Artigo
Danusa Calixto · jan 11, 2023 20min de leitura

Implementação do Open Authorization Framework do InterSystems IRIS (OAuth 2.0) - parte 3

Criado por Daniel Kutac, Engenheiro de vendas, InterSystems

Parte 3. Apêndice

Explicação sobre as classes OAUTH do InterSystems IRIS

Na parte anterior da nossa série, aprendemos a configurar o InterSystems IRIS para atuar como um cliente OAUTH, além de um servidor de autorização e autenticação (pelo OpenID Connect). Nesta parte final da série, vamos descrever classes que implementam o framework OAuth 2.0 do InterSystems IRIS. Também vamos discutir casos de uso para métodos selecionados de classes de API.

As classes de API que implementam o OAuth 2.0 podem ser separadas em três grupos diferentes de acordo com a finalidade. Todas as classes são implementadas no namespace %SYS. Algumas delas são públicas (por % pacote), outras não e não devem ser chamadas diretamente pelos desenvolvedores.

Classes internas

Estas classes pertencem ao pacote OAuth2.

A tabela a seguir lista algumas classes de interesse (para uma lista completa de classes, consulte a Referência de Classes online da sua instância do Caché). Nenhuma dessas classes deve ser usada diretamente por desenvolvedores de aplicativos, exceto as listadas abaixo.

Nome da classe	Descrição
OAuth2.AccessToken	PersistentOAuth2.AccessToken armazena um token de acesso do OAuth 2.0 e as informações relacionadas. É uma cópia do cliente OAUTH do token de acesso. OAuth2.AccessToken é indexado pela combinação de SessionId e ApplicationName. Portanto, apenas um escopo pode ser solicitado para cada SessionId/ApplicationName. Se uma segunda solicitação for feita com um escopo diferente e o token de acesso ainda não tiver sido concedido, o escopo na nova solicitação se tornará o escopo esperado.
OAuth2.Client	Persistente A classe OAuth2.Application descreve um cliente OAuth2 e faz referência ao servidor de autorização que usa para autorizar o aplicativo com base no RFC 6749. Um sistema cliente pode ser usado com vários servidores de autorização para diferentes aplicativos.
OAuth2.Response	Página CSP É a página de destino para respostas de um servidor de autorização do OAuth 2.0 usado de código do cliente OAuth 2.0 do InterSystems IRIS. A resposta é processada aqui e redirecionada ao alvo final.

OAuth2.ServerDefinition	Persistente Armazena informações do servidor de autorização usadas por um cliente OAUTH (essa instância do InterSystems IRIS). Podem ser definidas várias configurações de cliente para cada definição de servidor de autorização.
OAuth2.Server.AccessToken	Persistente Os tokens de acesso são gerenciados pelo OAuth2.Server.AccessToken no servidor OAUTH. A classe armazena o token de acesso e as propriedades relacionadas. Essa classe também é o meio de comunicação entre várias partes do servidor de autorização.
OAuth2.Server.Auth	Página CSP O servidor de autorização apoia o fluxo de controle de autorização para o código de autorização e os tipos de concessão implícitos conforme a especificação no RFC 6749. A classe OAuth2.Server.Auth é uma subclasse de %CSP.Page que atua como o endpoint de autorização e controla o fluxo de acordo com o RFC 6749.
OAuth2.Server.Client	Persistente OAuth2.Server.Configuration é uma classe persistente que descreve os clientes registrados com esse servidor de autorização.
OAuth2.Server.Configuration	Persistente Armazena a configuração do servidor de autorização. Todas as classes de configuração têm uma página correspondente no Portal de Gerenciamento de Sistemas onde os usuários preenchem os detalhes da configuração.

Objetos OAuth2.Client, OAuth2.ServerDefinition, OAuth2.Server.Client e OAuth2.Configuration podem ser abertos, modificados e salvos para criar ou modificar configurações sem usar a IU. Você pode usar essas classes para manipular configurações de maneira programática.

