OAuth 2.0 および OpenID Connect の使用法

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

OAuth 2.0 および OpenID Connect の使用法
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

OAuth 2.0 および OpenID Connect の概要

このページでは、OAuth 2.0 承認フレームワークと OpenID Connect の概要を説明します。別のページでは、OAuth 2.0 および OpenID Connect に対する InterSystems IRIS® のサポートについて説明します。

1.1 基本

OAuth 2.0 承認フレームワークを使用すると、サードパーティのアプリケーション (一般にクライアントと呼ばれる) による HTTP サービス (リソース) への限定的なアクセスを可能にすることができます。このアクセス制限により、クライアントが取得できる情報や使用できるサービスが限定されます。承認サーバは承認のやりとりを調整するか、アクセス権を直接付与します。OpenID Connect は、このフレームワークを拡張して認証を追加します。

1.2 ロール

OAuth 2.0 フレームワークでは、次の 4 つのロールが定義されています。

- リソース所有者 – 通常はユーザです。
- リソース・サーバ – 保護されているデータやサービスをホストするサーバです。
- クライアント – リソース・サーバへの限定的なアクセスを要求するアプリケーションです。これは、クライアント・サーバ・アプリケーションの場合もあれば、サーバを持たないアプリケーション (JavaScript アプリケーションやモバイル・アプリケーションなど) の場合もあります。
- 承認サーバ – リソース・サーバへのクライアントのアクセスを可能にするアクセス・トークンの発行を担うサーバです。このサーバは承認サーバと同じアプリケーションにすることも、異なるアプリケーションにすることもできます。

クライアント、リソース・サーバ、承認サーバは事前の準備によって相互に認識します。承認サーバは、通信可能なクライアント・サーバとリソース・サーバを指定したクライアントのレジストリを保持しています。クライアントが登録されると、承認サーバはクライアント ID とクライアント秘密鍵を生成します。クライアント秘密鍵は秘密にしておく必要があります。クライアントが承認サーバと通信する方法によっては、クライアント・サーバでクライアント秘密鍵が必要になることもあります。その場合は、クライアント秘密鍵をクライアント・サーバへ安全に伝える必要があります。JavaScript アプリケーションなどの一部のシナリオでは、クライアントはクライアント秘密鍵を保護することができません。このようなシナリオでは、クライアントはクライアント秘密鍵を必要としない方法で承認サーバと通信しなければなりません。

1.3 アクセス・トークン

アクセス・トークンには、ユーザやクライアントの ID に関する情報がメタデータと共に格納されます。このメタデータは有効期限、予期される発行者名、予期される対象者、スコープなどで構成されます。

通常、アクセス・トークンの目的は、HTTP 経由で提供されるリソース・サーバの特定のデータやサービスにクライアントがアクセスできるようにすることです。全体的な流れでは、クライアント・アプリケーションが承認サーバにアクセス・トークンを要求します。クライアントは、受け取ったアクセス・トークンをリソース・サーバへの HTTP 要求で使用します。リソース・サーバは、受け取った要求に有効なアクセス・トークンが含まれている場合にのみ、要求された情報を返します。

アクセス・トークンは取り消すこともできます (承認サーバがこの機能をサポートしている場合)。

1.3.1 アクセス・トークンの形態

InterSystems IRIS は、次の 2 種類のアクセス・トークンをサポートしています。

- JSON Web Token (JWT)。JWT は JSON オブジェクトです。JWT はデジタル署名か暗号化またはその両方を行うことができます。

  JWT の 1 種が ID トークンであり、OpenID Connect に固有のトークンです。

  JWT は署名か暗号化またはその両方を行うことができます。

- opaque アクセス・トークン (別名 : 参照トークン)。この形態のアクセス・トークンは、他の場所、具体的には承認サーバに格納されているトークンの識別子です。この識別子は長いランダムな文字列であり、これには推測を困難にするという意図が込められています。

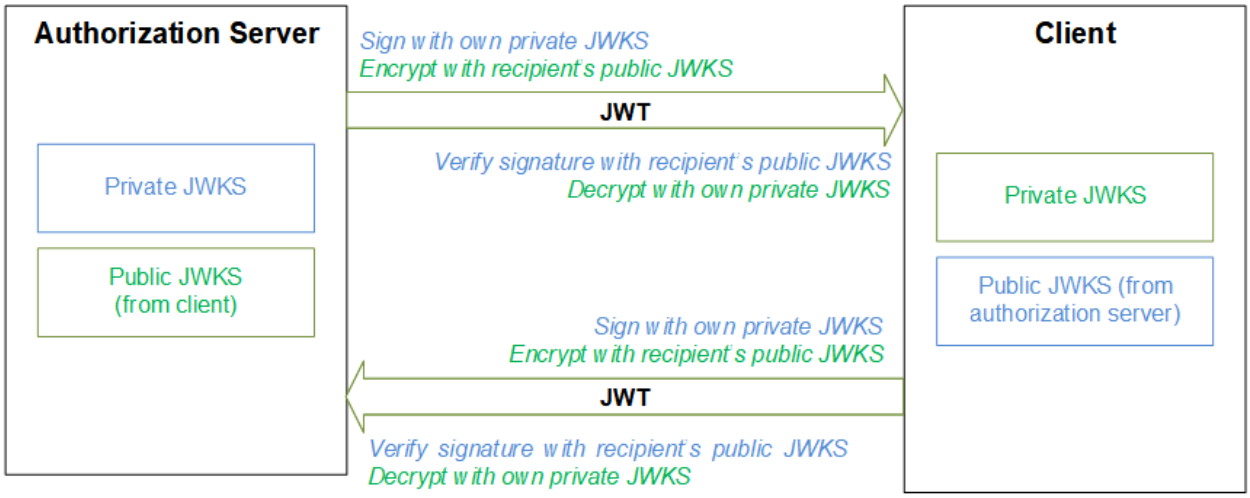
1.3.2 クレーム

アクセス・トークンには、ユーザまたはクライアントの ID の伝達や、トークンの有効期限、予期される発行者名、予期される対象者、スコープなどのメタデータの伝達を行う一連のクレームが含まれています。OpenID Connect Core の仕様ではクレームの標準セットが定義されていますが、その他のクレームも使用できます。

1.3.3 JWT および JWKS

上記のように、JWT は署名か暗号化またはその両方を行うことができます。ほとんど場合、OAuth 2.0 フレームワークの参加者は、この目的で JWKS (JSON Web Key Set) のペアを使用します。JWKS の任意のペアにおいて、一方の JWKS は秘密鍵で、(アルゴリズムあたり) 必要なすべての秘密鍵だけでなく、対称鍵として使用するクライアント秘密鍵も含んでいます。この JWKS は共有されません。他方の JWKS には、対応する公開鍵が含まれていて、公開されて利用できます。

それぞれの参加者は、秘密の JWKS を持っていて、対応する公開の JWKS を他の参加者に提供します。秘密の JWKS の所有者は、その JWKS を使用して、送信 JWT に署名し、受信 JWT を解読します。他の参加者は、以下の図に示すように、対応する公開の JWKS を使用して送信 JWT を暗号化し、受信 JWT の署名を確認します。



1.4 付与タイプと付与フロー

OAuth 2.0 フレームワークでは、承認サーバが承認の要求を処理する方法が付与タイプによって指定されます。クライアントは、承認サーバへの最初の要求に付与タイプを指定します。OAuth 2.0 仕様には 4 つの付与タイプと、追加のタイプを定義する拡張メカニズムが記載されています。通常、それぞれの付与タイプは異なる全体フローに相当します。

4 つの付与タイプは以下のとおりです。

- 承認コード – この付与タイプは対応するサーバを持つクライアント・アプリケーションでのみ使用することができます。この付与タイプでは、ユーザがユーザ名とパスワードを入力するログイン・ページが承認サーバによって表示されます。これらの情報はクライアントと共有されることはありません。ユーザ名とパスワードが有効なユーザと一致すると (要求の他の要素が適切な場合)、承認サーバは最初に承認コードを発行し、これをクライアントに返します。クライアントは、この承認コードを使用してアクセス・トークンを取得します。

  承認コードの要求はブラウザに表示され、その応答も同様です。一方、アクセス・トークンの要求はサーバ間のやりとりであり、その応答も同様です。したがって、アクセス・トークンはブラウザに表示されることはありません。

  Proof Key for Code Exchange (PKCE) は、横取りした承認コードで悪意のあるアクターがアクセス・トークンを取得できないようにする、承認コード・フローの拡張です。PKCE では、クライアントの承認コードの要求に、追加のシークレット値が含められます。承認サーバは承認コードの発行時に、このシークレットを保存します。以降のクライアントの承認コードをアクセス・トークンに交換する要求には、元のシークレットを含める必要があります。承認コードを横取りしたアクターはこのシークレットを知らないため、アクターがアクセス・トークンを取得するのを防ぐことができます。
- 暗黙 – 前述の付与タイプと同様に、承認サーバはログイン・ページを表示しますが、クライアントはユーザの資格情報にアクセスすることはできません。ただし、この暗黙付与タイプでは、クライアントがアクセス・トークンを直接要求して受け取ります。この付与タイプは、JavaScript クライアントやモバイル・アプリケーションなどの純正のクライアント・アプリケーションで役立ちます。
- リソース所有者のパスワード資格情報 – この付与タイプでは、クライアントがユーザ名とパスワードを入力するようにユーザに要求し、それにより得られた資格情報を使用して承認サーバからアクセス・トークンを取得します。信頼できるアプリケーションだけがこの付与タイプに適しています。
- クライアント資格情報付与タイプ – この付与タイプではユーザ・コンテキストは存在せず、クライアント・アプリケーションは人の介入なしに処理を行います。クライアントは、クライアント ID とクライアント秘密鍵を使用して承認サーバからアクセス・トークンを取得します。

RFC 7523 では、追加の付与タイプである JWT 承認について説明しています。この付与タイプでは、JSON Web トークン (JWT) ベアラー・トークンを使用して、OAuth 2.0 アクセス・トークンを要求し、クライアントを認証します。InterSystems IRIS は、OAuth 2.0 仕様の 4 つに加えて、この付与タイプをサポートしています。

通常、OAuth 2.0 フレームワークでは、すべての HTTP 要求が SSL/TLS で保護されています。

さらに、クライアントが承認サーバに要求を送信するときは、その要求を認証する必要があります。OAuth 2.0 仕様に、クライアントが要求を認証する方法が記述されています。

1.5 スコープ

承認サーバは、クライアントが `scope` 要求パラメータを使用してアクセス要求のスコープを指定するのを許可します。次に、承認サーバは `scope` 応答パラメータを使用して、発行したアクセス・トークンのスコープをクライアントに通知します。

OpenID Connect は OAuth 2.0 承認プロセスの拡張機能です。認証を要求するために、クライアントは承認サーバへの要求に `openid` スコープ値を組み込みます。承認サーバは、認証に関する情報を ID トークンと呼ばれる JWT で返します。ID トークンには、OpenID Connect Core 仕様に記載されている特定のクレーム・セットが含まれています。

1.6 承認サーバのエンドポイント

承認サーバは、さまざまな種類の要求を処理できる、以下の URL またはエンドポイントの一部またはすべてを提供します。

エンドポイント	目的
承認エンドポイント	承認コードを返します (承認コード付与タイプにのみ適用)。
トークン・エンドポイント	アクセス・トークンを返します。
Userinfo エンドポイント	認証されたユーザに関するクレームを収めた JSON オブジェクトを返します (OpenID Connect にのみ適用)。
トークン・イントロスペクション・エンドポイント	アクセス・トークンを調べて決定したクレームを収めた JSON オブジェクトを返します。
トークン取り消しエンドポイント	トークンを取り消します。

