記事
Shintaro Kaminaka · 2020年8月20日 23m read

InterSystems IRIS Open Authorization Framework（OAuth 2.0）の実装 - パート2

作成者：Daniel Kutac（InterSystems セールスエンジニア）
注意: _*使用されている URL に戸惑っている方のために。元の連載記事では、dk-gs2016 と呼ばれるマシンの画面を使用していました。 新しいスクリーンショットは別のマシンから取得されています。 *WIN-U9J96QBJSAG という URL は dk-gs2016**_であると見なしても構いません。

パート2. 認可サーバー、OpenID Connect サーバー

この短い連載の前のパートでは、OAUTH[1] クライアントとして機能する単純な使用事例について学びました。
今回は私たちの経験をまったく新しいレベルに引き上げましょう。 InterSystems IRIS がすべての OAUTH の役割を果たす、より複雑な環境を構築します。
クライアントの作成方法はすでに分かっていますので、認可サーバーだけでなく、OpenID Connect[2] プロバイダーにも注意を向けましょう。
前のパートと同様に、環境を準備する必要があります。
今回はより多くの変動要素があるため、より注意を要します。

具体例を見る前に、OpenID Connect について少し説明する必要があります。
前のパートの内容を覚えていらっしゃるかと思いますが、Google から認可してもらうため、まずは自身がGoogle で認証を受けることを求められていました。 認証は OAUTH フレームワークには含まれていません。
実際、OAUTH に依存しない多くの認証フレームワークがあります。 そのうちの1つに OpenID と呼ばれるものがあります。 当初は単独の構想で開始されましたが、最近では OAUTH フレームワークによって提供されるインフラストラクチャ、つまり通信とデータ構造が活用されています。
その結果、OpenID Connect が誕生しました。 事実、多くの人がこれを OAUTH の強化版と呼んでいます。
実際、OpenID Connect を使用すると、認可するだけでなく、OAUTH フレームワークのよく知られたインターフェースを使用して認証することもできます。

複雑な OpenID Connect のデモ

ここでは、パート 1 のクライアントコードの多くを活用します。
そうすることで多くの手間が省けるため、環境のセットアップに集中できます。

前提条件

今回は、SSL が有効になっている既存の Web サーバーに PKI インフラストラクチャを追加する必要があります。
OpenID Connect の要件である暗号化が必要です。 ユーザー認証が必要な場合は、第三者がエージェント（クライアント、認証サーバーなど）になりすますし、ネットワーク経由でユーザーの機密データを送信できないようにする必要があります。 そこで X.509 ベースの暗号化の出番です。
注意 : Cache 2017.1 以降では、X.509 証明書を使用して JWT / JWKS（JSON Web Key Set）を生成する必要はありません。
ここでは下位互換性と単純化を図るためにこのオプションを使用しています。

PKI

厳密に言えば、Caché PKI インフラストラクチャを使用する必要はまったくありません。ただし、openssl などのツールを直接使用してすべての証明書を生成するよりは楽です。

詳細は InterSystems IRIS
のドキュメントや他の場所でも確認できますので、ここでは証明書の生成に関する詳細は触れません。
証明書を生成した結果、3 つの公開鍵/秘密鍵のペアと関連する証明書が作成されます。
これらを次のように呼びます。

- 発行元認証局の rootca（rootca.cer）
- 認可サーバーおよび OpenID サーバー用の auth（auth.cer および auth.key）
- クライアントアプリケーションサーバー用の client（client.cer および client.key）

X.509 資格情報

デモ中に交換された JSON Web Token（JWT）に署名して検証できるよう、個々のサーバーで X.509
資格情報を定義する必要があります。

認可サーバーと認証サーバーの構成

ここでは X.509 資格情報の定義方法については詳しく説明せず、AUTHSERVER
インスタンスの資格情報のスクリーンショットを示します。

画像のように、AUTHSERVER にはその秘密鍵と証明書がありますが、CLIENT
に関しては公開鍵を含む証明書しかありません。

クライアントサーバーの構成

同様に、CLIENT インスタンスで資格情報が定義されています。

ここで、CLIENT には秘密鍵と証明書がありますが、AUTHSERVER
に関しては公開鍵を含む証明書しかありません。

リソースサーバーの構成

このセットアップの例では、RESSERVER インスタンスで X509 資格情報を定義する必要はありません。

OAUTH の構成

この連載のパート 1 で説明した構成と同様に、サーバーを OAUTH 用に構成する必要があります。
まずは、OAUTH 構成全体の中心的なコンポーネントである AUTHSERVER インスタンスから始めましょう。

