Artigo
Emily Scoparo · Set. 5 7min de leitura

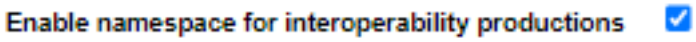
Usando a interoperabilidade para acessar Banco de Dados externo

Neste artigo, estarei ensinando como desenvolver uma *production* que acessa e retorna dados de um banco de dados externos.

Estarei usando o MySQL neste exemplo, porém a forma de conexão será a mesma para os demais bancos.

Farei a comunicação via REST e estarei utilizando o Postman para testar a produção de interoperabilidade.

1. INTRODUÇÃO:

- O primeiro passo é criar um namespace que seja habilitado para interoperabilidade, ou seja, que tenha esta caixinha selecionada: 

Neste namespace, iremos desenvolver as classes que vão implementar nossa production;
- Irei dividir o desenvolvimento em algumas partes:
  - Desenvolvendo o Business Operation;
  - Desenvolvendo o Business Service;
  - Criando o aplicativo Web;
  - Criando a Production;

2. DESENVOLVENDO O BUSINESS OPERATION:

- O Business Operation (BP) é a parte responsável pela conexão com os sistemas externos, no nosso caso, o MySQL;
- O BP é uma classe que estende a Ens.BusinessOperation. Ela chama os métodos que realizam as ações necessárias nos sistemas externos utilizando os adaptadores;
- Nossa classe vai ficar assim:

```
Class production.BO.Operation Extends Ens.BusinessOperation
{

Parameter ADAPTER = "EnsLib.SQL.OutboundAdapter";

Property Adapter As EnsLib.SQL.OutboundAdapter;

Parameter INVOCATION = "Queue";

Method Exibir(pRequest As production.BO.ObterObjeto.Request, Output pResponse As production.BO.ObterObjeto.Response) As %Statu
 {
     Quit ##class(production.BO.ObterObjeto.Method).Execute(##this,pRequest,.pResponse)
 }

XData MessageMap
 {
<MapItems>
     <MapItem MessageType="production.BO.ObterObjeto.Request">
          <Method>Exibir</Method>
     </MapItem>
 </MapItems>
 }
}
```

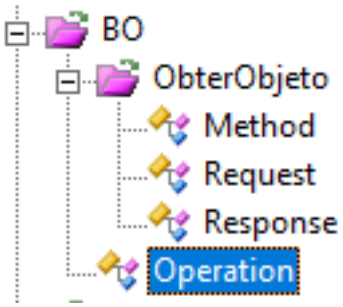
  - Ela sempre deve conter o Extends Ens.BusinessOperation;
  - O BO tem 3 classes para cada tipo de ação:
    - Request;
    - Response;
    - Method;
      - Cada ação tem uma requisição, faz as funções definidas na classe Method e devolve uma resposta, que iremos definir depois;
  - O parâmetro ADAPTER deve indicar qual adaptador iremos utilizar para a conexão com o sistema

externo (no nosso caso, um adaptador SQL);
- Lista de adaptadores do Iris:
  -> https://docs.intersystems.com/iris20221/csp/docbook/DocBook.UI.Page.cls?...

- Devemos também definir uma propriedade Adapter para utilizar os métodos da classe do adaptador;
- O parâmetros de invocação pode ser "Queue" ou "InProc";
- Abaixo definimos o XData MessageMap que indica qual método a classe Operation deve executar:
  - Você define qual vai ser a requisição;
  - No nosso exemplo, queremos somente obter os dados do Banco de dados, então definimos que:
    - Quando a classe Operation receber uma requisição de obter objeto, ela deve mandar para o método "Exibir", definido dentro da classe Operation;
- A classe BO será chamada por outras classes dentro da Production, como o BS ou o BP, por isso definimos os parâmetros que ela deve receber dessas outras classes:
  - No nosso caso, quando recebermos uma requisição para obter um objeto, o método Exibir será chamado;
  - O método exibir recebe uma requição e e devolve em Output uma resposta;
  - O método exibir retorna a execução do método Execute definido em Method de ObterObjeto (que recebe uma instância da classe operation, a requisição e devolve a resposta);

2.1. DESENVOLVENDO AS CLASSES PARA OBTER OS DADOS DO BANCO:

- Nosso BO deve ficar assim:



  - Cada ação contém 3 classes: Method, Request e Response. No nosso caso, só temos a ação ObterObjeto;
  - A classe Operation é responsável por chamar os métodos que realizam as ações (definimos no XData MessageMap);
- Desenvolvendo a classe Request:
  - Dentro da production, a comunicação é feita através de requisições e responstas;
  - As requisições são recebidas de outas classes, por isso não é importante saber de onde vem por agora, somente sua implementação;
  - Cada ação tem necessidades específicas, por isso montamos a classe de requisição seguindo as necessidades:

```
1  Class production.BO.ObterObjeto.Request Extends (%Persistent, Ens.Request)
2  {
3
4  Parameter RESPONSECLASSNAME = "production.BO.ObterObjeto.Response";
5
6  Property ID As %Integer;
7
```

  - Deve estender a Ens.Request;
  - Deve conter o parâmetro RESPONSECLASSNAME definido com o nome da classe resposta;
  - As outras propriedades dependem do que precisamos receber das demais classes para realizar o método da ação: no nosso caso, precisamos receber o ID do objeto no banco para realizarmos o SELECT, por isso temos a propriedade ID;
- Desenvolvendo a classe Response:
  - Toda requisição tem uma resposta;
  - Cada ação deve ter uma resposta específica, por isso desenvolvemos essa classe segundo nossas necessidades:

```
Class production.BO.ObterObjeto.Response Extends (%Persistent, Ens.Response, %JSON.Adaptor)
{

Property Resultado As production.PD.Objeto;

Property Erro As %Boolean;

Property ErroMessage As %String;

Method CopyToObject(Output object As %DynamicObject) As %Status
 {
     Do ..%JSONExportToString(.jsonString)
     Set object = ##class(%DynamicObject).%FromJSON(jsonString)
     ;Set object.ID = ..%Id()
     Return $$$OK
 }

Method CopyFromObject(object As %DynamicObject) As %Status
 {
     Do ..%JSONImport(object)
     Return $$$OK
 }
```

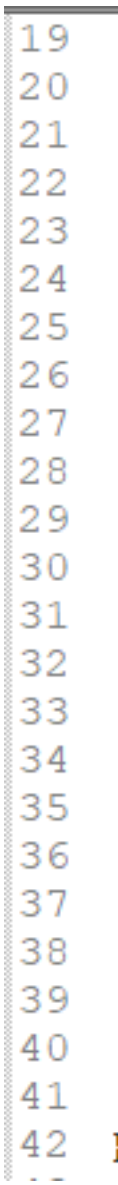
- No nosso caso, iremos retornar um objeto, por isso temos uma propriedade resultado que é um objeto:
  - Esse objeto deve conter as propriedades necessárias para armazenar os dados que queremos. Por exemplo, eu preciso armazenar os dados de um livro que irei resgatar do MySQL, então meu objeto vai ficar desse jeito:

```
Class production.PD.Objeto Extends (%Persistent, %XML.Adaptor, %JSON.Adaptor)
{

Property IDLivro As %Integer;

Property Titulo As %String;

Property Autor As %String;

Property Editora As %String;

Property Status As %Boolean;
```

  - Podeconter o %XML.Adaptor e o %JSON.Adaptor para podermos visualizar de outras formas;
- Coloquei as propriedades Erro e ErroMessage que receberão os erros que tiver na execução de Method;
- Os métodos abaixo são os que iremos utilizar para transformar as respostas em objetos dinâmicos para poder retorná-los;

- Desenvolvendo a classe Method:
  - A classe Method vai executar a ação que desejamos. No nosso caso, acessar e recuperar valores do Banco de Dados:

```
ClassMethod Execute(pHost As production.BO.Operation, pRequest As production.BO.ObterObjeto.Request, Output pResponse As produ
{

    Set tSc = $$$OK

    Try
    {
        Set tSc = pRequest.NewResponse(.pResponse)
        Throw:($$$ISERR(tSc))

        Kill pResultSet

        Set tSc = pHost.Adapter.ExecuteQuery(.pResultSet, "SELECT * FROM livros WHERE IDLivros = ?", pRequest.ID)
        Throw:($$$ISERR(tSc))
```



- No nosso caso, Method recebe a instância de Business Operation (para utilizarmos os métodos da sua propriedade Adapter), a requisição (para ter acesso ao ID e comparar no SELECT) e devolve Response;
- Utilizando a propriedade Adapter para realizamos o SELECT, instanciamos um objeto da classe que criamos para receber esses dados e colocamos ele na propriedad Resultado de Response;

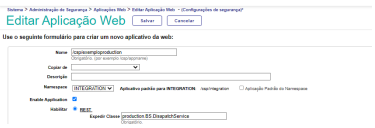
3 . DESENVOLVENDO O BUSINESS SERVICE:

- A classe Business Service (BS) é responsável por receber a requisição de um sistema que deseja acessar a Production (no nosso caso, via REST);
- A conexão também é feita via adaptadores;
- Nossa BS vai ficar assim:

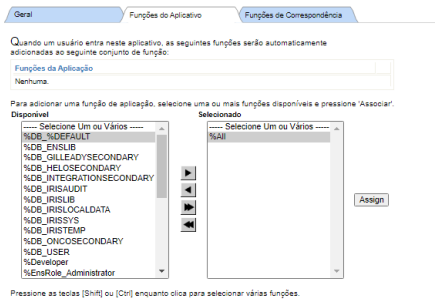
```
Class production.BS.DisapatchService Extends (%CSP.REST, Ens.BusinessService)
{

Parameter HandleCorsRequest = 0;

Parameter CHARSET = "utf-8";

Parameter CONTENTTYPE = "aSpplication/json";

XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
 {
<Routes>
      <Route Url="/objeto/:ID" Method="Get" Call="Exibir" />
 </Routes>
 }

ClassMethod Exibir(pID As %Integer) As %Status
 {
     Set tSc = $$$OK

     Try
     {
         Set tRequest = ##class(production.BO.ObterObjeto.Request).%New()
         Set tRequest.ID = pID
```

- As classes BS devem sempre estender a Ens.BusinessService;
- Como iremos utilizar REST, deve estender a %CSP.REST;
- As outras definições são todas do REST;
- No método exibir, iremos receber o ID do livro que desejamos buscar no banco e começar sua execução:
    - Primeiro instanciar um objeto de Requisição de ObterObjeto;
    - Adicionar o parâmetro em sua propriedade;
    - Depois chamamos o método GetService que definimos abaixo:
        - Ele cria um BS com o nome da classe que criamos e o retorna por referência em . tObjService;
    - Depois utilizamos o método SendRequestSync herdado por tObjService para enviar uma requisição síncrona (que espera a mensagem de resposta):
        - Colocamos o nome da classe da requisição como parâmetro (no nosso caso, a classe do BO), o objeto de Request que instanciamos e definimos que deve retornar o tResponse por referência;
    - Depois utilizamos o método que definimos lá em Response para copiar Response para um objeto dinâmico e retorná-lo por referência para tResponseDyn;
    - Depois instanciamos um objeto de %GlobalCharacterStream para poder exibir nosso resultado e utilizamos o método Write herdade por esse objeto:
        - Passamos o tResponseDyn.%ToJSON() como parâmetro para escrever o resultado em forma de JSON na tela;

4. CRIANDO O APLICATIVO WEB:

- Devemos definir o nome do aplicativo web e selecionar que ele será REST: 

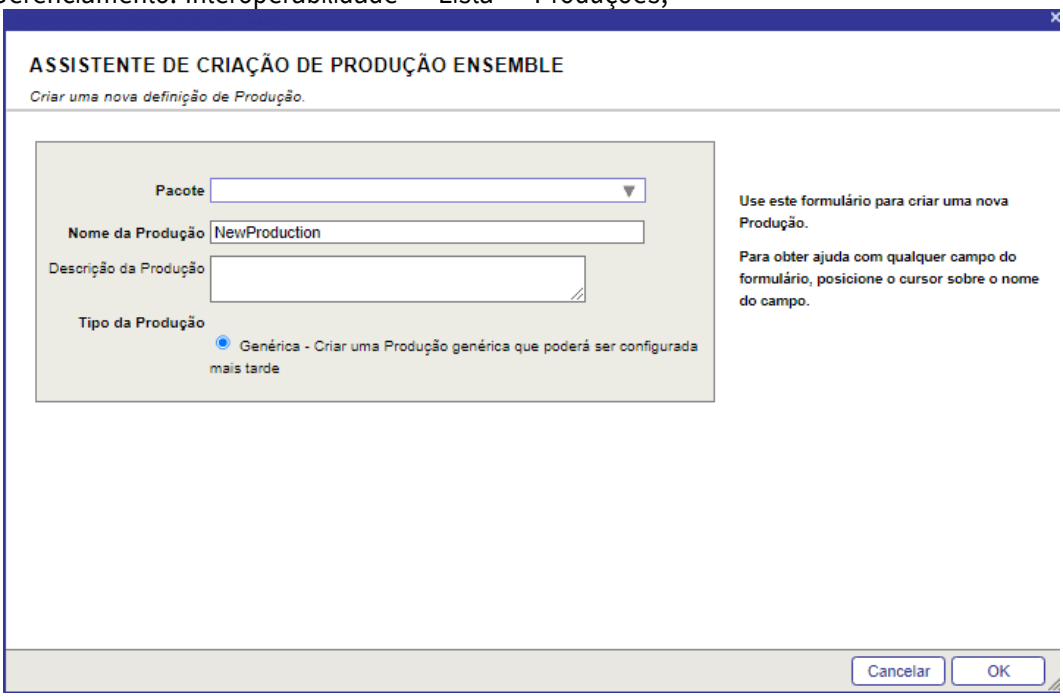
    - Em DispatchClass devemos colocar a classe BS;
    - Devemos ter certeza de que está no namespace onde desenvolvemos as classes;
    - Clicar em "Salvar";
    - Depois em "Funções do Aplicativo" e relacionar a função %All:



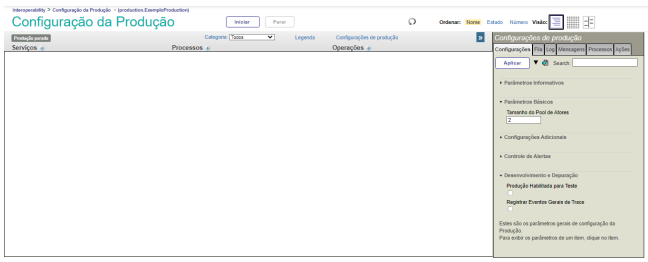
- Depois basta salvar e sair;

5. CRIANDO A PRODUCTION:

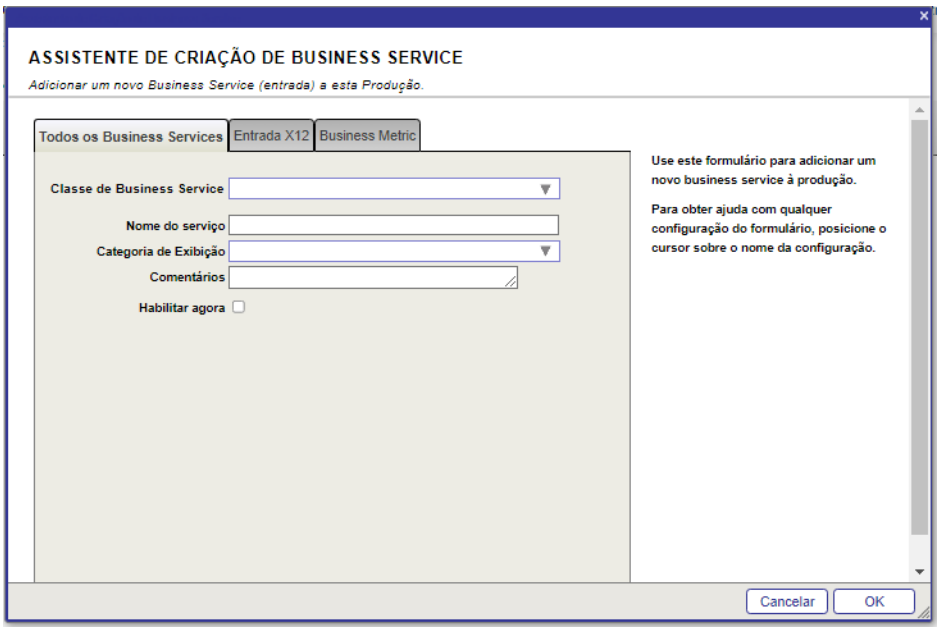
- Abrir o Portal de Gerenciamento: Interoperabilidade -> Lista -> Produções;
- Clicar em "Novo":



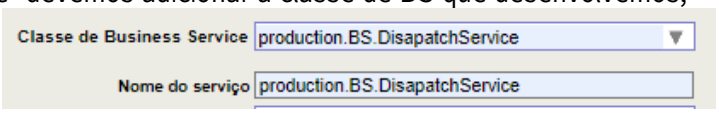
  - Selecione o pacote que deseja adicioná-la e depois o nome;
  - Depois clicar em "ok";
- Vamos adicionar o BS:
  - Nesta tela, clique no "+" ao lado de Serviços:



  - Vai abrir esta janela:

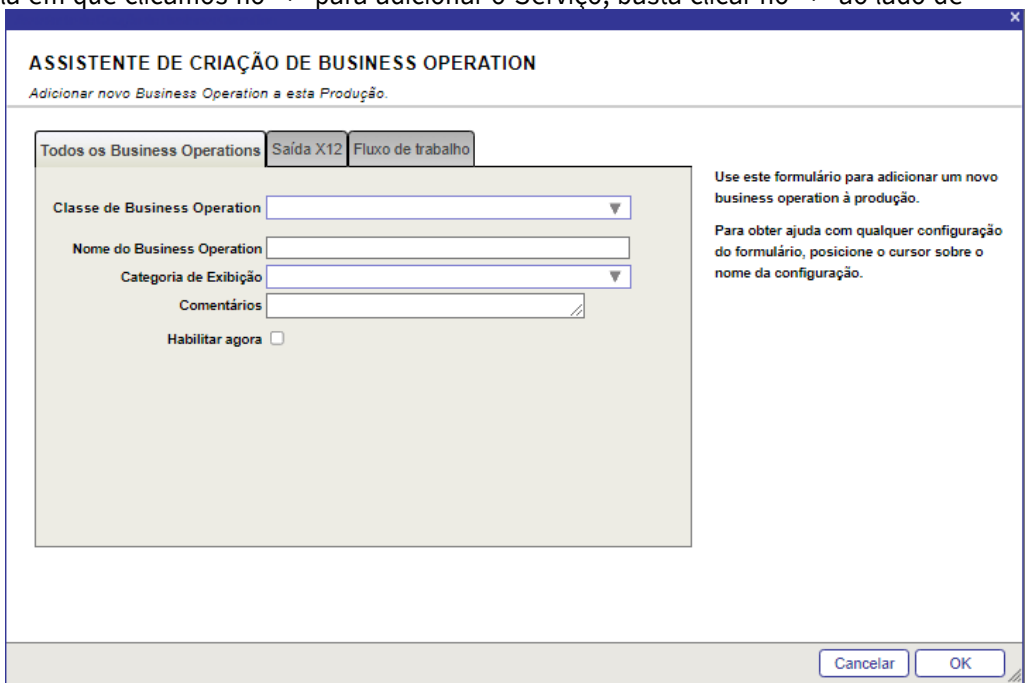


- Em "Classe de Business Service" devemos adicionar a classe de BS que desenvolvemos;
- Depois adicione o nome do BS:

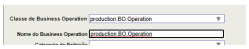


  - eu deixo ambos iguais, porém não é necessário;
- Depois clicar em "ok";

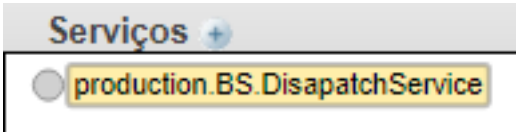
- Vamos adicionar o BO:
  - Na mesma tela em que clicamos no "+" para adicionar o Serviço, basta clicar no "+" ao lado de "Operações":



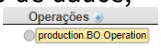
    - Nesta tela, é só adicionar a classe do BO e um nome para a Operação;



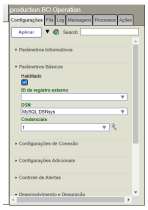
  - Depois é só clicar em "ok";
- Habilitando o BS:
  - Clique em cima do serviço para ele ficar selecionado:



  - Depois, na tela de configurações ao lado direito vá até "Parâmetros Básicos" e selecione a opção "habilitado":

    - Depois basta clicar em "Aplicar" no canto superior esquerdo da janela de configurações;
- Habilitando o BO:
    - Para habilitar o BO, devemos antes nos conectar ao banco de dados via DSN;
    - Depois de fazer a conexão, vamos ao início do Portal de Gerenciamento e vamos em Interoperabilidade -> configurar -> credenciais:
        - Clicamos em "Novo" e adicionamos as informações de conexão com seu banco de dados;
    - Depois, voltamos para a produção e clicamos em cima da operação para selecioná-la: 

    - Depois, na janela de configurações à direita, adicionamos o nome da nossa DSN e o ID da crecendial que adicionamos anteriormente. Depois, selecionamos a opção "Habilitado":



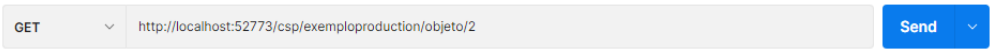
    - Basta clicar em "Aplicar";
- Agora basta iniciar a produção;

6. TESTE:

- Para testar, basta digitar a URL no postman junto com o ID (que utilizamos de parâmetro);
- Dados no MySQL:

	IDLivros	Titulo	Autor	Editora	Status
	2	Vidas Secas	Graciliano Ramos	Record	1
	3	Quarto de Despejo	Maria Carolina de Jesus	Ática	0
	4	Espectros	Cecília Meireles	Global	0
	5	A Hora da Estrela	Clarice Lispector	Rocco	1
*	NULL	NULL	NULL	NULL	NULL

- Retornos no Postman:



#InterSystems IRIS

URL de origem:https://pt.community.intersystems.com/post/usando-interoperabilidade-para-acessar-banco-de-dados-externo