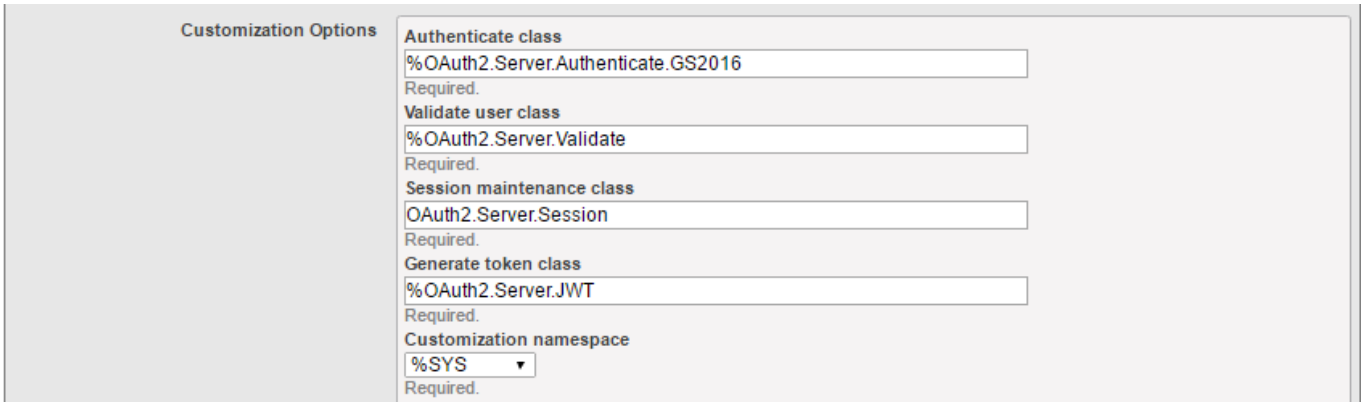
Classes de personalização do servidor

Estas classes pertencem ao pacote %OAuth2. O pacote contém um conjunto de classes internas — utilitários. Só descrevemos as classes que podem ser usadas por desenvolvedores. Estas classes são mencionadas na página de configuração do servidor do OAuth 2.0

%OAuth2.Server.Authenticate	Página CSP %OAuth2.Server.Authenticate atua como uma subclasse para todas as classes Authenticate escritas por usuários, além da classe Authenticate padrão. A classe Authenticate é usada pelo endpoint de autorização no OAuth2.Server.Auth para autenticar o usuário. Essa classe permite a personalização do processo de autenticação.Os seguintes métodos talvez sejam implementados para substituir o padrão no OAuth2.Server:· DirectLogin – use apenas quando não quiser mostrar a página de login· DisplayLogin – implementa o formulário de login do servidor de autorização· DisplayPermissions – implementa o formulário com uma lista de escopos solicitados Outras personalizações de aparência e visual podem ser feitas ao modificar o CSS. Os estilos CSS são definidos no método DrawStyle. loginForm é para o formulário DisplayLogin permissionForm é para o formulário DisplayPermissions
%OAuth2.Server.Validate	Página CSP Esta é a classe Validate User padrão incluída no servidor. A classe padrão usará o banco de dados do usuário da instância do Cache onde o servidor

	de autorização está localizado para validar o usuário. As propriedades aceitas serão o emissor (Issuer), as funções e o sub (Username). A Classe Validate User é especificada na configuração do servidor de autorização. Precisa conter um método ValidateUser, que validará uma combinação de nome de usuário/senha e retornará um conjunto de propriedades associadas ao usuário.
%OAuth2.Server.Generate	Objeto registrado O %OAuth2.Server.Generate é a classe Generate Token padrão incluída no servidor. A classe padrão gerará uma string aleatória como o token de acesso opaco. A classe Generate Token é especificada na configuração do servidor de autorização. Precisa conter um método GenerateAccessToken que será usado para gerar um token de acesso com base na array de propriedades retornada pelo método ValidateUser.
%OAuth2.Server.JWT	Objeto registrado O %OAuth2.Server.JWT é a classe Generate Token que cria um JSON Web Token incluído no servidor. A classe Generate Token é especificada na configuração do servidor de autorização. Precisa conter um método GenerateAccessToken que será usado para gerar um token de acesso com base na array de propriedades retornada pelo método ValidateUser.
%OAuth2.Utils	Objeto registrado Esta classe implementa, entre outros, o registro de várias entidades. Um código de amostra no capítulo Personalização mostra o possível uso.