2

InterSystems IRIS が OAuth 2.0 および OpenID Connect をサポートする方法

このページでは、OAuth 2.0 および OpenID Connect に対する InterSystems IRIS® のサポートについて説明します。

2.1 サポートされているシナリオ

OAuth 2.0 および OpenID connect に対する InterSystems IRIS のサポートにより、以下のいずれかまたはすべてを行うことができます。

- クライアントとして InterSystems IRIS Web アプリケーションを使用する
- リソース・サーバとして InterSystems IRIS Web アプリケーションを使用する
- 承認サーバとして InterSystems IRIS インスタンスを使用する

例えば、サードパーティ・テクノロジを採用した承認サーバのクライアントとして、InterSystems IRIS Web アプリケーションを使用することができます。また、InterSystems IRIS を基盤とする承認サーバとサードパーティのクライアントを使用できます。1 台以上のリソース・サーバを InterSystems IRIS または別のテクノロジで実装することもできます。

いずれの場合でも、承認サーバが最も複雑な要素であり、通常は最初に作成します。クライアントはそのあと作成します。一般にクライアントを作成するときは、サポートするスコープなど、承認サーバの機能と要件に関する知識を必要とします。

2.2 OAuth 2.0 および OpenID Connect に対する InterSystems IRIS のサポート

OAuth 2.0 および OpenID Connect に対する InterSystems IRIS のサポートは、以下の要素で構成されます。

- 管理ポータルの構成ページ。

  クライアント (またはリソース・サーバ) を構成する場合は、**[システム管理]**→**[セキュリティ]**→[OAuth 2.0]→**[クライアント構成]** のオプションを使用します。

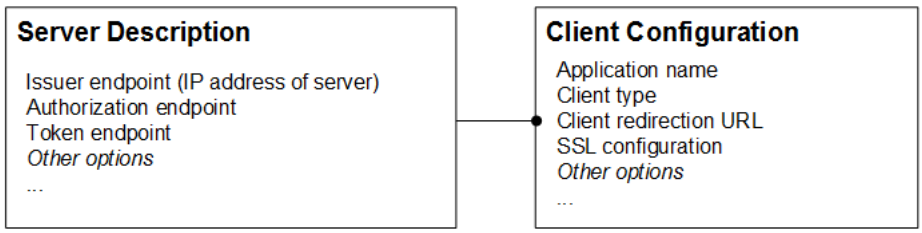
承認サーバを構成する場合は、**[システム管理]→[セキュリティ]→[OAuth 2.0]→[サーバ構成]** のオプションを使用します。

- **%SYS.OAuth2** パッケージのクラス。これらのクラスはクライアント API です。InterSystems IRIS Web アプリケーションを OAuth 2.0 クライアントとして定義すると、そのクライアントはこれらのクラスのメソッドを使用します。
- **%OAuth2** パッケージのクラス。OAuth 2.0 承認サーバとして InterSystems IRIS インスタンスを使用する場合は、**%OAuth2.Server** パッケージの 1 つ以上のクラスからサブクラスを作成することでサーバをカスタマイズします。**%OAuth2** の他のクラスは、コードが呼び出すユーティリティ・メソッドを提供します。
- **OAuth2** パッケージのクラス (**IRISSYS** データベース内)。このクラスには、InterSystems IRIS 内部で使用される永続クラスが含まれます。そのほとんどは無視できます。ただし、構成項目をプログラムで作成する場合は、このパッケージに含まれるクラスのサブセットを使用することになります。

以下のサブセクションに構成項目の概要を示します。

2.2.1 クライアントの構成項目

OAuth 2.0 クライアントとして機能している InterSystems IRIS インスタンス内で、サーバ記述 (承認サーバを記述) と クライアント構成 (クライアントを構成) という 2 つの結合された構成項目を指定クライアント・アプリケーションに対して定義する必要があります。特定のインスタンスでは、任意の数のサーバ記述を定義できます。下図が示すように、それぞれのサーバ記述は複数のクライアント構成があります。この図は、これらの構成項目に格納されている情報の一部も示しています。

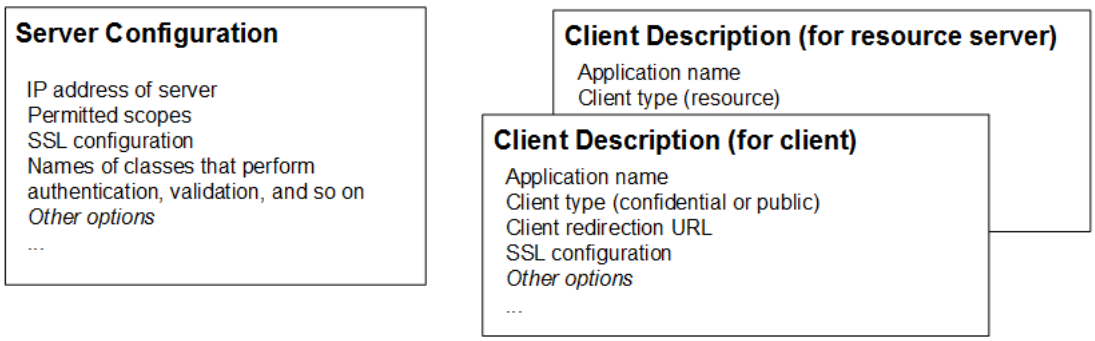


このアーキテクチャにより、同じ承認サーバを使用する複数のクライアント構成を定義することが可能になり、承認サーバの詳細を繰り返し定義する必要がなくなるため構成を簡素化することができます。

"OAuth 2.0 クライアントとしての InterSystems IRIS Web アプリケーションの使用法" に記載されているように、これらの項目は管理ポータルで作成できます。また、"プログラムによる構成項目の作成方法" に記載されているように、プログラムで構成項目を作成することもできます。

2.2.2 サーバの構成項目

OAuth 2.0 承認サーバとして機能している InterSystems IRIS インスタンス内で、サーバ構成 (承認サーバを構成) と複数のクライアント記述を定義する必要があります。下図は、これらの構成項目に格納されている情報の一部を示しています。



特定の InterSystems IRIS インスタンスは、1 つのサーバ構成と複数のクライアント記述を持つことができます。クライアント・アプリケーションごとに 1 つのクライアント記述が必要になります。クライアント記述は、承認サーバのエンドポイントを使用する各リソース・サーバでもそれぞれ 1 つ必要になります。承認サーバのエンドポイントを使用しないリソース・サーバでは、クライアント記述を作成する必要はありません。

"OAuth 2.0 承認サーバとしての InterSystems IRIS の使用法" に記載されているように、これらの項目は管理ポータルで作成できます。また、"プログラムによる構成項目の作成方法" に記載されているように、プログラムで構成項目を作成することもできます。

2.3 InterSystems IRIS でサポートされる標準規格

このセクションでは、OAuth 2.0 および OpenID Connect に対して InterSystems IRIS がサポートしている標準規格を示します。

- The OAuth 2.0 Authorization Framework (RFC 6749) ー https://datatracker.ietf.org/doc/rfc6749 を参照
- The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC 6750) ー https://datatracker.ietf.org/doc/rfc6750 を参照
- OAuth 2.0 Token Revocation (RFC 7009) ー https://datatracker.ietf.org/doc/rfc7009 を参照
- JSON Web Token (JWT) (RFC 7519) ー https://datatracker.ietf.org/doc/rfc7519 を参照
- OAuth 2.0 Token Introspection (RFC 7662) ー https://datatracker.ietf.org/doc/rfc7662 を参照
- OpenID Connect Core 1.0 ー http://openid.net/specs/openid-connect-core-1_0.html を参照
- OAuth 2.0 フォーム送信レスポンス・モード ー http://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html を参照
- JSON Web Key (JWK) (RFC 7517) ー https://datatracker.ietf.org/doc/rfc7517 を参照
- OpenID Connect Discovery 1.0 ー https://openid.net/specs/openid-connect-discovery-1_0.html を参照
- OpenID Connect ダイナミック・クライアント登録 ー http://openid.net/specs/openid-connect-registration-1_0-19.html を参照
- OAuth 2.0 クライアント認証および承認付与用 JSON Web トークン (JWT) プロファイル (RFC 7523) ー https://tools.ietf.org/html/rfc7523 を参照してください。
- Proof Key for Code Exchange (RFC 7636) ー https://tools.ietf.org/html/rfc7636 を参照してください。

3

OAuth 2.0 クライアントとしての InterSystems IRIS Web アプリケーションの使用法

このページでは、OAuth 2.0 フレームワークを使用するクライアント・アプリケーションとして InterSystems IRIS® の Web アプリケーションを使用する方法について説明します。

このページでは、InterSystems IRIS Web アプリケーションが Web サーバ/クライアント・アプリケーションのクライアントとして機能し、承認コード付与タイプを使用するシナリオについて主に説明します。その他の付与タイプとバリエーションの詳細は、バリエーションのセクションを参照してください。

注釈　OAuth 2.0 クライアントが Active Directory フェデレーション・サービス (ADFS) 承認サーバと通信する際、クライアント・コードで承認エンドポイントに特別なキーと値のペアを付加する必要があります。詳細は、"メソッドの詳細" の "`GetAuthorizationCodeEndpoint()`" の説明を参照してください。

3.1 InterSystems IRIS クライアントの前提条件

このページに記載されているタスクを開始する前に、以下のものが利用可能であることを確認します。

- OAuth 2.0 承認サーバ。後で、このサーバの具体的な詳細情報が必要になります。以下に示す詳細項目の一部は、InterSystems IRIS 内でクライアントを構成するときに適用されます。
  - 承認サーバの場所 (発行者エンドポイント)
  - 承認エンドポイントの場所
  - トークン・エンドポイントの場所
  - Userinfo エンドポイントの場所 (サポートされている場合。OpenID Connect Core を参照)
  - トークン・イントロスペクション・エンドポイントの場所 (サポートされている場合。RFC 7662 を参照)
  - トークン取り消しエンドポイントの場所 (サポートされている場合。RFC 7009 を参照)
  - 承認サーバによるダイナミック登録のサポートの有無

  以下に示すその他の詳細情報は、クライアント・コードの作成時に必要になります。
  - このサーバがサポートする付与タイプ
  - このサーバがサポートするスコープ。例えば、OpenID Connect Core で定義されている特別なスコープである `openid` および `profile` をサーバがサポートすることもあれば、サポートしないこともあります。

  - このサーバに対して行われる要求に関するその他の要件

- 承認サーバがダイナミック・クライアント登録をサポートしない場合、InterSystems IRIS アプリケーションを OAuth 2.0 承認サーバのクライアントとして登録する必要があります。また、このクライアントのクライアント ID とクライアント秘密鍵が必要です。詳細は、承認サーバの実装環境によって異なります。(サーバがダイナミック登録をサポートする場合、このページの説明に従って構成するときにクライアントを登録できます。)

3.2 構成要件

OAuth 2.0 クライアントとして InterSystems IRIS Web アプリケーションを使用するには、次の構成タスクを実行します。

- InterSystems IRIS にサービスを提供している Web サーバでは、その Web サーバが SSL を使用するように構成します。SSL を使用するように Web サーバを構成する方法は、このドキュメントの対象外であるため、ここでは取り上げません。
- クライアントが使用する InterSystems IRIS の SSL 構成を作成します。

