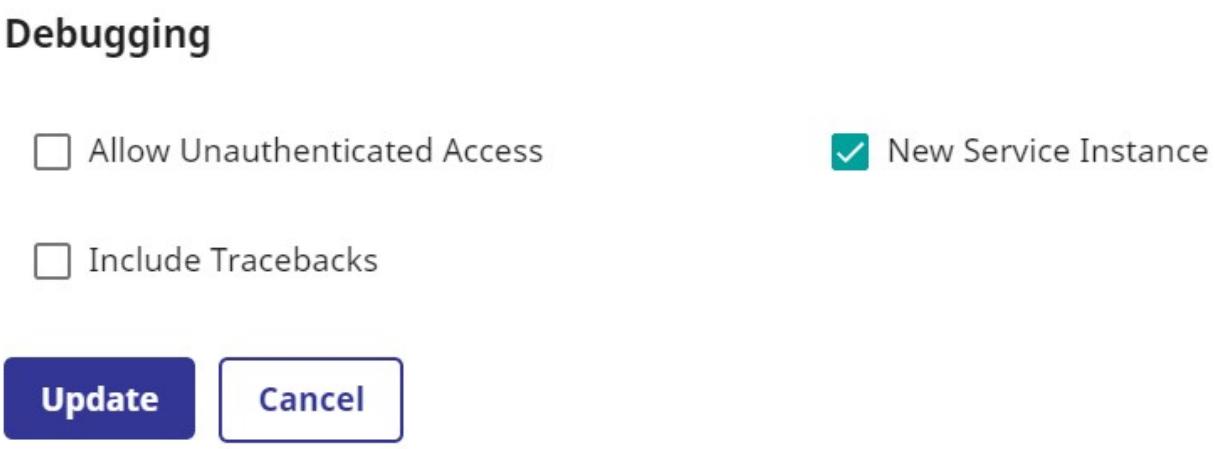
カスタマイズを実装する際のTips

パート1、パート2ではIRIS for HealthのFHIRリポジトリをカスタマイズする方法をご紹介してきました。
ここで、カスタマイズの際のTipsを一つご紹介します。

IRISのFHIRリポジトリ処理では、パフォーマンス最適化のために、個別のプロセス内でFHIR処理用のサービスインスタンスが起動され、再利用される仕組みになっています。そのため、条件によっては、サーバ側のカスタム処理を書き換えたのにそれが反映されないことがあります。そのような際にはいくつかの対応方法があります。

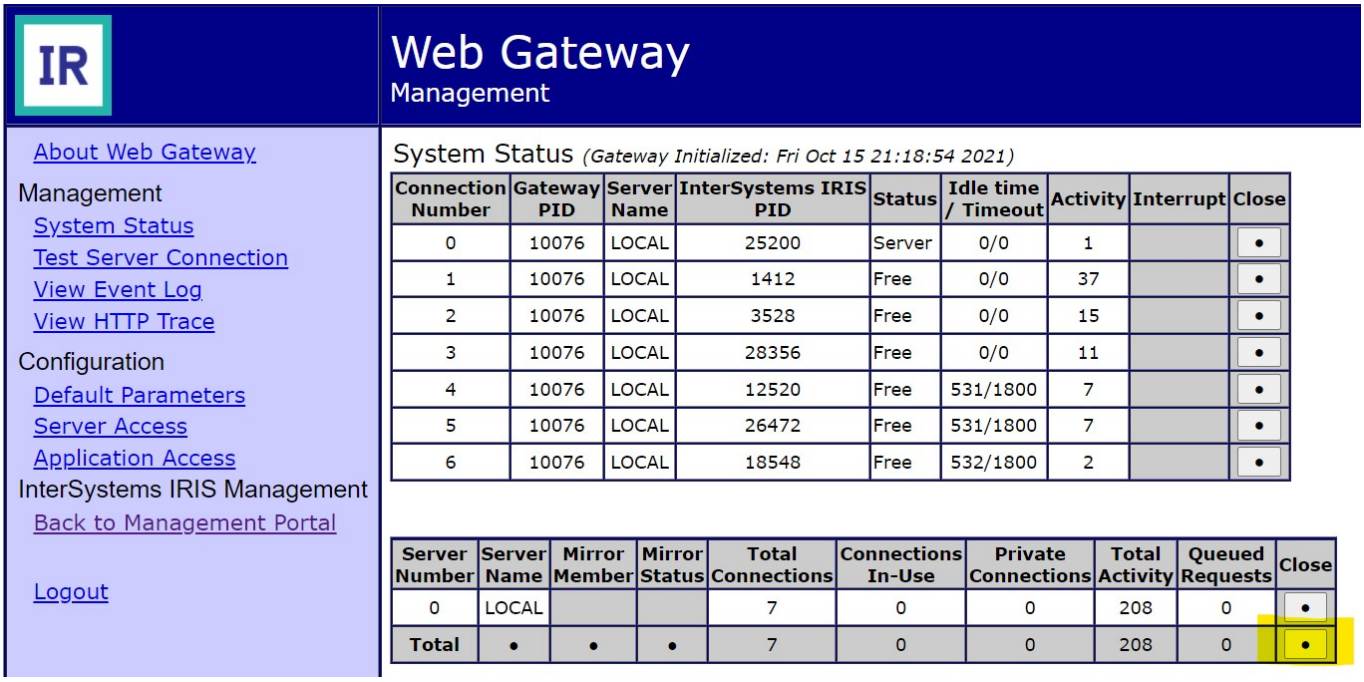
1. FHIR Server Configurationページで「New Service Instance」のチェックを有効にする

開発環境であれば、この方法をお勧めします。FHIRリクエスト毎に新しいService Instanceが立ち上がるので、必ず変更が反映されます。ただし、パフォーマンスへの影響を考えると、本番環境で設定することはお勧めしません
。



2. Web Gateway Management 画面からすべてのセッションをクリアする

System Status画面ですべてのセッションをクリアすることによって、起動しているIRIS側のプロセスもすべて再起動されます。下記画像の黄色の部分をクリックします。



3. 再起動する(常に最強の手段)

まとめ

いかがでしたでしょうか?

IRIS for HealthのFHIRリポジトリカスタマイズ機能を利用することによって、プロジェクト開発で求められる様々な要件に柔軟に対応することができます。もちろん、カスタマイズを適用することが、FHIRの持つ汎用性に影響を及ぼすことを考慮する必要がありますが、パート1/2の記事でご紹介したカスタマイズ機能を効果的に使えば、より高度なFHIRアプリケーションの構築も可能になります。

パート1、パート2の内容を含むIRISのクラスを こちらのGitHubサイト からダウンロードすることができます。

また新たなカスタマイズ手法が実装されたらこちらの開発者コミュニティで紹介したいと思います。

#FHIR #InterSystems IRIS for Health

ソースURL: https://jp.community.intersystems.com/post/fhir%E3%83%AA%E3%83%9D%E3%82%B8%E3%83%88%E3%83%AA%E3%82%92%E3%82%AB%E3%82%B9%E3%82%BF%E3%83%9E%E3%82%A4%E3%82%BA%E3%81%97%E3%82%88%E3%81%86%EF%BC%81%E3%83%91%E3%83%BC%E3%83%882%EF%BC%88%E3%82%AB%E3%82%B9%E3%82%BF%E3%83%A0operation%E7%B7%A8%EF%BC%89