A imagem a seguir mostra a seção correspondente da configuração do servidor de autorização OAuth 2.0



Caso você use o OpenID Connect com token de identidade formatado JWT (idtoken), substitua a classe Generate Token padrão *%OAuth2.Server.Generate* com *%OAuth2.Server.JWT* na configuração ou deixe a classe Generate padrão.

Discutiremos as opções de personalização com mais detalhes mais tarde em um capítulo separado.

Classes de API públicas

As classes de API públicas são usadas por desenvolvedores de aplicativos para fornecer valores corretos para o fluxo de mensagens de aplicativos da Web, bem como para realizar a validação de tokens de acesso, introspecção e assim por diante.

Essas classes são implementadas no pacote %SYS.OAuth2. A tabela lista algumas das classes implementadas.

%SYS.OAuth2.AccessToken	Objeto registrado A classe %SYS.OAuth2.AccessToken define as operações do

	cliente que permitem que um token de acesso seja usado para autorizar um servidor de recursos. O token subjacente é armazenado no OAuth2.AccessToken no banco de dados CACHESYS. OAuth2.AccessToken é indexado pela combinação de SessionId e ApplicationName. Portanto, apenas um escopo pode ser solicitado para cada SessionId/ApplicationName. Se uma segunda solicitação for feita com um escopo diferente e o token de acesso ainda não tiver sido concedido, o escopo na nova solicitação se tornará o escopo esperado.
%SYS.OAuth2.Authorization	Objeto registrado A classe %SYS.OAuth2.Authorization contém as operações usadas para autorizar um cliente ao obter um token de acesso. O token subjacente é armazenado no OAuth2.AccessToken no banco de dados CACHESYS. OAuth2.AccessToken é indexado pela combinação de SessionId e ApplicationName. Portanto, apenas um escopo pode ser solicitado para cada SessionId/ApplicationName. Se uma segunda solicitação for feita com um escopo diferente e o token de acesso ainda não tiver sido concedido, o escopo na nova solicitação se tornará o escopo esperado. Observe que essa classe está no CACHELIB e, portanto, disponível em qualquer lugar. No entanto, o armazenamento do token está no CACHESYS e, portanto, não está diretamente disponível para a maior parte do código.
%SYS.OAuth2.Validation	Objeto registrado A classe %SYS.OAuth2.Validation define os métodos usados para validar (ou invalidar) um token de acesso. O token subjacente é armazenado no OAuth2.AccessToken no banco de dados CACHESYS. OAuth2.AccessToken é indexado pela combinação de SessionId e ApplicationName. Portanto, apenas um escopo pode ser solicitado para cada SessionId/ApplicationName. Se uma segunda solicitação for feita com um escopo diferente e o token de acesso ainda não tiver sido concedido, o escopo na nova solicitação se tornará o escopo esperado.

Vamos analisar alguns métodos e classes desse grupo mais a fundo.

Cada classe de aplicação cliente, que usa o token de acesso, PRECISA conferir a validade dele. Isso é feito em algum lugar no método OnPage (ou o método correspondente na página ZENMojo ou ZEN).

Este é o fragmento de código:

```
 // Check if we have an access token from oauth2 server
 set isAuthorized=##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,,"sc
ope1,
     scope2",.accessToken,.idtoken,.responseProperties,.error)

 // Continue with further checks if an access token exists.
 // Below are all possible tests and may not be needed in all cases.
 // The JSON object which is returned for each test is just displayed.
 if isAuthorized {
    // do whatever – call resource server API to retrieve data of interest
 }
```

Sempre que chamamos a API do servidor de recursos, precisamos fornecer o token de acesso. Isso é feito pelo método **AddAccessToken** da classe *%SYS.OAuth2.AccessToken*, veja o fragmento de código aqui

```
 set httpRequest=##class(%Net.HttpRequest).%New()
  // AddAccessToken adds the current access token to the request.
  set sc=##class(%SYS.OAuth2.AccessToken).AddAccessToken(
    httpRequest,,
    ..#SSLCONFIG,
    ..#OAUTH2APPNAME)
 if $$$ISOK(sc) {
    set sc=httpRequest.Get(.. Service API url …)
 }
```