AUTHSERVER

System Management Portal で、**System Administration > Security > OAuth 2.0 > Server Configuration** を開きます。
メニューのリンクをクリックし、次のフォーム項目に入力します。

- Host name（ホスト名）
- Port（ポート、省略可）
- Prefix（プレフィックス、省略可） – これら 3 つのフィールドは *Issuer endpoint*（発行者エンドポイント）を構成します。
- 更新トークンを返す条件を指定します。
- Supported grant types（サポートされているグラント種別）をチェックします。このデモでは 4 つの種別すべてにチェックします。 ただし、認可コードのみが使用されます。
- 必要に応じて Audience required（オーディエンスを要求）をチェックします。これにより、*aud* プロパティが認可コードと暗黙的な許可(Implicit)のリクエストに追加されます。
- 必要に応じて Support user session（ユーザーセッションをサポート）をチェックします。これにより、認可サーバーが現在このブラウザを使用しているユーザーのログイン状態を維持するために *httpOnly* Cookie が使用されます。 2
回目以降のアクセストークンの要求では、ユーザー名とパスワードの入力は求められません。
- Endpoint intervals（エンドポイントの間隔）を指定します。
- このサーバーでサポートされるスコープ（Supportted scopes）を定義します。
- Customization
Options（カスタマイズオプション）のデフォルト値を受け入れるか、カスタム値を入力します。**注意: **JWT が単なる opaque トークンではなくアクセストークンとして使用されるよう、Generate token class（生成トークンクラス）の値を **%OAuth2.Server.Generate** から **%OAuth2.Server.JWT** に変更してください。
- OAuth 2.0 の要求に従い、HTTP 上の SSL を確立するための登録済み SSL 構成の名前を入力します。
- JSON Web Token（JWT）の設定を入力します。

以下はサンプル構成のスクリーンショットです。

サーバー構成を定義したら、サーバークライアント構成を入力する必要があります。
サーバー構成フォームのページ内で、「Client Configurations」（クライアント構成）ボタンをクリックし、CLIENT インスタンスおよび RESSERVER インスタンスの「Create New Configuration」（新しい構成の作成）をクリックします。
(IRISを使用して構成している場合は、「クライアントディスクリプション」の構成ページから以下の設定を行います。)
以下の画像は CLIENT の構成を示しています。

JWT Token（JWT トークン）タブはデフォルト値である空のままにします。
ご覧のとおり、実際のアプリケーションの場合とは異なり、フィールドには無意味なデータが入力されています。
同様に、RESSERVER の構成を示します。

ご覧のとおり、リソースサーバーに必要なのは非常に基本的な情報だけです。具体的には、Client type（クライアント種別）をリソースサーバーに設定する必要があります。 CLIENT では、より詳細な情報とクライアント種別を入力する必要があります（クライアントはクライアントシークレットをサーバーで維持し、クライアントエージェントには送信しない Web アプリケーションとして動作するため、機密(confidential)を選択します）。

CLIENT

SMP で、**System Administration > Security > OAuth 2.0 > Client Configurations** を開きます。
「Create Server Configuration」（サーバー構成の作成）ボタンをクリックし、フォームに入力して保存します。

Issuer Endpoint（発行者エンドポイント）が、AUTHSERVER インスタンスで前に定義した値に対応していることを必ず確認してください！
また、認可サーバーのエンドポイントを Web サーバーの構成に従って変更する必要があります。
この場合は各入力フィールドに「authserver」を埋め込んだだけです。
次に、新しく作成された発行者エンドポイントの横にある「**Client Configurations**」（クライアント構成）リンクをクリックし、「**Create Client Configuration**」（クライアント構成の作成）ボタンをクリックします。