  これはクライアント SSL 構成です。証明書は不要です。この構成は Web サーバへの接続に使用されます。この接続を通じて、クライアントは、承認サーバと通信してアクセス・トークンを取得し、Userinfo エンドポイントの呼び出し、イントロスペクション・エンドポイントの呼び出しなどを行います。

  SSL 構成の作成の詳細は、インターシステムズの "TLS ガイド" を参照してください。

  それぞれの SSL 構成には一意の名前が付いています。参照用に、このドキュメントではこの SSL 構成を `sslconfig` と呼びますが、一意の名前を任意に付けることができます。
- クライアントの OAuth 2.0 構成項目を作成します。そのためには、サブセクションの説明に従って最初にサーバ記述を作成し、次にクライアント構成を作成します。

  両項目の必要なオプションを管理ポータルで見つけるために、**[システム管理]**→**[セキュリティ]**→**[OAuth 2.0]**→**[クライアント構成]** を選択します。このページには、(承認サーバとして使用されているマシン以外の) クライアント・マシンで OAuth 2.0 構成を作成するときに必要なオプションが表示されます。

  クライアント・マシンでは、**[システム管理]**→**[セキュリティ]**→**[OAuth 2.0]**→**[サーバ構成]** のメニューを使用しないでください。

3.2.1 サーバ記述の作成 (Discovery の使用)

1. 管理ポータルで **[システム管理]**→**[セキュリティ]**→**[OAuth 2.0]**→**[クライアント構成]** を選択します。

   このインスタンスで利用可能なサーバ記述を示すページが表示されます。どの行でも、**[発行者エンドポイント]** 列はサーバ記述の発行者エンドポイントを示します。**[クライアント数]** 列は、指定されたサーバ記述に関連付けられたクライアント構成の数を示します。最後の列では **[クライアント構成]** リンクにより、関連付けられているクライアント構成の作成、表示、編集、削除を行うことができます。
2. **[サーバ構成の作成]** を選択します。

   管理ポータルに、サーバ記述の詳細を入力できる新たなページが表示されます。
3. 以下の詳細を指定します。
   - **[発行者エンドポイント]** (必須) – 承認サーバの識別に使用するエンドポイントの URL を入力します。
   - **[SSL/TLS 構成]** (必須) – ダイナミック・クライアント登録要求時に使用する SSL/TLS 構成を選択します。
   - **[登録アクセス・トークン]** – オプションで、ダイナミック・クライアント登録要求を承認するためのベアラー・トークンとして使用する初期登録アクセス・トークンを入力します。

4. **[検出と保存]** を選択します。

   次に InterSystems IRIS は、指定された承認サーバと通信し、サーバ記述に必要な情報を取得して、情報を保存します。

   管理ポータルにサーバ記述の一覧が再度表示されます。

3.2.1.1 サーバ記述の手動による作成 (Discovery なし)

(Discovery を使用せずに) 手動でサーバ記述を作成するには、まずサーバ記述ページを表示して (上記の手順 1 および 2)、次に **[手動]** を選択します。これによって、以下のようにページに多数のオプション・セットが表示されます。

- **[発行者エンドポイント]** (必須) – 承認サーバの識別に使用するエンドポイントの URL を入力します。
- **[承認エンドポイント]** (必須) – 承認サーバに承認コードを要求するときに使用するエンドポイント URL を入力します。
- **[トークン・エンドポイント]** (必須) – 承認サーバにアクセス・トークンを要求するときに使用するエンドポイント URL を入力します。
- **[Userinfo エンドポイント]** (必須) – 承認サーバのアクセス・トークンを使用して承認の Userinfo 要求を行うときに使用するエンドポイント URL を入力します。
- **[トークン・イントロスペクション・エンドポイント]** – `client_id` と `client_secret` を使用して承認のトークン・イントロスペクション要求を行うときに使用するエンドポイント URL を入力します。RFC 7662 を参照してください。
- **[トークン取り消しエンドポイント]** – `client_id` と `client_secret` を使用して承認のトークン取り消し要求を行うときに使用するエンドポイント URL を入力します。RFC 7009 を参照してください。
- **[JSON Web Token (JWT) の設定]** – JWT のシグニチャの検証および解読を承認サーバからクライアントが行うために使用する公開鍵のソースを指定します。

  既定では、承認サーバは JWKS (JSON Web Key Set) のペアを生成します。一方の JWKS は秘密鍵で、(アルゴリズムあたり) 必要なすべての秘密鍵だけでなく、対称鍵として使用するクライアント秘密鍵も含んでいます。この JWKS は共有されません。他方の JWKS には、対応する公開鍵が含まれていて、公開されて利用できます。また、サーバ記述を作成するプロセスでも、JWT のシグニチャの検証と暗号化に使用するために、承認サーバからクライアントに公開の JWKS がコピーされます。

  - **[URL から JWKS]** – 公開の JWKS を指す URL を指定した後、InterSystems IRIS に JWKS をロードします。
  - **[ファイルから JWKS]** – 公開の JWKS を含むファイルを選択し、そのファイルを InterSystems IRIS にロードします。
  - **[X509 証明書]** – 詳細は、"証明書と JWT (JSON Web Token)" の "OAuth 2.0 承認サーバの証明書の使用" を参照してください。

  これらのオプションのいずれかにアクセスするには、まず **[ダイナミック登録以外のソース]** を選択します。

これらの値を指定し、**[保存]** を選択します。

3.2.2 クライアントの構成および動的な登録

ここでは、クライアントの構成を作成し、クライアントを動的に登録する方法について説明します。

1. 管理ポータルで **[システム管理]**→**[セキュリティ]**→**[OAuth 2.0]**→**[クライアント構成]** を選択します。

   管理ポータルにサーバ記述の一覧が表示されます。

2. このクライアント構成を関連付けるサーバ記述の行で **[クライアント構成]** リンクをクリックします。

管理ポータルに、サーバ記述に関連付けられているクライアント構成の一覧が表示されます。これは、最初は空のリストです。

3. **[クライアント構成の作成]** をクリックします。

   管理ポータルに、詳細を入力できる新たなページが表示されます。

4. **[一般]** タブで、以下の詳細を指定します。

   - **[アプリケーション名]** ― アプリケーションの短い名前を指定します。
   - **[クライアント名]** ― エンド・ユーザに表示するクライアント名を指定します。
   - **[説明]** ― アプリケーションの説明を入力します (オプション)。
   - **[有効]** ― このアプリケーションが使用されないようにする場合は、このチェック・ボックスのチェックを外すこともできます。
   - **[クライアント・タイプ]** ― 以下のいずれかを選択します。
     - **[機密]** ― RFC 6749 に従って、クライアントを機密クライアントに指定します。

       このページでは、クライアントが承認コード付与タイプを使用するシナリオについて主に説明します。このシナリオでは、**[クライアント・タイプ]** に **[機密]** を指定します。その他の付与タイプは、"バリエーション" を参照してください。
     - **[パブリック]** ― RFC 6749 に従って、クライアントをパブリック・クライアントに指定します。
     - **[リソース・サーバ]** ― クライアントを専用のリソース・サーバに指定します。
   - **[SSL/TLS 構成]** ― クライアントで使用するために作成した SSL 構成を選択します (例:`sslconfig`)。
   - **[応答を受信するために承認サーバに対して指定されるクライアント URL]** ― InterSystems IRIS OAuth 2.0 クライアントで必要とされる内部宛先の URL を指定します。この宛先でアクセス・トークンが保存され、ブラウザは元のクライアント・アプリケーションへリダイレクトされます。

     この URL を指定するには、次のオプションの値を入力します。
     - **[ホスト名]** ― 承認サーバのホスト名または IP アドレスを指定します。
     - **[ポート]** ― Web ゲートウェイ構成の変更に対応するために必要な場合は、この値を指定します。
     - **[接頭語]** ― Web ゲートウェイ構成の変更に対応するために必要な場合は、この値を指定します。

     指定した URL は以下の形式をとります。

     ```
     https://hostname:port/prefix/csp/sys/oauth2/OAuth2.Response.cls
     ```

     [ポート] の指定を省略すると、そのコロンが省略されます。同様に、**[接頭語]** の指定を省略すると、`hostname:port` と `csp` の間のスラッシュが 1 つだけになります。(さらに、**[TLS/SSL を使用する]** オプションのチェックを外すと、URL は `https` ではなく `http` で始まります。)
   - **[TLS/SSL を使用する]** ― リダイレクト・ページを開くときに TLS/SSL を使用しない正当な理由がある場合を除き、このオプションを選択します。
   - **[フロントチャネルログアウト URL]** ― 必要に応じて HTTP ベースのフロントチャネル・ログアウト URL を指定します。サーバはこの URL を登録し、それを使用してクライアント上のユーザをログアウトします。フロントチャネル・ログアウトをサポートしないクライアントを作成するには、この URL を空白のままにします。このフィールドの上のボックスには指定した URL が表示され、それに 'IRISLogout=end' が追記されます。

     注釈　InterSystems IRIS クライアントでフロントチャネル・ログアウトがサポートされるようにするには、クライアント・アプリケーションの **[セッション Cookie のスコープ]** を **[　]** に設定します。アプリケーションの設定の構成の詳細は、"アプリケーションの作成および編集" を参照してください。

- **[必要な付与タイプ]** – クライアントで限定使用する OAuth 2.0 付与タイプを指定します。
- **[認証タイプ]** – 承認サーバへの HTTP 要求で使用する認証タイプを選択します (RFC 6749 または OpenID Connect Core のセクション 9 で規定)。**[なし]**、**[基本]**、**[フォームエンコードされた本文]**、**[クライアント秘密鍵 JWT]**、または **[秘密鍵 JWT]** のいずれかを選択します。
- **[認証サーバがログアウトURLを呼び出す際に iss および sid クエリパラメータが必要]** – 承認サーバからフロントチャネル・ログアウト URL が呼び出されるときにクエリ・パラメータとして iss (発行者) と sid (セッション ID) を必要とするには、このオプションを選択します。
- **[認証署名アルゴリズム]** – トークン・エンドポイントでこのクライアントを認証するために使用される JWT の署名に必要なアルゴリズムを選択します ([認証タイプ] が **[クライアント秘密鍵 JWT]** または **[秘密鍵 JWT]** である場合)。オプションを選択しない場合、OpenID プロバイダおよび Relying Party でサポートされる任意のアルゴリズムが使用されます。

5. **[クライアント情報]** タブで、以下の詳細を指定します。
   - **[ロゴ URL]** – クライアント・アプリケーションのロゴの URL です。
   - **[クライアント・ホームページ URL]** – クライアント・アプリケーションのホーム・ページの URL です。
   - **[ポリシー URL]** – クライアント・アプリケーションのポリシー・ドキュメントの URL です。
   - **[サービス条件の URL]** – クライアント・アプリケーションのサービス条件ドキュメントの URL です。
   - **[既定のスコープ]** – アクセス・トークン要求の既定のスコープを空白で区切られたリストで指定します。この既定値は、承認サーバで許可されているスコープと一致している必要があります。
   - **[連絡先電子メール]** – クライアント・アプリケーションの責任者への連絡に使用する適切な電子メール・アドレスをコンマで区切ったリストです。
   - **[既定の最長経過時間]** – 既定の最長認証経過時間 (秒単位) を指定します。このオプションを指定した場合、最長認証経過時間に達したとき、エンド・ユーザをアクティブに再認証する必要があります。max_age 要求パラメータは、この既定値をオーバーライドします。このオプションを省略した場合に、既定で最長認証経過時間は設定されません。