No código de amostra fornecido nas partes anteriores da nossa série, é possível ver este código no método OnPreHTTP da primeira página do aplicativo (Cache1N). Esse é o melhor local para realizar a verificação do token de acesso para a página inicial do aplicativo.

```
ClassMethod OnPreHTTP() As %Boolean [ ServerOnly = 1 ]
{
 set scope="openid profile scope1 scope2"
    #dim %response as %CSP.Response
 if ##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,,
    scope,.accessToken,.idtoken,.responseProperties,.error) {
      set %response.ServerSideRedirect="Web.OAUTH2.Cache2N.cls"
 }
 quit 1
}
```

O método **IsAuthorized** da classe *SYS.OAuth2.AccessToken* no código acima é verificar se o token de acesso válido existe e, se não existe, permite mostrar o conteúdo da página com um botão de login/link apontando para o formulário de autenticação do servidor de autorização. Caso contrário, redireciona para a segunda página, que faz o trabalho de recuperar os dados.

Podemos, no entanto, alterar o código para que fique assim:

```
ClassMethod OnPreHTTP() As %Boolean [ ServerOnly = 1 ]
{
 set scope="openid profile scope1 scope2"
 set sc=##class(%SYS.OAuth2.Authorization).GetAccessTokenAuthorizationCode(
    ..#OAUTH2APPNAME,scope,..#OAUTH2CLIENTREDIRECTURI,.properties)
 quit +sc
}
```

Essa variante tem um efeito diferente. Ao usar o método **GetAccessTokenAuthorizationCode** da classe *%SYS.OAuth2.Authorization*, navegamos diretamente até o formulário de autenticação do servidor de autenticação, sem mostrar o conteúdo da primeira página do nosso aplicativo.

Isso pode ser útil nos casos em que o aplicativo da Web é invocado de um aplicativo nativo do dispositivo móvel, onde algumas informações do usuário já foram mostradas pelo aplicativo nativo (o launcher) e não há necessidade de exibir a página da Web com um botão apontando para o servidor de autorização.

Se você usa o token JWT assinado, então precisa validar o conteúdo dele. Isso é feito pelo seguinte método:

```
 set valid=##class(%SYS.OAuth2.Validation).ValidateJWT(applicationName,accessToken,sc
ope,,.jsonObject,.securityParameters,.sc)
```

Veja a descrição detalhada dos parâmetros do método na documentação de Referência de Classes.

Personalização

Vamos passar algum tempo descrevendo quais opções o OAUTH oferece para a personalização da IU de autenticação/autorização.

Suponha que a política da sua empresa exija um comportamento mais restritivo de concessão de escopo. Por exemplo, você pode executar um aplicativo de home banking que se conecta a vários sistemas bancários no seu banco. O banco só concede acesso ao escopo que contém informações sobre a conta bancária real que está sendo recuperada. Como o banco administra milhões de contas, é impossível definir o escopo estático para cada conta. Em vez disso, você pode gerar o escopo em tempo real — durante o processamento da autorização, como parte do código da página de autorização personalizada.

Para o propósito da demonstração, precisamos adicionar mais um escopo à configuração do servidor — veja a imagem.

Também adicionamos a referência à classe Authenticate personalizada, chamada %OAuth2.Server.Authenticate.Bank.