以上です！ ここまでの手順で CLIENT と AUTHSERVER の両方を構成しました。
多くの使用事例ではこれだけで十分です。リソースサーバーは単なる AUTHSERVER のネームスペースにすぎず、結果的にすでに保護されている場合があるからです。 しかし、外部の医師が内部の臨床システムからデータを取得しようとしている使用事例に対応する場合を考えてみましょう。 このような医師がデータを取得できるようにするため、この医師のアカウント情報を監査目的と法医学的な理由でリソースサーバー内に確実に保存したいと思います。 この場合は、続けて RESSERVER を定義する必要があります。

RESSERVER

SMP で、**System Administration > Security > OAuth 2.0 > Client Configurations** を開きます。
「**Create Server Configuration**」（サーバー構成の作成）ボタンをクリックし、フォームに入力して保存します。

ここでは、Cache 2017.1 に実装された新機能である検出機能を使用しました。
ご覧のように、この構成は CLIENT インスタンスの対応する構成と同じデータを使用しています。
次に、新しく作成された発行者エンドポイントの横にある「**Client Configurations**」（クライアント構成）リンクをクリックし、「**Create Client Configuration**」（クライアント構成の作成）ボタンをクリックします。

X.509 資格情報から JWT を作成することはお勧めしませんが、ここでは互換性のために使用しました。

CLIENTとほとんど同じ手順でしたが、必要なものでした。
しかし、これで先に進んでコーディングできるようになりました！

クライアントアプリケーション

話をできる限り簡単にするため、パート1で説明した Google の例から多くのコードをリサイクルします。
クライアントアプリケーションは /csp/myclient で実行されるわずか 2 つの CSP
ページからなるアプリケーションです。セキュリティは強制せず、未認証ユーザーとして実行されます。

ページ 1

```
Class Web.OAUTH2.Cache1N Extends %CSP.Page
{

Parameter OAUTH2CLIENTREDIRECTURI = "https://dk-
gs2016/client/csp/myclient/Web.OAUTH2.Cache2N.cls";

Parameter OAUTH2APPNAME = "demo client";

ClassMethod OnPage() As %Status
{
  &html<<html>

<body>
  <h1>Cache&acute; OAuth2 ??????????????</h1>
  <p>???????????OAuth2 ???????? Cache&acute; API ????????????????
  <p>???? Cache&acute; ?????????????????????????????? Cache&acute; ??????????????????
?????????
 >

  // ?????????????????????????? URL ???????
  // ???? URL ?????????????

  // DK: 'dankut' ??????????????
  set scope="openid profile scope1 scope2"
  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>",!
  }

  &html<
  <div class="portalLogoBox"><a class="portalLogo" href="#(url)#">Authorize for <b>IS
C</b></a></div>
  </body></html>>
  Quit $$$OK
}

ClassMethod OnPreHTTP() As %Boolean [ ServerOnly = 1 ]
{
```

```
  #dim %response as %CSP.Response
  set scope="openid profile scope1 scope2"
  if ##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,,scope,.accessTok
en,.idtoken,.responseProperties,.error) {
    set %response.ServerSideRedirect="Web.OAUTH2.Cache2N.cls"
  }
  quit 1
}

}
```