6. **[JWT 設定]** タブで、以下の詳細を指定します。
   - **[X509 資格情報から JWT 設定を作成]** – 証明書に関連付けた秘密鍵を署名と暗号化で使用する場合は、このオプションを選択します。この場合は、"証明書と JWT (JSON Web Token)" の "OAuth 2.0 クライアントの証明書の使用" も参照してください。

     注釈　インターシステムズでは、**[X509 資格情報から JWT 設定を作成]** オプションが使用されることはまれで、次に説明する既定の動作が代わりに使用されると考えています。

     このオプションのチェックを外したままにすると、システムは JWKS (JSON Web Key Set) のペアを生成します。一方の JWKS は秘密鍵で、(アルゴリズムあたり) 必要なすべての秘密鍵だけでなく、対称鍵として使用するクライアント秘密鍵も含んでいます。この JWKS は共有されません。他方の JWKS には、対応する公開鍵が含まれていて、公開されて利用できます。ダイナミック登録プロセスでは、公開の JWKS を承認サーバにもコピーし、承認サーバがこのクライアントから JWT の暗号化およびシグニチャの検証を実行できるようにします。
   - **[署名アルゴリズム]** – 署名済み JWT の作成に使用する署名アルゴリズムを選択します。JWT を署名しない場合は空白のままにします。
   - **[暗号化アルゴリズム]** – 暗号化された JWT の作成に使用する暗号化アルゴリズムを選択します。JWT を暗号化しない場合は空白のままにします。値を選択する場合は、**[キー・アルゴリズム]** も指定する必要があります。
   - **[キー・アルゴリズム]** – 暗号化された JWT の作成に使用するキー管理アルゴリズムを選択します。JWT を暗号化しない場合は空白のままにします。

7. 承認サーバがダイナミック登録をサポートする場合は、入力したすべてのデータを再確認した後、**[ダイナミック登録と保存]** をクリックします。次に、InterSystems IRIS は承認サーバを接続し、クライアントを登録して、クライアント ID とクライアント秘密鍵を取得します。

   承認サーバがダイナミック登録をサポートしない場合については、この後のサブセクションを参照してください。

3.2.2.1 クライアントの構成 (ダイナミック登録なし)

承認サーバがダイナミック登録をサポートしない場合は、上記の最後の手順に代わって以下の手順を実行します。

1. **[クライアント資格情報]** タブを選択し、次の詳細を指定します。
   - **[クライアント ID]** – 承認サーバが提供したクライアント ID を入力します。
   - **[クライアント秘密鍵]** – 承認サーバが提供したクライアント秘密鍵を入力します。**[クライアント・タイプ]** が **[機密]** の場合は、この値が必須となります。

     このページでは、クライアントが承認コード付与タイプを使用するシナリオについて主に説明します。このシナリオでは、**[クライアント秘密鍵]** の値を指定します。その他の付与タイプは、"バリエーション" を参照してください。

   [クライアント ID 発行日時]、**[クライアント秘密鍵有効期限]**、および **[登録クライアント URI]** の値を入力しないでください。
2. **[保存]** を選択します。

3.3 コード要件の概要

注釈　このセクションでは、トークンの要求時にクライアントが承認コード付与タイプを使用するときに必要とされるコードについて説明します。その他の付与タイプは、"バリエーション" を参照してください。

InterSystems IRIS Web アプリケーションが OAuth 2.0 クライアントとして機能するには、この Web アプリケーションで以下のようなロジックを使用する必要があります。

1. アクセス・トークンと ID トークン (必要な場合) を取得します。"トークンの取得" を参照してください。
2. アクセス・トークンと (必要に応じて) ID トークンを検証し、要求されたリソースの使用に必要な権限をユーザが保有しているかどうかを確認します。"トークンの検証" を参照してください。
3. 権限が適切な場合は、"HTTP 要求へのアクセス・トークンの追加" の説明に従ってリソース・サーバを呼び出します。

以降のセクションでこれらの手順に関する情報を提供します。

3.4 トークンの取得

注釈　このセクションでは、トークンの要求時にクライアントが承認コード付与タイプを使用するときに必要とされるコードに関する情報を提供します。既定では、この承認コード付与タイプには、Proof Key for Code Exchange (PKCE) 拡張が含まれます。その他の付与タイプおよび PKCE なしの承認コードについては、"バリエーション" を参照してください。

トークンを取得するには、以下のようなトークン取得の手順を使用します。サブセクションでは、ここで使用するメソッドを詳しく説明します。

1. %SYS.OAuth2.AccessToken クラスの IsAuthorized() メソッドを呼び出します。その際、最初にアクセス・トークンの適切なスコープ (1 つ以上) を決定する必要があります。

   以下に例を示します。

   ObjectScript

   ```
   set myscopes="openid profile scope1 scope2"
   set isAuth=##class(%SYS.OAuth2.AccessToken).IsAuthorized("myclient",,myscopes,
                   .accessToken,.idtoken,.responseProperties,.error)
   ```

   このメソッドは、アクセス・トークンが既にローカルに保存されているかどうかを確認します。

2. error 引数がエラーを返したかどうかを確認し、返した場合はそのエラーを適切に処理します。この引数にエラーが含まれる場合、関数 $ISOBJECT() は 1 を返します。それ以外の場合、$ISOBJECT() は 0 を返します。

   ObjectScript

   ```
      if $isobject(error) {
     //error handling here
   }
   ```

3. IsAuthorized() が 1 を返す場合は "トークンの検証" へ進みます。

4. それ以外の場合は、%SYS.OAuth2.Authorization クラスの GetAuthorizationCodeEndpoint() メソッドを呼び出します。その際、以下の情報を必要とします。

   - アクセス・トークンを返した後に承認サーバがリダイレクトされる完全な URL。これは、クライアントのリダイレクト・ページです (元のページと同じにすることも、別のページにすることもできます)。
   - 要求のスコープ (1 つ以上)。
   - 要求に含まれるパラメータ。例えば、場合によっては `claims` パラメータを渡す必要があります。

   以下に例を示します。

   ObjectScript

   ```
   set scope="openid profile scope1 scope2"
   set redirect="https://localhost/csp/openid/SampleClientResult.csp"

   set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint("myclient",
         scope,redirect,.properties,.isAuthorized,.sc)
   if $$$ISERR(sc) {
     //error handling here
   }
   ```

   このメソッドは、InterSystems IRIS OAuth 2.0 クライアントが必要とする内部宛先の完全な URL (クエリ・パラメータを含む) を返します。

   このメソッドに対する Proof Key for Code Exchange (PKCE) の既定の動作を変更するには、"メソッドの詳細" で `properties` 引数の詳細を参照してください。

5. GetAuthorizationCodeEndpoint() が返した URL を開くオプション (ボタンなど) を提供することで、ユーザは要求を承認することができます。

   InterSystems IRIS は、ユーザには表示されないこの内部 URL で承認コードを取得し、それをアクセス・トークンに交換し、クライアントのリダイレクト・ページにブラウザをリダイレクトします。

3.4.1 メソッドの詳細

このサブセクションでは、前のサブセクションで使用したメソッドについて詳しく説明します。

IsAuthorized()

場所:このメソッドはクラス %SYS.OAuth2.AccessToken にあります。

```
ClassMethod IsAuthorized(applicationName As %String,
                         sessionId As %String,
                         scope As %String = "",
                         Output accessToken As %String,
                         Output IDToken As %String,
                         Output responseProperties,
                         Output error As %OAuth2.Error) As %Boolean
```

このクライアントおよびセッションのアクセス・トークンがローカルに保存され、さらにそのアクセス・トークンが scope 引数で提供されるすべてのスコープを承認する場合、このメソッドは 1 を返します。(このメソッドは IRISSYS データベースでアクセス・トークンを探します。これらのトークンは、有効期限が切れると自動的に削除されます。)

それ以外の場合、このメソッドは 0 を返します。

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- sessionId はセッション ID を指定します。既定のセッション (%session.SessionId) をオーバーライドする場合のみこれを指定します。
- scope は、スペースで区切られたスコープのリストです。例えば、`"openid profile scope1 scope2"` のような形式をとります。

  `openid` および `profile` は、OpenID Connect Core で定義されている特別なスコープです。
- 出力として返される accessToken はアクセス・トークンです (存在する場合)。
- 出力として返される IDToken は ID トークンです (存在する場合)。(これは、OpenId Connect を使用している場合にのみ適用されます。具体的には、要求でスコープ `openid` が使用された場合です。)ID トークンは JWT です。
- 出力として返される responseProperties は、応答のパラメータを含む多次元配列です。この配列の構造は以下のとおりです。

配列ノード	配列値
responseProperties(parametername)、ここで parametername はパラメータの名前です (`token_type`、`expires_in` など)。	指定されたパラメータの値。

- 出力として返される error は、空の文字列であるか (エラーがない場合)、エラー情報を含む %OAuth2.Error のインスタンスです (エラーがある場合)。

  %OAuth2.Error は、Error、ErrorDescription、ErrorUri という 3 つの文字列プロパティを持ちます。

GetAuthorizationCodeEndpoint()

場所:このメソッドはクラス %SYS.OAuth2.Authorization にあります。

```
ClassMethod GetAuthorizationCodeEndpoint(applicationName As %String,
                                         scope As %String,
                                         redirectURL As %String,
                                         ByRef properties As %String,
                                         Output isAuthorized As %Boolean,
                                         Output sc As %Status,
                                         responseMode As %String
                                         sessionId As %String = "") As %String
```

このメソッドは、InterSystems IRIS が承認コードを要求するときに使用するローカルな内部ページの URL を、必要なすべてのクエリ・パラメータと共に返します。(このページはユーザには表示されません。)

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- scope は、スペースで区切られた、アクセスが要求されているスコープのリストです。例えば、"scope1 scope2 scope3" のような形式をとります。

  既定値は、指定された applicationName のクライアント構成によって決まります。
- redirectURL は、クライアントのリダイレクト・ページの完全な URL です。クライアントにアクセス・トークンを返した承認サーバは、このページにブラウザをリダイレクトします。
- 参照によって渡される properties は、要求に追加されるパラメータを含む多次元配列です。この配列は以下の構造を持つ必要があります。

配列ノード	配列値
properties (parametername)、ここで parametername はパラメータの名前です。	指定されたパラメータの値。この値は、スカラ値、ダイナミック・オブジェクトのインスタンス、ダイナミック・オブジェクトの UTF-8 のシリアル化されたエンコード形式のいずれかです。 JSON オブジェクトの値を持つパラメータを要求に含める場合は、ダイナミック・オブジェクトを使用します。その 1 つのシナリオが OpenID Connect で定義されている claims パラメータです。ダイナミック・オブジェクトの詳細は、"JSON の使用" を参照してください。 request または request_uri パラメータを使用する場合は、"JWT としての要求オブジェクトの受け渡し" セクションを参照してください。 code_verifier パラメータを使用して、承認サーバに送信するシークレット PKCE 値を変更できます。既定では、PKCE シークレット は、SHA-256 ハッシュを使用したランダムな 43 文字の文字列から生成されます。カスタム文字列からシークレットを生成するには (例えば、128 文字を使用する場合)、code_verifier パラメータにカスタム値を設定します。