Então, como é a classe de autenticação bancária? Veja uma possível variante da classe. Ela melhora os formulários de autenticação e autorização com dados fornecidos pelo usuário. As informações que fluem entre os métodos **BeforeAuthenticate**, **DisplayPermissions** e **AfterAuthenticate** são transmitidas pela variável de *propriedades* da classe *%OAuth2.Server.Properties*

```
Class %OAuth2.Server.Authenticate.Bank Extends %OAuth2.Server.Authenticate
{
/// Add CUSTOM BESTBANK support for account scope.
ClassMethod BeforeAuthenticate(scope As %ArrayOfDataTypes, properties As %OAuth2.Serv
er.Properties) As %Status
{
 // If launch scope not specified, then nothing to do
 If 'scope.IsDefined("account") Quit $$$OK
 // Get the launch context from the launch query parameter.
 Set tContext=properties.RequestProperties.GetAt("accno")
 // If no context, then nothing to do
 If tContext="" Quit $$$OK

 try {
    // Now the BestBank context should be queried.
    Set tBankAccountNumber=tContext
    // Add scope for accno. -> dynamically modify scope (no account:<accno> scope exi
sts in the server configuration)
    // This particular scope is used to allow the same accno to be accessed via accou
nt
    // if it was previously selected by account or account:accno when using cookie su
pport
    Do scope.SetAt("Access data for account "_tBankAccountNumber,"account:"_tBankAcco
untNumber)
    // We no longer need the account scope, since it has been processed.
    // This will prevent existence of account scope from forcing call of DisplayPermi
ssions.
    Do scope.RemoveAt("account")
```

```
    // Add the accno property which AfterAuthenticate will turn into a response prope
rty
    Do properties.CustomProperties.SetAt(tBankAccountNumber,"account_number")
 } catch (e) {
    s ^dk("err",$i(^dk("err")))=e.DisplayString()
 }
 Quit $$$OK
}

/// Add CUSTOM BESTBANK support for account scope.
/// If account_number custom property was added by either BeforeAuthenticate (account
)
/// or DisplayPermissions (account:accno), then add the needed response property.
ClassMethod AfterAuthenticate(scope As %ArrayOfDataTypes, properties As %OAuth2.Serve
r.Properties) As %Status
{
 // There is nothing to do here unless account_number (account) or accno (account:acc
no) property exists
 try {
    // example of custom logging
    If $$$SysLogLevel>=3 {
     Do ##class(%OAuth2.Utils).LogServerScope("log ScopeArray-
CUSTOM BESTBANK",%token)
    }
    If properties.CustomProperties.GetAt("account_number")'="" {
     // Add the accno query parameter to the response.
     Do properties.ResponseProperties.SetAt(properties.CustomProperties.GetAt("accoun
t_number"),"accno")
    }
 } catch (e) {
    s ^dk("err",$i(^dk("err")))=e.DisplayString()
 }
 Quit $$$OK
}

/// DisplayPermissions modified to include a text for BEST BANK account.
ClassMethod DisplayPermissions(authorizationCode As %String, scopeArray As %ArrayOfDa
taTypes, currentScopeArray As %ArrayOfDataTypes, properties As %OAuth2.Server.Propert
ies) As %Status
{
 Set uilocales = properties.RequestProperties.GetAt("ui_locales")
 Set tLang = ##class(%OAuth2.Utils).SelectLanguage(uilocales,"%OAuth2Login")
 // $$$TextHTML(Text,Domain,Language)
 Set ACCEPTHEADTITLE = $$$TextHTML("OAuth2 Permissions Page","%OAuth2Login",tLang)
 Set USER = $$$TextHTML("User:","%OAuth2Login",tLang)
 Set POLICY = $$$TextHTML("Policy","%OAuth2Login",tLang)
 Set TERM = $$$TextHTML("Terms of service","%OAuth2Login",tLang)
 Set ACCEPTCAPTION = $$$TextHTML("Accept","%OAuth2Login",tLang)
 Set CANCELCAPTION = $$$TextHTML("Cancel","%OAuth2Login",tLang)
 &html<<html>>
 Do ..DrawAcceptHead(ACCEPTHEADTITLE)
 Set divClass = "permissionForm"
 Set logo = properties.ServerProperties.GetAt("logo_uri")
 Set clientName = properties.ServerProperties.GetAt("client_name")
 Set clienturi = properties.ServerProperties.GetAt("client_uri")
 Set policyuri = properties.ServerProperties.GetAt("policy_uri")
 Set tosuri = properties.ServerProperties.GetAt("tos_uri")
 Set user = properties.GetClaimValue("preferred_username")
```

```
 If user="" {
    Set user = properties.GetClaimValue("sub")
 }
 &html<<body>>
 &html<<div id="topLabel"></div>>