ページ 2

```
Class Web.OAUTH2.Cache2N Extends %CSP.Page
{

Parameter OAUTH2APPNAME = "demo client";

Parameter OAUTH2ROOT = "https://dk-gs2016/resserver";

Parameter SSLCONFIG = "SSL4CLIENT";

ClassMethod OnPage() As %Status
{
    &html<<html>

<body>>

    // OAuth2 ??????????????????????????????
    set isAuthorized=##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,,
"scope1 scope2",.accessToken,.idtoken,.responseProperties,.error)

    // ????????????????????????
    // ??????????????????????????????????????????
    // ????????? JSON ????????????????????
    if isAuthorized {
        write "<h3>Authorized!</h3>",!


        // JWT ????????????????????????????????
        set valid=##class(%SYS.OAuth2.Validation).ValidateJWT(..#OAUTH2APPNAME,access
Token,"scope1 scope2",,.jsonObject,.securityParameters,.sc)
        if $$$ISOK(sc) {
            if valid {
                write "Valid JWT"_"<br>",!
            } else {
                write "Invalid JWT"_"<br>",!
            }
            write "Access token="
            do jsonObject.%ToJSON()
            write "<br>",!
        } else {
            write "JWT Error="_..EscapeHTML($system.Status.GetErrorText(sc))_"<br>",!

        }
        write "<br>",!

        // ????????????????????????????????RFC 7662 ??????????
```

```
        set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection(..#OAUTH2APPNAME,acc
essToken,.jsonObject)
        if $$$ISOK(sc) {
            write "Introspection="
            do jsonObject.%ToJSON()
            write "<br>",!
        } else {
            write "Introspection Error="_..EscapeHTML($system.Status.GetErrorText(sc)
)_"<br>",!
        }
        write "<br>",!

        if idtoken'="" {
            // ID ???????????OpenID Connect Core ????????????
            set valid=##class(%SYS.OAuth2.Validation).ValidateIDToken(
                ..#OAUTH2APPNAME,
                idtoken,
                accessToken,,,
                .jsonObject,
                .securityParameters,
                .sc)
            if $$$ISOK(sc) {
                if valid {
                    write "Valid IDToken"_"<br>",!
                } else {
                    write "Invalid IDToken"_"<br>",!
                }
                write "IDToken="
                do jsonObject.%ToJSON()
                write "<br>",!
            } else {
                write "IDToken Error="_..EscapeHTML($system.Status.GetErrorText(sc))_
"<br>",!
            }
        } else {
            write "No IDToken returned"_"<br>",!
        }
        write "<br>",!

        // ????????????????????????????????????????????

        // Userinfo ?????????????????????OpenID Connect Core ????????????
        set sc=##class(%SYS.OAuth2.AccessToken).GetUserinfo(
            ..#OAUTH2APPNAME,
            accessToken,,
            .jsonObject)
        if $$$ISOK(sc) {
            write "Userinfo="
            do jsonObject.%ToJSON()
            write "<br>",!
        } else {
            write "Userinfo Error="_..EscapeHTML($system.Status.GetErrorText(sc))_"<b
r>",!
        }
        write "<p>",!

        /************************************************
        *                                              *
        *   ?????????????????????   *
```

```
        *                                                   *
        *****************************************************/

        // ????? 1 - ???????? - ?????????????????????????
        //     ??????????????????????????????????
        //  ???????????????

        // ????? 2 - ????????OpenID Connect??????
        //  ???????????? CSP ??????????????????????
        //  - ??????????????????????


        write "<h4>?????????????????","</h4>",!
        set httpRequest=##class(%Net.HttpRequest).%New()
        // AddAccessToken ??????????????????????????
        set sc=##class(%SYS.OAuth2.AccessToken).AddAccessToken(
            httpRequest,,
            ..#SSLCONFIG,
            ..#OAUTH2APPNAME)
        if $$$ISOK(sc) {
            set sc=httpRequest.Get(..#OAUTH2ROOT_"/csp/portfolio/oauth2test.demoResou
rce.cls")
        }
        if $$$ISOK(sc) {
            set body=httpRequest.HttpResponse.Data
            if $isobject(body) {
                do body.Rewind()
                set body=body.Read()
            }
            write body,"<br>",!
        }
        if $$$ISERR(sc) {
            write "Resource Server Error="_..EscapeHTML($system.Status.GetErrorText(s
c))_"<br>",!
        }
        write "<br>",!

        write "<h4>Call resource server - no auth, just token validity check","</h4>"
,!
        set httpRequest=##class(%Net.HttpRequest).%New()
        // AddAccessToken??????????????????????????
        set sc=##class(%SYS.OAuth2.AccessToken).AddAccessToken(
            httpRequest,,
            ..#SSLCONFIG,
            ..#OAUTH2APPNAME)
        if $$$ISOK(sc) {
            set sc=httpRequest.Get(..#OAUTH2ROOT_"/csp/portfolio2/oauth2test.demoReso
urce.cls")
        }
        if $$$ISOK(sc) {
            set body=httpRequest.HttpResponse.Data
            if $isobject(body) {
                do body.Rewind()
                set body=body.Read()
            }
            write body,"<br>",!
        }
        if $$$ISERR(sc) {
            write "Resource Server Error="_..EscapeHTML($system.Status.GetErrorText(s
```

```
c))_"<br>",!
        }
        write "<br>",!
    } else {
        write "Not Authorized!<p>",!
        write "<a href='Web.OAUTH2.Cache1N.cls'>Authorize me</a>"
    }
    &html<</body></html>>
    Quit $$$OK
}

}
```

