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

FHIRリポジトリをカスタマイズしよう！パート１

開発者の皆さん、こんにちは。
IRIS for Health 2021.1がリリースされてからしばらくたちますが、多くのユーザさんにFHIRリポジトリ機能をお使いいただいています。
今日はFHIRリポジトリのサーバ側の処理をカスタマイズする機能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リポジトリカスタマイズ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はカスタマイズに必要なクラスが異なることがあります（例えば2020.1では後述のRepoManagerクラスはまだありません。）

FHIRサーバカスタマイズの2つのアプローチ

FHIRサーバ機能をカスタマイズするには2つのアプローチがあり、１つはこの記事でご紹介する、Interactionクラス群をカスタマイズしてFHIRリポジトリを拡張する方法、もう一つは、相互運用プロダクションでロジックを実装してサーバとしての動作を変更する方法です。

後者は厳密にはFHIRリポジトリとしての実装はそのままで、リポジトリに受け渡される前、あるいはリポジトリから応答を受けた後に、そのデータを参照、変更してカスタマイズを実装する流れになります。
IRISのInteroperability機能(Ensemble)に慣れているかたはこちらのアプローチがわかりやすいかもしれません。しかし、すべてのFHIRリクエストがトレースされメッセージが保存されるため、データ容量やパフォーマンスには気を配る必要があります。
この方法にご興味がある方は、こちらの相互運用プロダクションのドキュメントマニュアルをご参照ください。

FHIRリポジトリを構築する前の作業

FHIRリポジトリカスタマイズで最も注意しなければいけない点、それはFHIRリポジトリを構築する際にカスタマイズができるように、この後記載する手順を実施しておかなければいけないという点です。つまり、デフォルト設定でFHIRリポジトリを構築し、しばらく運用してから、「こういうカスタマイズ処理を組み込みたいな」と思っても、簡単に追加することができないのです。（新しくカスタマイズ可能なリポジトリを構築し、データ移行するなどの方法は取れます）

この制限があるため、FHIRリポジトリ運用当初はカスタマイズ予定がなくても、下記の手順を実行しカスタマイズ機能を組み込んでおくという選択肢もあります。

それでは、そのカスタマイズ方法をご紹介します。
まず、以下の3つのクラスをそれぞれ継承してカスタムクラスを作成します。この記事では共通のパッケージとしてCustomFSとし、クラス名の先頭にはMyを追加しています。

ベースクラス	継承して作成したクラス
HS.FHIRServer.Storage.Json.Interactions	CustomFS.MyInteractions
HS.FHIRServer.Storage.Json.InteractionsStrategy	CustomFS.MyInteractionsStrategy
HS.FHIRServer.Storage.Json.RepoManager	CustomFS.MyRepoManager

継承したInteractionStrategyクラスとRepoManagerクラスはロジックを実装したりはしませんが、以下の設定を行う必要があります。

- 両クラスの **StrategyKey**パラメータに共通の一意の識別子(文字列ならなんでも）を設定する
- IntectionStrategyの**InteractionClass**で自分の作成したInteractionsクラスを指定する
- RepoManagerの**StrategyClass**で自分の作成したInteractionStrategyクラスを指定する

図で関連を表現するとこんな感じになります。

実際のカスタマイズ内容は継承したInteractionsクラスの各メソッドをオーバーライドして実装していくことになります。

カスタマイズのクイックスタート

以下の表はドキュメントマニュアルに記載されている内容そのままですが、転載します。
これらのメソッドをオーバライドしてロジックを記述し、カスタマイズを行います。

目標	HS.FHIRServer.Storage.Json.Interactions のサブクラスのアクション
特定のFHIR相互作用のカスタマイズ	相互作用に対応するメソッド(**Add,Read,Update**)をオーバーライドします。
すべての要求の処理	**OnBeforeRequest** をオーバーライドして、ユーザに透過的なロジックを実装します。FHIR クライアントに要求の処理が異なることを認識させる場合は、カスタムの FHIR 操作を作成します。
すべての要求の後処理	**OnAfterRequest** をオーバーライドして、ユーザに透過的なロジックを実装します。FHIR クライアントに要求の処理が異なることを認識させる場合は、カスタムの FHIR 操作を作成します。
Read 相互作用の結果の後処理	**PostProcessRead** をオーバーライドします (例)。
Search 相互作用の結果の後処理	**PostProcessSearch** をオーバーライドします (例)。
カスタムの FHIR 操作の追加	**OperationHandlerClass** パラメータをオーバーライドして、**HS.FHIRServer.Storage.BuiltInOperations** のサブクラスの名前を指定します。"カスタムの FHIR 操作" を参照してください。
バンドルの処理方法のカスタマイズ	**BatchHandlerClass** パラメータをオーバーライドして、カスタム・クラスの名前を指定します。既定のハンドラ・クラスは **HS.FHIRServer.DefaultBundleProcessor** です。
OAuth トークンの処理方法のカスタマイズ	**OAuth2TokenHandlerClass** パラメータをオーバーライドして、カスタム・クラスの名前を指定します。既定のハンドラ・クラスは *HS.FHIRServer.Util.OAuth2Token** です。

この表のリンク先ドキュメントマニュアル
にもいくつか、実装例が記載されていますが、この記事ではもう少しシンプルなパターンのサンプルを記載していきたいと思います。

OnAfterRequestメソッド

まずはOnAfterRequestメソッドを題材にどんな引数を受け取るのか見てみましょう。

```
Method OnAfterRequest(pFHIRService As HS.FHIRServer.API.Service, pFHIRRequest As HS.F
HIRServer.API.Data.Request, pFHIRResponse As HS.FHIRServer.API.Data.Response)
{
```