 &html<<div class="#(divClass)#">>
 If user '= "" {
    &html<
     <div>
     <span id="left" class="userBox">#(USER)#<br>#(##class(%CSP.Page).EscapeHTML(user
))#</span>
     >
 }
 If logo '= "" {
    Set espClientName = ##class(%CSP.Page).EscapeHTML(clientName)
   &html<<span class="logoClass"><img src="#(logo)#" alt="#(espClientName)#" title="#
(espClientName)#" align="middle"></span>>
 }
 If policyuri '= "" ! (tosuri '= "") {
   &html<<span id="right" class="linkBox">>
    If policyuri '= "" {
     &html<<a href="#(policyuri)#" target="_blank">#(POLICY)#</a><br>>
    }
    If tosuri '= "" {
     &html<<a href="#(tosuri)#" target="_blank">#(TERM)#</a>>
    }
   &html<</span>>
 }
 &html<</div>>
 &html<<form>>
 Write ##class(%CSP.Page).InsertHiddenField("","AuthorizationCode",authorizationCode)
,!
 &html<<div>>
 If $isobject(scopeArray), scopeArray.Count() > 0 {
    Set tTitle = $$$TextHTML(" is requesting these permissions:","%OAuth2Login",tLang
)
   &html<<div class="permissionTitleRequest">>
    If clienturi '= "" {
     &html<<a href="#(clienturi)#" target="_blank">#(##class(%CSP.Page).EscapeHTML(cl
ientName))#</a>>
    } Else {
     &html<#(##class(%CSP.Page).EscapeHTML(clientName))#>
    }
   &html<#(##class(%CSP.Page).EscapeHTML(tTitle))#</div>>
    Set tCount = 0
    Set scope = ""
    For {
     Set display = scopeArray.GetNext(.scope)
     If scope = "" Quit
     Set tCount = tCount + 1
     If display = "" Set display = scope
     Write "<div class='permissionItemRequest'>"_tCount_". "_##class(%CSP.Page).Escap
eHTML(display)_"</div>"
    }
 }

 If $isobject(currentScopeArray), currentScopeArray.Count() > 0 {
    Set tTitle = $$$TextHTML(" already has these permissions:","%OAuth2Login",tLang)
   &html<<div>>
```

```
   &html<<div class="permissionTitleExisting">>
    If clienturi '= "" {
     &html<<a href="#(clienturi)#" target="_blank">#(##class(%CSP.Page).EscapeHTML(cl
ientName))#</a>>
    } Else {
     &html<#(##class(%CSP.Page).EscapeHTML(clientName))#>
    }
   &html<#(##class(%CSP.Page).EscapeHTML(tTitle))#</div>>
    Set tCount = 0
    Set scope = ""
    For {
     Set display = currentScopeArray.GetNext(.scope)
     If scope = "" Quit
     Set tCount = tCount + 1
     If display = "" Set display = scope
     Write "<div class='permissionItemExisting'>"_tCount_". "_##class(%CSP.Page).Esca
peHTML(display)_"</div>"
    }
   &html<</div>>
 }

 /*********************************/
 /*  BEST BANK CUSTOMIZATION      */
 /*********************************/
 try {
    If properties.CustomProperties.GetAt("account_number")'="" {
     // Display the account number obtained from account context.
     Write "<div class='permissionItemRequest'><b>Selected account is "_properties.Cu
stomProperties.GetAt("account_number")_"</b></div>",!

     // or, alternatively, let user add some more information at this stage (e.g. lin
ked account number)
     //Write "<div>Account Number: <input type='text' id='accno' name='p_accno' place
holder='accno' autocomplete='off' ></div>",!
    }
 } catch (e) {
    s ^dk("err",$i(^dk("err")))=e.DisplayString()
 }

 /* original implementation code continues here... */
 &html<
   <div><input type="submit" id="btnAccept" name="Accept" value="#(ACCEPTCAPTION)#"/>
</div>
   <div><input type="submit" id="btnCancel" name="Cancel" value="#(CANCELCAPTION)#"/>
</div>
    >
 &html<</form>
 </div>>
 Do ..DrawFooter()
 &html<</body>>
 &html<<html>>
 Quit 1
}

/// For CUSTOM BESTBANK we need to validate that patient entered,
/// ! javascript in this method is only needed when we let user enter some addtional
data
/// within DisplayPermissions method !
ClassMethod DrawAcceptHead(ACCEPTHEADTITLE)
```

```
{
 &html<<head><title>#(ACCEPTHEADTITLE)#</title>>
 Do ..DrawStyle()
 &html<
 <script type="text/javascript">
 function doAccept()
 {
    var accno = document.getElementById("accno").value;
    var errors = "";
    if (accno !== null) {
     if (accno.length < 1) {
       errors = "Please enter account number name";
     }
    }
    if (errors) {
     alert(errors);
     return false;
    }

    // submit the form
    return true;
 }
 </script>
 >
 &html<</head>>
}

}
```