次のスクリーンショットで処理を説明します。

AUTHSERVER インスタンスの認可 / OpenID Connect 認証サーバーのログインページ

AUTHSERVER のユーザー同意ページ

その後、最終的に結果ページが表示されます。

ご覧のとおり、実際にコードを読んでもパート 1 で示したクライアントのコードとほとんど違いはありません。
ページ 2 には違いがあります。 これはデバッグ情報であり、JWT の有効性をチェックしています。 返ってきた JWT を検証した後、AUTHSERVER
からのユーザー識別情報に関するデータに対してイントロスペクションを実行できました。
ここではこの情報をページに出力しただけですが、それ以上のこともできます。 上記で説明した外部の医師の使用事例と同様に、必要に応じて識別情報を使用し、それを認証目的でリソースサーバーに渡すことができます。
または、この情報をパラメーターとしてリソースサーバーへの API 呼び出しに渡すこともできます。

次の段落では、ユーザー識別情報の使用方法について詳しく説明します。

リソースアプリケーション

リソースサーバーは認可 / 認証サーバーと同じサーバーにすることができ、多くの場合はそうなります。
しかし、このデモでは 2 つのサーバーを独立した InterSystems IRIS インスタンスにしました。

したがって、リソースサーバーでセキュリティコンテキストを使用する方法には 2 つあります。

方法 1 – 認証なし

これは最も単純なケースです。 認可 / 認証サーバーはまったく同じ Caché インスタンスです。
この場合、単一の目的のために特別に作成された csp
アプリケーションにアクセストークンを渡すだけで、OAUTH
を使用するクライアントアプリケーションにデータを提供し、データを要求することを認可できます。

リソース csp アプリケーションの構成（ここでは /csp/portfolio2
と呼びました）は、以下のスクリーンショットのようになります。

最小限のセキュリティをアプリケーション定義に組み込み、特定の CSP ページのみを実行できるようにします。

または、リソースサーバーは従来の Web ページの代わりに REST API を提供できます。
実際のシナリオでは、セキュリティコンテキストを微調整するのはユーザー次第です。

ソースコードの例：

```
Class oauth2test.demoResource Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
    set accessToken=##class(%SYS.OAuth2.AccessToken).GetAccessTokenFromRequest(.sc)
    if $$$ISOK(sc) {
        set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection("RESSERVER resource"
,accessToken,.jsonObject)
        if $$$ISOK(sc) {
            // ?????? jsonObject ???????????

            w "<p><h3>Hello from Cach&eacute; server: <i>/csp/portfolio2</i> applicat
ion!</h3>"
            w "<p>running code as <b>$username = "_$username_"</b> with following <b>
$roles = "_$roles_"</b> at node <b>"_$p($zu(86),"*",2)_"</b>."
        }
    } else {
        w "<h3>NOT AUTHORIZED!</h3>"
        w "<pre>"
        w
        i $d(%objlasterror) d $system.OBJ.DisplayError()
        w "</pre>"
    }
    Quit $$$OK
}

}
```

方法 2 – 委任認証

これはもう 1 つの極端なケースです。ユーザーがリソースサーバーの内部ユーザーと同等のセキュリティコンテキストで作業しているかのように、リソースサーバーでユーザーの識別情報を可能な限り最大限に活用したいと考えています。

解決方法の 1 つは、委任認証を使用することです。

この方法を実行するには、さらにいくつかの手順を実行してリソースサーバーを構成する必要があります。

· 委任認証を有効にする

· ZAUTHENTICATE ルーチンを提供する

· Web アプリケーションを構成する（この場合、/csp/portfolio で呼び出しました）

ここではユーザー識別情報とそのスコープ（セキュリティプロファイル）を提供した AUTHSERVER を信頼しているため、ZAUTHENTICATE ルーチンの実装は非常に単純で簡単です。そのため、ここではいかなるユーザー名も受け入れ、それをスコープと共にリソースサーバーのユーザーデータベースに渡しています（OAUTH スコープと InterSystems IRIS のロール間で必要な変換を行ったうえで）。 それだけです。 残りの処理は InterSystems IRIS によってシームレスに行われます。