このメソッドは、3つの引数を受け取ります。主に使うのはpFHIRRequestとpFHIRResponseの二つです。
このメソッドは
On**After**
Requestメソッドなので、FHIRリポジトリ上での処理が完了した後に起動されるメソッドです。そのため、FHIRリポジトリへの要求であるpFHIRRequestと、これから要求元へ返すpFHIRResponseの二つを参照できます。

FHIRリポジトリへのJSONリクエストおよびレスポンスは

```
pFHIRRequest.Json
pFHIRResponse.Json
```

に含まれます。これはJsonの構造通りにデータアクセスできる
%DynamicObject型の変数なので、このようにアクセスできます。

```
pFHIRResponse.Json.resourceType //-> Bundle??
pFHIRResponse.Json.total //->Bundle??????Bundle???????10???
pFHIRResponse.Json.entry.%Get(0).resource.gender //->Bundle?????Patient??????????
```

ログなどの目的でこの内容を文字列あるいはストリームに保存しておきたい場合は、%ToJSON()メソッドを使って変換します。
他にもこれらのインスタンスは

プロパティ	含んでいるデータ
pFHIRResponse.Id	追加・更新されたリソースの論理ID
pFHIRResponse.VId	追加・更新されたリソースのバージョンID
pFHIRResponse.Status	処理のステータス

など様々な情報を含んでいます。詳細はHS.FHIRServer.API.DataとHS.FHIRServer.API.Data.Responseのクラスリファンレスドキュメントをご覧ください。

OnBeforeRequest/PostProcessReadメソッドを実装してみる

それでは具体的にOnBeforeRequestメソッドを使って実装してみましょう。

IRIS for Health のFHIRリポジトリでは、こちらの記事(IRIS for Health上でFHIRリポジトリ＋OAuth2認可サーバ/リソースサーバ構成を構築するパート3(OAuth2スコープ編))で紹介したOAuth2アクセストークンのScopeを使う方法で、このScopeのユーザにはこのリソースの参照権限だけ与える、こちらには更新権限を与える、といったアクセスコントロールが可能です。ただ、これはあくまでもリソース単位のコントロールになるので、「このリソースにこういうデータがあるときは登録させたくない」「このリソースにこういうデータが含まれているときは応答として返したくない」などの要望の実装はこのカスタマイズを使って実現することができます。

ではまず、Patientリソースの登録時に、電話番号情報を格納するtelecom要素の最初のデータに 401,403,400 で始まる電話番号が入っていたらエラーを返す、というロジックを実装してみたいと思います。最初の3桁はエラーとして返すHTTPエラーコードと紐づけています。

以下のようなコードになります。本来はチェックロジックなども含めるべきですが、サンプルのためシンプルなコードにしています。

```
Method OnBeforeRequest(pFHIRService As HS.FHIRServer.API.Service, pFHIRRequest As HS.
FHIRServer.API.Data.Request, pTimeout As %Integer)
{
    //POST or PUT?Patient??????
    if ((pFHIRRequest.RequestMethod="POST")||(pFHIRRequest.RequestMethod="PUT"))&&(pF
HIRRequest.RequestPath["Patient") {

        if $IsObject(pFHIRRequest.Json) {
            s tele=pFHIRRequest.Json.telecom.%Get(0).value

            if $Extract(tele,1,3)="401" $$$ThrowFHIR($$$HttpOnlyResponse(401))
            if $Extract(tele,1,3)="403" $$$ThrowFHIR($$$HttpOnlyResponse(403))
            if $Extract(tele,1,3)="400" $$$ThrowFHIR($$$GeneralError, "????????????????
????????", $$$OutcomeNotSupported(400))

        }

    }
```

エラーとして応答を返す場合は、このサンプルのように直接Throwすることができます。

```
$$$ThrowFHIR($$$HttpOnlyResponse(401))
```

次はPostProcessReadメソッドです。
先ほどは登録時
の電話番号をチェックして応答
を返しましたが、今回は返すデータをチェックして**40**
で始まっている場合は、マスクして応答するようにしたいと思います。

```
Method PostProcessRead(pResourceObject As %DynamicObject) As %Boolean
{
    //40???????????Patient?telecom????????
    if pResourceObject.resourceType="Patient" {
        if $Extract(pResourceObject.telecom.%Get(0).value,1,2)="40" {
            //return 0
            set pResourceObject.telecom.%Get(0).value="******"
        }
    }

    return 1
}

}
```

この方法ではあくまで、応答を返す際に値を上書きしているだけなので、リポジトリ上の値は変更していないことに注意してください。
また、このPostProcessReadは、リソースのIDまで指定して検索するようなケースにだけ有効です。

```
http://localhost:52785/csp/healthshare/customfs/fhir/r4/Patient/3
```

以下のように全件取得や条件取得の際は応答がBundleとして返されるので、この変換は適用されません。

```
http://localhost:52785/csp/healthshare/customfs/fhir/r4/Patient
http://localhost:52785/csp/healthshare/customfs/fhir/r4/Patient?gender=female
```

Bundleに含まれる応答をカスタマイズする場合には **OnProcessSearch** をカスタイズします。

まとめ

いかがでしたでしょうか？
実際のFHIRアプリケーション開発においては、標準のFHIR仕様の範囲だけで実装することが難しい場合は、今回ご紹介したカスタマイズ機能を適切に使用して、開発に役立てることができます。
次回は、カスタマイズ機能パート２として、カスタムオペレーションの実装例をご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%88%EF%BC%91