- このクライアントおよびセッションのアクセス・トークンがローカルに保存されている場合、出力として返される isAuthorized の値は 1 になります (スコープはチェックされていません)。それ以外の場合、このパラメータの値は 0 になります。さきほど IsAuthorized() メソッドを呼び出したので、この出力引数を確認する必要はありません。
- 出力として返される sc には、このメソッドが設定したステータス・コードが含まれます。
- responseMode は、承認サーバの応答モードを指定します。このモードは "query" (既定値)、"fragment"、または "form_post" にできます。ほとんどの場合、既定値が適切な値となります。
- sessionId はセッション ID を指定します。既定のセッション (%session.SessionId) をオーバーライドする場合のみこれを指定します。

注釈　Active Directory フェデレーション・サービス (ADFS) サーバは、承認エンドポイント URL にキーと値のペア resource=urn:microsoft:userinfo が含まれることを期待しています。GetAuthorizationCodeEndpoint の properties 引数を使用して、このキーと値のペアをサーバ記述で定義された URL の末尾に追加できます。管理ポータルで承認エンドポイントを変更してこの情報を含めることは避けてください。以下のコードを使用して、GetAuthorizationCodeEndpoint メソッドを呼び出す前に、properties 引数を変更してください。

```
set properties("resource") = "urn:microsoft:userinfo"
set url = ##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(appName, scopes,
clientRedirectURI, .properties, .isAuthorized,.sc)
```

"バリエーション：OnPreHTTP 内でのリダイレクトの実行" も参照してください。

3.5 トークンの検証

アクセス・トークンと (必要に応じて) ID トークンを受け取ったクライアントは、追加のチェックを実行して、要求されたリソースの使用に必要な権限をユーザが保有しているかどうかを確認します。この検証を実行するために、クライアントは、ここで説明するメソッドを使用して追加情報を取得することができます。

ValidateIDToken()

場所：このメソッドはクラス %SYS.OAuth2.Validation にあります。

```
ClassMethod ValidateIDToken(applicationName As %String,
                            IDToken As %String,
                            accessToken As %String,
                            scope As %String,
                            aud As %String,
                            Output jsonObject As %RegisteredObject,
                            Output securityParameters As %String,
                            Output sc As %Status) As %Boolean
```

このメソッドは署名済みの OpenID Connect ID トークン (IDToken) を検証し、オブジェクト (jsonObject) を作成して ID トークンのプロパティを格納します。ID トークンを検証するために、このメソッドは、対象者 (aud が指定されている場合)、エンドポイント (サーバ記述で指定されているものとの一致が必要)、スコープ (scope が指定されている場合)、シグニチャを確認します。このメソッドは、ID トークンの有効期限が切れていないことも確認します。

さらに、ID トークンの at_hash プロパティに基づいてアクセス・トークン (accessToken) も検証します。

このメソッドは ID トークンが有効な場合は 1 を返し、それ以外の場合は 0 を返します。さらに、複数の引数を出力として返します。

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- IDToken は ID トークンです。
- accessToken はアクセス・トークンです。
- scope は、スペースで区切られたスコープのリストです。例えば、"scope1 scope2 scope3" のような形式をとります。
- aud は、このトークンを使用する対象者を指定します。トークンに関連付けられた aud プロパティがある場合 (通常、対象者はトークンの要求時に指定されているため)、aud はトークンの対象者と一致します。aud が指定されていない場合、対象者のチェックは行われません。

- 出力として返される jsonObject は、IDToken のプロパティを含むダイナミック・オブジェクトです。ID トークンは JWT です。ダイナミック・オブジェクトの詳細は、"JSON の使用" を参照してください。
- 出力として返される securityParameters は、ヘッダから取得されたセキュリティ情報を含む多次元配列です。このセキュリティ情報は、シグニチャや復号化の検証で必要に応じて追加使用されます。ValidateJWT() の securityParameters 引数を参照してください。
- 出力として返される sc には、このメソッドが設定したステータス・コードが含まれます。

このメソッドが成功 (1) を返す場合は、jsonObject を検証し、必要に応じて含まれているクレームを使用して、要求されたリソースへのアクセスを許可するかどうかを決定します。必要に応じて securityParameters を使用します。

GetUserinfo()

場所:このメソッドはクラス %SYS.OAuth2.AccessToken にあります。

```
ClassMethod GetUserinfo(applicationName As %String,
                        accessToken As %String,
                        IDTokenObject As %RegisteredObject,
                        Output jsonObject As %RegisteredObject,
                        Output securityParameters As %String) As %Status
```

このメソッドはアクセス・トークンを Userinfo エンドポイントへ送信し、クレームを含む応答を受信し、このエンドポイントが返したクレームを含むオブジェクト (jsonObject) を作成します。応答が JWT を返すとその応答は復号化され、署名が確認されてから jsonObject が作成されます。引数 IDTokenObject が指定されている場合、このメソッドは、Userinfo エンドポイントの sub クレームが IDTokenObject の sub クレームと一致していることも検証します。

要求は、指定されたアクセス・トークンによって承認されます。

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- accessToken はアクセス・トークンです。
- IDTokenObject (オプション) は、ID トークンを含むダイナミック・オブジェクトです。ダイナミック・オブジェクトの詳細は、"JSON の使用" を参照してください。
- 出力として返される jsonObject は、Userinfo エンドポイントが返したクレームを含むダイナミック・オブジェクトです。
- 出力として返される securityParameters は、ヘッダから取得されたセキュリティ情報を含む多次元配列です。このセキュリティ情報は、シグニチャや復号化の検証で必要に応じて追加使用されます。ValidateJWT() の securityParameters 引数を参照してください。

このメソッドが成功 (1) を返す場合は、jsonObject を検証し、必要に応じて含まれているクレームを使用して、要求されたリソースへのアクセスを許可するかどうかを決定します。必要に応じて securityParameters を使用します。

3.6 HTTP 要求へのアクセス・トークンの追加

アクセス・トークンを受信して検証したクライアント・アプリケーションは、リソース・サーバへの HTTP 要求を作成できます。アプリケーションによっては、この HTTP 要求でアクセス・トークンを必要とすることもあります。

(ベアラー・トークン HTTP 承認ヘッダとして) アクセス・トークンを HTTP 要求へ追加するには、以下の手順を実行します。

1. %Net.HttpRequest のインスタンスを作成し、必要に応じてプロパティを設定します。

   このクラスの詳細は、"インターネット・ユーティリティの使用法" の" HTTP 要求の送信" を参照してください。

2. %SYS.OAuth2.AccessToken の AddAccessToken() メソッドを呼び出します。このメソッドにより、アクセス・トークンが HTTP 要求に追加されます。このメソッドは、以下のとおりです。

```
ClassMethod AddAccessToken(httpRequest As %Net.HttpRequest,
                           type As %String = "header",
                           sslConfiguration As %String,
                           applicationName As %String,
                           sessionId As %String) As %Status
```

   このメソッドは、指定されたアプリケーションおよびセッションに関連付けられているベアラー・アクセス・トークンを RFC 6750 で定義されているリソース・サーバへの要求に追加します。引数は以下のとおりです。

   - httpRequest は変更する %Net.HttpRequest のインスタンスです。
   - type は、HTTP 要求にアクセス・トークンを含める方法を指定します。
     - "header" ― ベアラー・トークン HTTP ヘッダを使用します。
     - "body" ― フォームエンコードされた本文を使用します。この場合の要求は、本文がフォームエンコードされた POST でなければなりません。
     - "query" ― クエリ・パラメータを使用します。
   - sslConfiguration は、この HTTP 要求で使用する InterSystems IRIS SSL 構成です。これを省略すると、InterSystems IRIS はクライアント構成に関連付けられている SSL 構成を使用します。
   - applicationName はクライアント・アプリケーションの名前です。
   - sessionId はセッション ID を指定します。既定のセッション (%session.SessionId) をオーバーライドする場合のみこれを指定します。

   このメソッドは、ステータス・コードを返すので、コードでこのステータス・コードを確認する必要があります。

3. "インターネット・ユーティリティの使用法" の "HTTP 要求の送信" の説明に従って HTTP 要求を送信します。そのためには、Get() や Put() などのメソッドを呼び出します。
4. 前の手順で返されたステータスを確認します。
5. HTTP 要求の HttpResponse プロパティとして利用できる HTTP 応答を必要に応じて検証します。

   "インターネット・ユーティリティの使用法" の "HTTP 要求の送信" を参照してください。

以下に例を示します。

ObjectScript

```
set httpRequest=##class(%Net.HttpRequest).%New()
// AddAccessToken adds the current access token to the request.
set sc=##class(%SYS.OAuth2.AccessToken).AddAccessToken(httpRequest,,"sslunittest",applicationName)
if $$$ISOK(sc) {
   set sc=httpRequest.Get("https://myresourceserver/csp/openid/openid.SampleResource.cls")
}
```

3.7 Web クライアントの代行認証の定義 (オプション)

必要に応じて、OAuth 2.0 クライアントとして使用する InterSystems IRIS Web クライアントの代行認証を定義できます。InterSystems IRIS には、これが可能な以下の 2 つの方法が用意されています。

- ZAUTHENTICATE ルーチンを作成して使用する方法。OAuth 2.0 で使用するために用意されているサンプルから作成します。クライアント・コードで %session.Login() を呼び出す必要もあります。
- カスタム・ログイン・ページを作成して使用する方法。ZAUTHENTICATE ルーチンを作成して使用する必要もあります (OAuth 2.0 で使用するために用意されている同じサンプルから作成)。ただし、クライアント・コードで %session.Login() を呼び出す必要はありません。

次のサブセクションで詳細を説明します。最後のサブセクションでは、ZAUTHENTICATE のサンプルについて説明します。

"REST サービスの作成" の "REST サービスの保護" にある "REST アプリケーションおよび OAuth 2.0" も参照してください。

重要　HealthShare® で認証を使用している場合は、インターシステムズが提供する ZAUTHENTICATE ルーチンを使用する必要があります。独自のルーチンは作成できません。

3.7.1 OAuth 2.0 クライアント用の ZAUTHENTICATE ルーチンの作成および使用

OAuth 2.0 クライアントとして使用する InterSystems IRIS Web クライアント用に ZAUTHENTICATE ルーチンを作成して使用するには、以下の手順をすべて実行します。

- クライアント・コードで、**%SYS.OAuth2.AccessToken** クラスの IsAuthorized() メソッドを呼び出してアクセス・トークンの取得に成功した後、%session 変数の Login() メソッドを呼び出します。ユーザ名として OAuth 2.0 のアプリケーション名を指定し、パスワードとして Web セッション ID を指定します。
- ZAUTHENTICATE ルーチンを作成します。このルーチンは、ロールや他のユーザ・プロパティの指定など、ユーザ・アカウントの基本的な設定を実行する必要があります。

  インターシステムズが提供するサンプル・ルーチン **OAUTH2.ZAUTHENTICATE.mac** をコピーして変更することができます。このルーチンは GitHub の Samples-Security サンプルに含まれています (https://github.com/intersystems/Samples-Security)。"InterSystems IRIS で使用するサンプルのダウンロード" で説明されているようにサンプル全体をダウンロードすることもできますが、単に GitHub でルーチンを開いて、その内容をコピーするほうが簡単です。

  ZAUTHENTICATE ルーチンを定義する場合、このルーチンは **%SYS** ネームスペース内にあり、**ZAUTHENTICATE** という名前である必要があります。"OAUTH2.ZAUTHENTICATE.mac サンプルの注意事項" を参照してください。代行認証に関する一般的な情報は、"代行 (ユーザ定義) 認証コードの作成" と "代行認証" を参照してください。
- [認証オプション] ページで、InterSystems IRIS インスタンスに対して代行認証を有効にします。