これは ZAUTHENTICATE ルーチンの例です。

```
#include %occErrors
#include %occInclude

ZAUTHENTICATE(ServiceName, Namespace, Username, Password, Credentials, Properties) PU
```

```
BLIC
{
    set tRes=$SYSTEM.Status.OK()
    try {
        set Properties("FullName")="OAuth account "_Username
        //set Properties("Roles")=Credentials("scope")
        set Properties("Username")=Username
        //set Properties("Password")=Password
        // Credentials ??? GetCredentials() ????????????????????????
        set Properties("Password")="xxx"    // OAuth2 ??????????????????
        set Properties("Roles")=Password
    } catch (ex) {
        set tRes=$SYSTEM.Status.Error($$$AccessDenied)
    }
    quit tRes
}

GetCredentials(ServiceName,Namespace,Username,Password,Credentials) Public
{
    s ts=$zts
    set tRes=$SYSTEM.Status.Error($$$AccessDenied)

     try {
         If ServiceName="%Service_CSP" {
            set accessToken=##class(%SYS.OAuth2.AccessToken).GetAccessTokenFromReques
t(.sc)
            if $$$ISOK(sc) {
                set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection("RESSERVER r
esource",accessToken,.jsonObject)
                if $$$ISOK(sc) {
                    // ToDo: ??????????????????OpenID????????????????????????
                    set Username=jsonObject.username
                    set Credentials("scope")=$p(jsonObject.scope,"openid profile ",2)
                    set Credentials("namespace")=Namespace
                    // temporary hack
                    //set Password="xxx"
                    set Password=$tr(Credentials("scope")," ",",")
                    set tRes=$SYSTEM.Status.OK()
                } else {
                    set tRes=$SYSTEM.Status.Error($$$GetCredentialsFailed)
                }
            }
        } else {
            set tRes=$SYSTEM.Status.Error($$$AccessDenied)
        }
     } catch (ex) {
         set tRes=$SYSTEM.Status.Error($$$GetCredentialsFailed)
    }
    Quit tRes
}
```

CSP ページ自体は非常にシンプルになります。

```
Class oauth2test.demoResource Extends %CSP.Page
{

ClassMethod OnPage() As %Status
```

```
{
    // ??????????????????????????
    // ????????????????????

    // ????????????????????????????????????????????
    // ????????????????????????????????????????
    // ????????????????????????????????????????
    // ????????????????????????
    w "<p><h3>Hello from Cach&eacute; server: <i>/csp/portfolio</i> application!</h3>
"
    w "<p>running code as <b>$username = "_$username_"</b> with following <b>$roles =
 "_$roles_"</b> at node <b>"_$p($zu(86),"*",2)_"</b>."
    Quit $$$OK
}

}
```

そして最後に、/csp/portfolio の Web アプリケーション構成を示します。

あなたが本当に心配であったなら、最初のバリエーションで行ったように *Permitted クラス*を設定できたかもしれません。 または、REST API を使用していたかもしれません。
しかし、これらの設定に関してはこの記事の中では説明しません。

次回は、InterSystems IRIS OAUTH フレームワークによって導入される個々のクラスについて説明します。
それらの API、およびそれらを呼び出すタイミングと場所について説明します。

[1] この記事で OAUTH について言及する場合、常に RFC 6749（https://tools.ietf.org/html/rfc6749）で規定されている OAuth 2.0 を指しています。
ここでは簡略化のために短縮形の OAUTH を使用しています。

[2] OpenID Connect は OpenID Foundation（http://openid.net/connect）によって管理されています。

#OAuth2 #セキュリティ #アクセス制御 #認証 #Caché #Ensemble #InterSystems IRIS

ソースURL:https://jp.community.intersystems.com/post/intersystems-iris-open-authorization-framework%EF%BC%88oauth-20%EF%BC%89%E3%81%AE%E5%AE%9F%E8%A3%85-%E3%83%91%E3%83%BC%E3%83%882