Como você pode ver, a classe %OAuth2.Server.Properties contém várias arrays, que são passadas. São elas:

· RequestProperties — contém parâmetros da solicitação de autorização

· CustomProperties — contêiner para a troca de dados entre o mencionado acima

· ResponseProperties — contêiner para as propriedades serem adicionadas ao objeto de resposta JSON a uma solicitação de token

· ServerProperties — contém propriedades compartilhadas que o servidor de autorização expõe para o código de personalização (por exemplo, logouri, clienturi, etc⋯)

Além disso, ela contém várias propriedades de "declarações", que são usadas para especificar quais declarações devem ser retornadas pelo servidor de autorização.

Para chamar essa página de autenticação corretamente, modificamos nosso código inicial da página do cliente para que fique assim:

```
 set scope="openid profile scope1 scope2 account"
 // this data comes from application (a form data or so...) and sets a context for ou
r request
 // we can, through subclassing the Authenticate class, display this data to user so
he/she can decide
 // whether to grant access or not
 set properties("accno")="75-452152122-5320"
 set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(
   ..#OAUTH2APPNAME,
    scope,
   ..#OAUTH2CLIENTREDIRECTURI,
```

```
    .properties,
   .isAuthorized,
    .sc)
 if $$$ISERR(sc) {
    write "GetAuthorizationCodeEndpoint Error="
   write ..EscapeHTML($system.Status.GetErrorText(sc))_"<br>",!
 }
```

Como você pode ver, adicionamos o escopo da conta e o nó "accno" da array de propriedades com um valor de contexto, que pode se originar em diferentes partes do nosso aplicativo. Esse valor é transmitido dentro do token de acesso ao servidor de recursos para processamento adicional.

Existe um cenário da vida real que usa a lógica descrita acima — o padrão FHIR para a troca de históricos eletrônicos de pacientes.

Depuração

O framework do OAUTH tem depuração integrada. Isso é muito útil, pois toda a comunicação entre o cliente e os servidores é criptografada. O recurso de depuração permite capturar dados de tráfego gerados pelas classes de API antes que sejam enviados pela rede. Para depurar seu código, você pode implementar uma rotina ou classe simples de acordo com o código abaixo. Você precisa implementar este código em todas as instâncias de comunicação do InterSystems IRIS! Nesse caso, é melhor fornecer um nome de arquivo que indique a função dele dentro do processo de fluxo do OAUTH. (O código de amostra abaixo é salvo como uma rotina rr.mac, mas você decide o nome.)

```
 // d start^rr()
start() public {
 new $namespace
 set $namespace="%sys"
 kill ^%ISCLOG
 set ^%ISCLOG=5
 set ^%ISCLOG("Category","OAuth2")=5
 set ^%ISCLOG("Category","OAuth2Server")=5
 quit
}

 // d stop^rr()
stop() public {
 new $namespace
 set $namespace="%sys"
 set ^%ISCLOG=0
 set ^%ISCLOG("Category","OAuth2")=0
 set ^%ISCLOG("Category","OAuth2Server")=0
 quit

}

 // display^rr()
display() public {
 new $namespace
 set $namespace="%sys"
 do ##class(%OAuth2.Utils).DisplayLog("c:\temp\oauth2_auth_server.log")
 quit
}
```

Em seguida, antes de começar a testar, abra um terminal e invoque d start^rr() em todos os nós do InterSystems

IRIS (cliente, servidor de autorização ou servidor de recursos). Depois de concluído, execute d stop^rr() e d display^rr() para preencher os arquivos de log.

Resumo

Nesta série de artigos, aprendemos a usar a implementação do OAuth 2.0 do InterSystems IRIS. Começando com a demonstração simples do aplicativo cliente na parte 1, seguido pela amostra complexa descrita na parte 2. Por fim, descrevemos as classes mais importantes da implementação do OAuth 2.0 e explicamos quando elas devem ser chamadas nos aplicativos dos usuários.

Quero agradecer em especial a Marvin Tener, pela paciência infinita ao responder às minhas perguntas, às vezes idiotas, e por revisar a série.

#Autenticação #Controle de acesso #OAuth2 #Segurança #Caché #Ensemble #InterSystems IRIS

URL de origem:https://pt.community.intersystems.com/post/implementa%C3%A7%C3%A3o-do-open-authorization-framework-do-intersystems-iris-oauth-20-parte-3