  この手順と次の手順の詳細は、"代行認証" を参照してください。
- 関連する Web アプリケーションに対して代行認証を有効にします。

3.7.2 OAuth 2.0 クライアント用のカスタム・ログイン・ページの作成および使用

OAuth 2.0 クライアントとして使用する InterSystems IRIS Web クライアント用にカスタム・ログイン・ページを作成して使用するには、以下の手順をすべて実行します。

- **%OAuth2.Login** のサブクラスを作成します。サブクラスで以下を実行します。
  - アプリケーション名、スコープ・リスト、およびレスポンス・モード (オプション) を指定します。以下のいずれかまたは両方を実行してこれらの項目を指定できます。
    - **%OAuth2.Login** のサブクラスのパラメータを指定。
    - DefineParameters() クラス・メソッドのオーバーライド。パラメータの指定と対照的に、このテクニックでは実行中にこれらの値を設定できます。

これらのパラメータは、以下のとおりです。

- APPLICATION ― これは、ログインするアプリケーションのアプリケーション名である必要があります。
- SCOPE ― これは、アクセス・トークン要求に使用するスコープ・リストを指定します。これは、空白で区切られた文字列リストである必要があります。
- RESPONSEMODE ― これは、レスポンスのモードを指定します。可能な値は "query" (既定値)、"fragment"、または "form_post" です。

DefineParameters() クラス・メソッドには以下のシグニチャがあります。

```
ClassMethod DefineParameters(Output application As %String, Output scope As %String, Output
responseMode As %String)
```

このメソッドは、アプリケーション名、スコープ・リスト、およびレスポンス・モードを出力引数として返します。このメソッドの既定の実装では、APPLICATION、SCOPE、および RESPONSEMODE の各クラス・パラメータの値が返されます。

– %OAuth2.Login のサブクラスで、GetAccessTokenAuthorizationCode() 呼び出しのプロパティ・リストも指定します。そのためには、DefineProperties() クラス・メソッドをオーバーライドします。このメソッドには、以下のシグニチャがあります。

```
ClassMethod DefineProperties(Output properties As %String)
```

このメソッドは、properties 配列を出力として返します。これは、トークン要求に含める追加プロパティを指定するローカル配列です。properties 配列は以下の形式をとります。

ノード	値
properties(name)、ここで name はパラメータの名前です。	指定されたパラメータの値。

JSON オブジェクトである要求パラメータを追加するには、%DynamicObject のインスタンスであるプロパティ要素を作成します。または、UTF-8 でエンコードされたシリアル化オブジェクトである文字列を作成します。

request または request_uri 要求パラメータを追加するには、%SYS.OAuth2.Request クラスを使用して JWT を作成します。次に必要に応じて、properties("request") を JWT と等しい値に設定するかまたは properties("request_uri") を JWT の URL と同じ値に設定します。

- 関連 Web アプリケーションを構成して、カスタム・ログイン・ページを使用します。
- 前のセクションの説明に従って、ZAUTHENTICATE ルーチンを作成して使用します。ただし、中黒で示された最初の項目を除きます。(このシナリオでは、クライアント・コードで %session.Login() を呼び出す必要はありません。)

3.7.3 OAUTH2.ZAUTHENTICATE.mac サンプルの注意事項

OAUTH2.ZAUTHENTICATE.mac サンプル (https://github.com/intersystems/Samples-Security から取得) は、前のサブセクションで説明した両方のシナリオをサポートします。このサンプルでは、GetCredentials() サブルーチンは以下のようになります。

```
GetCredentials(ServiceName,Namespace,Username,Password,Credentials) Public {
    If ServiceName="%Service_WebGateway" {
        // Supply user name and password for authentication via a subclass of %OAuth2.Login
        Set Username="OAuth2"
        Set Password=$c(1,2,3)
    }
    Quit $$$OK
}
```

このサブルーチンは、ユーザ名とパスワードが指定されていないと呼び出されます (これはカスタム・ログイン・ページが使用された場合です)。サービス **%Service_WebGateway** では、このサンプルは、ユーザ名とパスワードを (後の処理で呼び出す) ZAUTHENTICATE() サブルーチンでも使用する特定の値に設定します。

ZAUTHENTICATE() サブルーチンには以下が含まれています。

```
If Username="OAuth2",Password=$c(1,2,3) {
    // Authentication is via a subclass of %OAuth2.Login that sets the query parameter CSPOAUTH2
    // with a hash value that allows GetCurrentApplication to determine the application --
    // username/password is supplied by GetCredentials.
    Set sc=##class(OAuth2.Response).GetCurrentApplication(.applicationName)
    Set sessionId=%session.SessionId
} Else {
    // If authentication is based on %session.Login, then application and session id are passed in.
    Set applicationName=Username
    Set sessionId=Password
}
```

後の手順では、次のように isAuthorized() メソッドを呼び出します。

```
Set
isAuthorized=##class(%SYS.OAuth2.AccessToken).IsAuthorized(applicationName,sessionId,,.accessToken,,,.error)
```

isAuthorized() が 1 を返す場合、後述のコードは、イントロスペクション・エンドポイントを呼び出し、そこから取得した情報を使用してユーザを定義します。

```
Set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection(applicationName,accessToken,.jsonObject)
...
Set Username="OAuth2"_jsonObject.sub
Set Properties("FullName")="OAuth account "_Username
Set Properties("Username")=Username
Set Properties("Password")=""    // we don't really care about oauth2 account password
// Set the roles and other Properties as appropriate.
Set Properties("Roles")=roles
```

コードで異なるロジックを使用して、ユーザの定義に必要な情報を取得できます。代わりに以下の方法でこの情報を取得することもできます。

- OpenID Connect を使用している場合には IDToken から取得。この場合、**%SYS.OAuth2.Validate** の ValidateIDToken() を呼び出します。
- OpenID Connect の場合には Userinfo エンドポイントから取得。この場合、**%SYS.OAuth2.AccessToken** の GetUserinfo() を呼び出します。

どの場合でも、ユーザ名が通常の InterSystems IRIS ユーザ名と一致しないユーザを定義する必要があります。

また、アプリケーションの要件に応じて、ルーチンでロールおよび Properties 配列のその他の部分を設定する必要もあります。"代行 (ユーザ定義) 認証コードの作成" と "代行認証" を参照してください。

3.8 バリエーション

このページでは、InterSystems IRIS Web アプリケーションが承認コード付与タイプを使用するシナリオについて主に説明します。このセクションでは、いくつかのバリエーションについて説明します。

- PKCE なしの承認コード付与タイプ
- 暗黙付与タイプ
- パスワード資格情報付与タイプ
- クライアント資格情報付与タイプ
- OnPreHttp 内でのリダイレクトの実行 (承認コードおよび暗黙付与タイプに該当)

- JWT としての要求オブジェクトの受け渡し
- 他のエンドポイントの呼び出し

基本シナリオでは、クライアントは承認サーバからアクセス・トークンを受け取り、必要に応じて承認サーバの追加エンドポイント (イントロスペクション・エンドポイントか Userinfo エンドポイントまたはその両方) を呼び出します。その後、クライアントはリソース・サーバを呼び出します。リソース・サーバがこれらのエンドポイントを個別に呼び出すこともできます。

3.8.1 バリエーション : PKCE の無効化

既定では、承認コード付与タイプを使用するクライアントは、Proof Key for Code Exchange (PKCE) 拡張を活用します。ほとんどの場合、クライアントはこの既定の動作を変更するべきではありません。PKCE は広く受け入れられている重要なセキュリティ機能であるためです。ただし、まれに、PKCE を無効にしたいことがあります。

PKCE を無効にするには、メソッドを呼び出す前に、`GetAuthorizationCodeEndpoint()` の `properties` 引数を変更します。コードには、以下の行を含める必要があります。

```
Set properties("code_verifier")=""
```

`GetAuthorizationCodeEndpoint()` の `properties` 引数の詳細は、"メソッドの詳細" を参照してください。

3.8.2 バリエーション : 暗黙付与タイプ

このバリエーションでは、クライアントはトークンの要求時に暗黙付与タイプを使用します。

構成要件 : "クライアントの構成" の手順を参照してください。ただし、[**クライアント・タイプ**] は使用事例に応じて指定します。

コード要件 : 全体の流れは承認コード付与タイプと似ていますが、GetAuthorizationCodeEndpoint() は呼び出しません。代わりに、**%SYS.OAuth2.Authorization** クラスの GetImplicitEndpoint() メソッドを呼び出します。

```
ClassMethod GetImplicitEndpoint(applicationName As %String,
                                scope As %String,
                                redirectURL As %String,
                                idtokenOnly As %Boolean = 0,
                                responseMode As %String,
                                ByRef properties As %String,
                                Output isAuthorized As %Boolean,
                                Output sc As %Status
                                sessionId as %String="") As %String
```

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- scope は、スペースで区切られた、アクセスが要求されているスコープのリストです。例えば、`"openid profile scope3 scope4"` のような形式をとります。

  既定値は、指定された applicationName のクライアント構成によって決まります。
- redirectURL は、承認サーバがクライアント・サーバにアクセス・トークンを返した後にブラウザをリダイレクトするページの URL です。
- idtokenOnly は ID トークンのみを取得可能にします。この引数が 0 の場合、メソッドはアクセス・トークンと ID トークン (要求に適切なスコープが含まれている場合) の両方を取得します。この引数が 1 の場合、メソッドはアクセス・トークンを取得しません。
- responseMode は、承認サーバの応答モードを指定します。このモードは `"query"` (既定値)、`"fragment"`、または `"form_post"` にできます。

- 参照によって渡される properties は、要求に追加されるパラメータを含む多次元配列です。この配列は以下の構造を持つ必要があります。

配列ノード	配列値
properties (parametername)、ここで parametername はパラメータの名前です。	指定されたパラメータの値。この値は、スカラ値、ダイナミック・オブジェクトのインスタンス、ダイナミック・オブジェクトの UTF-8 のシリアル化されたエンコード形式のいずれかです。 JSON オブジェクトの値を持つパラメータを要求に含める場合は、ダイナミック・オブジェクトを使用します。その 1 つのシナリオが OpenID Connect で定義されている `claims` パラメータです。ダイナミック・オブジェクトの詳細は、"JSON の使用" を参照してください。 `request` または `request_uri` パラメータを使用する場合は、"JWT としての要求オブジェクトの受け渡し" セクションを参照してください。

- このクライアントおよびセッションのアクセス・トークンがローカルに保存され、さらにそのアクセス・トークンが scope 引数で提供されるすべてのスコープを承認する場合、出力として返される isAuthorized の値は 1 になります。それ以外の場合、このパラメータの値は 0 になります。
- 出力として返される sc には、このメソッドが設定したステータス・コードが含まれます。
- sessionId はセッション ID を指定します。既定のセッション (%session.**SessionId**) をオーバーライドする場合のみこれを指定します。

"バリエーション : OnPreHTTP 内でのリダイレクトの実行" も参照してください。

3.8.3 バリエーション : パスワード資格情報付与タイプ

このバリエーションでは、クライアントはトークンの要求時にパスワード資格情報付与タイプを使用します。リソース所有者に属するパスワードがクライアントにある場合は、この付与タイプを使用できます。クライアント・アプリケーションは、ページのリダイレクトは行わずにトークン・エンドポイントへの HTTP POST 操作を実行できます。InterSystems IRIS は、これを行うメソッドを提供します。

構成要件 : "クライアントの構成" の手順を参照してください。ただし、**[クライアントの秘密鍵]** を指定する必要はありません。(一般に、クライアント秘密鍵は、クライアント秘密鍵を必要とし、かつこれを保護できる場合にのみ使用してください。)

コード要件 : アプリケーションは以下を実行します。

1. "トークンの取得" の説明に従って、**%SYS.OAuth2.AccessToken** の IsAuthorized() メソッドを呼び出し、戻り値 (およびエラーの有無) を確認します。
2. IsAuthorized() が 0 を返した場合は、**%SYS.OAuth2.Authorization** の GetAccessTokenPassword() メソッドを呼び出します。

```
ClassMethod GetAccessTokenPassword(applicationName As %String,
                                   username As %String,
                                   password As %String,
                                   scope As %String,
                                   ByRef properties As %String,
                                   Output error As %OAuth2.Error) As %Status
```

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- username はユーザ名です。

- password は対応するパスワードです。
- scope は、スペースで区切られた、アクセスが要求されているスコープのリストです。例えば、"scope1 scope2 scope3" のような形式をとります。

  既定値は、指定された applicationName のクライアント構成によって決まります。
- 参照によって渡される properties は、要求に追加されるパラメータを含む多次元配列です。この配列は以下の構造を持つ必要があります。

配列ノード	配列値
properties (parametername)、ここで parametername はパラメータの名前です。	指定されたパラメータの値。この値は、スカラ値、ダイナミック・オブジェクトのインスタンス、ダイナミック・オブジェクトの UTF-8 のシリアル化されたエンコード形式のいずれかです。 JSON オブジェクトの値を持つパラメータを要求に含める場合は、ダイナミック・オブジェクトを使用します。その 1 つのシナリオが OpenID Connect で定義されている claims パラメータです。ダイナミック・オブジェクトの詳細は、"JSON の使用" を参照してください。

- 出力として返される error は、Null であるか、エラー情報を含む %OAuth2.Error のインスタンスです。

このメソッドはトークン・エンドポイントへの HTTP POST 操作を実行し、アクセス・トークンを受信して保存します (存在する場合)。

3. error 引数を確認し、それに応じて処理を進めます。
4. "トークンの検証" と "HTTP 要求へのアクセス・トークンの追加" の説明に従って作業を進めます。

3.8.4 バリエーション : クライアント資格情報付与タイプ

このバリエーションでは、クライアントはトークンの要求時にクライアント資格情報付与タイプを使用します。この付与タイプを使用すると、クライアント・アプリケーションはユーザから独立してリソース・サーバと通信できます。ユーザ・コンテキストは存在しません。クライアント・アプリケーションは、ページのリダイレクトは行わずにトークン・エンドポイントへの HTTP POST 操作を実行できます。InterSystems IRIS は、これを行うメソッドを提供します。

構成要件 : "クライアントの構成" の手順を参照してください。[**クライアント・タイプ**] に [**プライベート**] を指定し、[**クライアント秘密鍵**] を指定します。

コード要件 : アプリケーションは以下を実行します。

1. "トークンの取得" の説明に従って、%SYS.OAuth2.AccessToken の IsAuthorized() メソッドを呼び出し、戻り値 (およびエラーの有無) を確認します。
2. IsAuthorized() が 0 を返した場合は、%SYS.OAuth2.Authorization の GetAccessTokenClient() メソッドを呼び出します。

```
ClassMethod GetAccessTokenClient(applicationName As %String,
                                 scope As %String,
                                 ByRef properties As %String,
                                 Output error As %OAuth2.Error) As %Status
```

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- scope は、スペースで区切られた、アクセスが要求されているスコープのリストです。例えば、"scope1 scope2 scope3" のような形式をとります。

既定値は、指定された applicationName のクライアント構成によって決まります。

- 参照によって渡される properties は、要求に追加されるパラメータを含む多次元配列です。前のサブセクションで説明した GetAccessTokenPassword() の properties 引数を参照してください。
- 出力として返される error は、Null であるか、エラー情報を含む **OAuth2.Error** のインスタンスです。

このメソッドはトークン・エンドポイントへの HTTP POST 操作を実行し、アクセス・トークンを受信して保存します (存在する場合)。

3. error 引数を確認し、それに応じて処理を進めます。
4. "トークンの検証" と "HTTP 要求へのアクセス・トークンの追加" の説明に従って作業を進めます。

3.8.5 OnPreHTTP 内でのリダイレクトの実行

承認コードおよび暗黙付与タイプの場合、上記の説明では次の手順に従います。

1. **%SYS.OAuth2.AccessToken** の IsAuthorized() メソッドを呼び出します。
2. GetAuthorizationCodeEndpoint() メソッド (承認コード付与タイプ用)、または GetImplicitEndpoint() メソッド (暗黙付与タイプ用) を呼び出します。
3. 上記の手順で返された URL を開くオプション (ボタンなど) を提供することで、ユーザは要求を承認することができます。

その代わりに、(アプリケーション内で) ページ・クラスの OnPreHttp() メソッドを変更することもできます。この場合、GetAccessTokenAuthorizationCode() メソッド (承認コード付与タイプ用) または GetAccessTokenImplicit() メソッド (暗黙付与タイプ用) を呼び出します。このメソッドは、最初にページのコンテンツを表示せずに、承認サーバの認証フォームにブラウザを直接移動します (必要な場合)。

3.8.6 バリエーション: JWT としての要求オブジェクトの受け渡し

OpenID Connect Core 仕様のセクション 6 で規定されているように、InterSystems IRIS は JWT としての要求オブジェクトの受け渡しもサポートしています。値または参照によって要求オブジェクトを渡すことができます。

いずれの場合も、**%SYS.OAuth2.Request** クラスのメソッドを使用します。このセクションに記載されていないその他のメソッドについては、クラス・リファレンスを参照してください。

3.8.6.1 値による要求オブジェクトの受け渡し

`request` パラメータを使用して JWT として要求オブジェクトを渡すには、以下の手順を実行します。

1. **%SYS.OAuth2.Request** クラスの MakeRequestJWT() メソッドを呼び出します。

```
ClassMethod MakeRequestJWT(applicationName As %String,
                           ByRef properties As %String,
                           Output sc As %Status) As %String
```

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- 参照によって渡される properties は、要求に追加されるパラメータを含む多次元配列です。この配列は以下の構造を持つ必要があります。

配列ノード	配列値
properties (parametername)、ここで parametername はパラメータの名前です。	指定されたパラメータの値。この値は、スカラ値、ダイナミック・オブジェクトのインスタンス、ダイナミック・オブジェクトの UTF-8 のシリアル化されたエンコード形式のいずれかです。

- 出力として返される sc には、このメソッドが設定したステータス・コードが含まれます。

このメソッドは JWT の文字列を返します。以下に例を示します。

ObjectScript

```
// create jwt
set jwt=##class(%SYS.OAuth2.Request).MakeRequestJWT("myapp",.properties,.sc)
```

2. GetAuthorizationCodeEndpoint() または GetImplicitEndpoint() の引数として使用する properties 配列を変更します。前の手順で作成した JWT と等しくなるようにノード properties`("request")` を設定します。以下に例を示します。

ObjectScript

```
set properties("request")=jwt
```

3. GetAuthorizationCodeEndpoint() または GetImplicitEndpoint() を呼び出すときに properties 配列を指定します。以下に例を示します。

ObjectScript

```
set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint("myapp",
      scope,redirect,.properties,.isAuthorized,.sc, responseMode)
```

3.8.6.2 参照による要求オブジェクトの受け渡し

`request_uri` パラメータを使用して JWT として要求オブジェクトを渡すには、以下の手順を実行します。

1. %SYS.OAuth2.Request クラスの UpdateRequestObject() メソッドを呼び出します。

```
ClassMethod UpdateRequestObject(applicationName As %String,
                                requestName As %String,
                                ByRef properties As %String,
                                Output sc As %Status) As %SYS.OAuth2.Request
```

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- requestName は要求の名前です。
- 参照によって渡される properties は、要求に追加されるパラメータを含む多次元配列です。この配列は以下の構造を持つ必要があります。

配列ノード	配列値
properties (parametername)、ここで parametername はパラメータの名前です。	指定されたパラメータの値。この値は、スカラ値、ダイナミック・オブジェクトのインスタンス、ダイナミック・オブジェクトの UTF-8 のシリアル化されたエンコード形式のいずれかです。

- 出力として返される sc には、このメソッドが設定したステータス・コードが含まれます。

このメソッドは %SYS.OAuth2.Request のインスタンスの作成、保存、返却を行います。

ObjectScript

```
 // create requestobject
 set
requestobject=##class(%SYS.OAuth2.Request).UpdateRequestObject("myapp","myrequest",.properties,.sc)
```

2. 保存されている要求オブジェクトの URL を取得します。そのためには、このインスタンスの GetURL() メソッドを呼び出します。GetURL() は、最初の引数の出力としてステータス・コードを返すので、コードでこれを確認します。

   ObjectScript

   ```
    Set requesturl=requestobject.GetURL()
   ```

3. GetAuthorizationCodeEndpoint() または GetImplicitEndpoint() の引数として使用する properties 配列を変更します。前の手順で取得した URL と等しくなるようにノード properties`("request_uri")` を設定します。以下に例を示します。

   ```
   set properties("request_uri")=requesturl
   ```

4. GetAuthorizationCodeEndpoint() または GetImplicitEndpoint() を呼び出すときに properties 配列を指定します。以下に例を示します。

   ```
    set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint("myapp",
          scope,redirect,.properties,.isAuthorized,.sc, responseMode)
   ```

3.8.7 バリエーション：承認サーバの他のエンドポイントの呼び出し

%SYS.OAuth2.Authorization のメソッドを使用すると、承認サーバの特定のエンドポイント・セットを呼び出すことができます。承認サーバに他のエンドポイントがある場合は、以下の一般的なプロセスを使用してそのエンドポイントを呼び出します。

1. %Net.HttpRequest のインスタンスを作成し、そのプロパティの設定とメソッドの呼び出しを必要に応じて行い、要求を定義します。

   ```
   Set httpRequest=##class(%Net.HttpRequest).%New()
   Set httpRequest.ContentType="application/x-www-form-urlencoded"
   ...
   ```

   このクラスの詳細は、"インターネット・ユーティリティの使用法" の" HTTP 要求の送信" を参照してください。

2. この要求に認証を追加するには、%SYS.OAuth2.AccessToken の AddAuthentication() メソッドを呼び出します。

   ```
   ClassMethod AddAuthentication(applicationName As %String, httpRequest As %Net.HttpRequest) As
   %Status
   ```

   引数は以下のとおりです。

   - applicationName は OAuth 2.0 クライアントの名前です。
   - httpRequest は %Net.HttpRequest のインスタンスです。

   InterSystems IRIS は指定されたクライアントを検索し、その [認証タイプ]、[SSL 構成]、その他の情報を使用して適切な認証を要求に追加します。

3. 必要に応じてクライアント構成を開くことで、そこに含まれるプロパティを使用できます。そのためには、%SYS ネームスペースに切り替えて、OAuth2.Client の Open() メソッドを呼び出し、クライアント名を引数として渡します。

ObjectScript

```
New $NAMESPACE
set $NAMESPACE="%SYS"
Set client=##class(OAuth2.Client).Open(applicationName,.sc)
If client="" Quit
```

4. HTTP 要求オブジェクトの Post()、Get()、または Put() メソッドを必要に応じて呼び出し、承認サーバのトークン・エンドポイントを引数として提供します。以下に例を示します。

ObjectScript

```
set sc=httpRequest.Post(client.ServerDefinition.TokenEndpoint)
```

5. 必要に応じて追加処理を実行します。

3.9 アクセス・トークンの取り消し

承認サーバがトークンの取り消しをサポートする場合、管理ポータル経由またはプログラムによってアクセス・トークンを取り消すことができます。

3.9.1 ユーザのアクセス・トークンの取り消し

指定されたユーザのアクセス・トークンをすべて取り消すには、次の手順を実行します。

1. **[システム管理]→[セキュリティ]→[OAuth 2.0]→[管理]** を選択します。
2. ユーザ ID を **[ユーザのトークンの取り消し]** フィールドに入力します。
3. **[取り消す]** を選択します。

このタスクを実行するには、%Admin_Secure リソースの USE 権限を持つユーザとしてログインする必要があります。

3.9.2 プログラムによるアクセス・トークンの取り消し

クライアントがアクセス・トークンを取り消す必要がある場合は、**%SYS.OAuth2.AccessToken** の RevokeToken() メソッドを使用します。指定されたトークンを保持しているセッションが削除されると、システムはこのメソッドを自動的に呼び出します (取り消しエンドポイントが指定されている場合)。

```
ClassMethod RevokeToken(applicationName As %String, accessToken As %String) As %Status
```

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- accessToken はアクセス・トークンです。

要求は、`client_id` と `client_secret` が applicationName に関連付けられている、基本的な承認 HTTP ヘッダによって承認されます。

以下に例を示します。

```
set sc=##class(%SYS.OAuth2.AccessToken).RevokeToken("myclient",accessToken)
if $$$ISERR(sc) {
    //error handling here
}
```

サーバで取り消しエンドポイントが指定されていない場合や、[クライアント秘密鍵] が指定されていない場合は、このメソッドを使用できません。

%SYS.OAuth2.AccessToken はメソッド RemoveAccessToken() も提供します。このメソッドはクライアントからアクセス・トークンを削除しますが、サーバからはアクセス・トークンを削除しません。

3.10 JWT で使用するキーの回転

ほとんどの場合、新しい公開/秘密鍵ペアをクライアントに生成させることができます。これは、非対称の RS256、RS384、および RS512 の各アルゴリズムに使用する RSA 鍵にのみ適用されます。(例外は、[X509 証明書] として、[ダイナミック登録以外のソース] を指定する場合です。この場合、新しいキーは生成できません。)

新しい公開/秘密鍵ペアの生成は、キーの回転と呼ばれています。このプロセスは、新しい秘密 RSA 鍵および関連する公開 RSA 鍵を秘密と公開の JWKS に追加します。

クライアントでキーの回転を実行すると、クライアントは新しい秘密 RSA 鍵を使用して、承認サーバに送信する JWT に署名します。同様に、クライアントは新しい公開 RSA 鍵を使用して、承認サーバに送信する JWT を暗号化します。クライアントは、承認サーバから受信した JWT を解読するために新しい RSA 鍵を使用しますが、これが失敗したら古い RSA 鍵を使用するため、古い公開 RSA 鍵を使用して作成された JWT を解読できます。最後に、承認サーバから受信した署名済みの JWT をクライアントが検証できない場合、クライアントに承認サーバの公開の JWKS の URL があると、クライアントは新しい公開の JWKS を取得し、署名の検証を再試行します。(クライアントが動的に登録された場合や、構成で [URL から JWKS] オプションが指定された場合、クライアントは承認サーバの公開の JWKS の URL を持っています。それ以外の場合にはクライアントはこの URL を持っていません。)

指定されたクライアント構成に対してキーを回転するには、次の手順を実行します。

1. 管理ポータルで [システム管理]→[セキュリティ]→[OAuth 2.0]→[クライアント構成] を選択します。
2. クライアント構成が関連付けられたサーバ記述を選択します。

   これによってシステムは、サーバ記述に関連付けられたクライアント構成をすべて表示します。
3. キーを回転させたいクライアントの構成を選択します。
4. [キーの回転] ボタンを選択します。

注釈　対称の HS256、HS384、および HS512 の各アルゴリズムは、常に対称鍵としてクライアントの秘密鍵を使用します。

3.10.1 クライアント用のキーの回転の API

クライアントでキーをプログラムによって回転させるには、OAuth2.Client の RotateKeys() メソッドを呼び出します。

新しい承認サーバの公開の JWKS を取得するには、OAuth2.ServerDefinition の UpdateJWKS() メソッドを呼び出します。

これらのメソッドの詳細は、クラス・リファレンスを参照してください。

3.11 承認サーバからの新しい公開の JWKS の取得

ほとんどの場合、承認サーバは JWKS の公開/秘密鍵ペアを生成します。公開の JWKS をクライアントが受信する方法はさまざまです。1 つの方法は、承認サーバが URL で公開の JWKS を提供する方法です。"サーバ記述の手動による作成 (Discovery なし)" の [URL から JWKS] オプションを参照してください。

承認サーバが [URL から JWKS] で定義されていて、承認サーバが JWKS の新しいペアを生成する場合、同じ URL から新しい公開の JWKS をクライアントに取得させることができます。そのためには、以下の操作を実行します。

1. 管理ポータルで [システム管理]→[セキュリティ]→[OAuth 2.0]→[クライアント構成] を選択します。
2. クライアント構成が関連付けられたサーバ記述を選択します。

   これによってシステムは、サーバ記述に関連付けられたクライアント構成をすべて表示します。
3. クライアントの構成を選択します。
4. [JWKS の更新] ボタンを選択します。

承認サーバが [URL から JWKS] で定義されておらず、承認サーバが JWKS の新しいペアを生成する場合、公開の JWKS を取得し、これをクライアントに送信し、ファイルからロードする必要があります。

4

OAuth 2.0 リソース・サーバとしての InterSystems IRIS Web アプリケーションの使用法

このページでは、OAuth 2.0 フレームワークを使用するリソース・サーバとして InterSystems IRIS® の Web アプリケーションを使用する方法について説明します。

このページでは、リソース・サーバが承認サーバのイントロスペクション・エンドポイントを使用するシナリオについて主に説明します。バリエーションの詳細は、前のセクションを参照してください。

JWT の署名、暗号化、シグニチャの検証、および解読に使用するキーの回転のプロセスについては別途説明します。

4.1 InterSystems IRIS リソース・サーバの前提条件

このページに記載されているタスクを開始する前に、以下のものが利用可能であることを確認します。

- OAuth 2.0 承認サーバ。
- リソース・サーバが承認サーバのエンドポイントを使用する場合は、そのリソース・サーバを OAuth 2.0 承認サーバのクライアントとして登録することができます。詳細は、承認サーバの実装環境によって異なります。

  この場合は、後でこのサーバの具体的な詳細情報も必要になります。

  - 承認サーバの場所 (発行者エンドポイント)
  - トークン・エンドポイントの場所
  - Userinfo エンドポイントの場所 (サポートされている場合。OpenID Connect Core を参照)
  - トークン・イントロスペクション・エンドポイントの場所 (サポートされている場合。RFC 7662 を参照)
  - トークン取り消しエンドポイントの場所 (サポートされている場合。RFC 7009 を参照)
  - 承認サーバによるダイナミック登録のサポートの有無
- 承認サーバがダイナミック登録をサポートしない場合、リソース・サーバのクライアント ID およびクライアント秘密鍵が必要です。承認サーバがこの 2 つの情報を生成します (1 回限り)。ユーザは、この情報をリソース・サーバ・マシンへ安全に伝える必要があります。

4.2 構成要件

"構成要件" を参照してください。クライアント構成の作成手順を以下のように変更します。

- [**アプリケーション名**] にはリソース・サーバのアプリケーション名を指定します。
- [**クライアント・タイプ**] には [**リソース・サーバ**] を指定します。

  [**リソース・サーバ**] をタイプとして指定した場合、構成ページにはリソース・サーバに適用されるオプションのみが表示されることに注意してください。
- [ClientID] にはリソース・サーバのクライアント ID を使用します。
- [clientSecret] にはリソース・サーバのクライアント秘密鍵を使用します。

4.3 コード要件

OAuth 2.0 リソース・サーバは要求を受け取り、そこに含まれるアクセス・トークンを検証し、(そのアクセス・トークンに応じて) 要求された情報を返します。

InterSystems IRIS リソース・サーバを作成するには、リソース・サーバの Web アプリケーションが使用するネームスペースの **%CSP.REST** のサブクラスを作成します。このクラスでは、URL マップおよび対応するメソッドを作成します。これらメソッドでは、以下を実行します。

1. **%SYS.OAuth2.AccessToken** の GetAccessTokenFromRequest() メソッドを呼び出します。このメソッドは、以下のとおりです。

   ```
   ClassMethod GetAccessTokenFromRequest(Output sc As %Status) As %String
   ```

   このページが受信した HTTP 要求にアクセス・トークンがある場合、メソッドはそれを返します。RFC 6750 にある 3 つのフォーマットの 1 つが使用されます。出力として返されるパラメータ sc は、エラー検出の有無を示すステータス・コードです。要求で SSL/TLS を使用しなかった場合はエラーが発生します。さらに、要求に有効なベアラー・ヘッダが含まれていなかった場合にもエラーが発生します。
2. ステータス・コードがエラーであるかどうかを確認します。

   ステータスがエラーの場合、メソッドは適切なエラーを返します (要求された情報は返しません)。
3. ステータス・コードがエラーでない場合は、アクセス・トークンを検証します。そのためには、ValidateJWT() を使用するか、独自のカスタム・メソッドを使用します。"メソッドの詳細" を参照してください。
4. 必要に応じて、追加情報を得るために GetIntrospection() メソッドを呼び出します。このメソッドは承認サーバのイントロスペクション・エンドポイントを呼び出し、アクセス・トークンに関するクレームを取得します。このメソッドは、以下のとおりです。

   ```
   ClassMethod GetIntrospection(applicationName As %String,
                                accessToken As %String,
                                Output jsonObject As %RegisteredObject) As %Status
   ```

   引数は以下のとおりです。

   - applicationName はクライアント・アプリケーションの名前です。
   - accessToken は以前に返されたアクセス・トークンです。
   - 出力として返される jsonObject は、承認サーバがこのアクセス・トークンに関して作成したクレームを含む JSON オブジェクトです。

5. 情報に対するユーザの要求を認可すべきであることを前述の手順が示している場合は、要求された処理を実行し、要求された情報を返します。

以下に例を示します。

```
// This is a dummy resource server which just gets the access token from the request
// and uses the introspection endpoint to ensure that the access token is valid.
// Normally the response would not be security related, but would ocntain some interesting
// data based on the request parameters.
set accessToken=##class(%SYS.OAuth2.AccessToken).GetAccessTokenFromRequest(.sc)
if $$$ISOK(sc) {
   set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection("demo resource",accessToken,.jsonObject)

    if $$$ISOK(sc) {
        write "OAuth 2.0 access token used to authorize resource server (RFC 6749)<br>"
        write "Access token validated using introspection endpoint (RFC 7662)<br>"
        write "   scope='"_jsonObject.scope_"'<br>"
        write "   user='"_jsonObject.username_"'",!
    } else {
        write "Introspection Error="_..EscapeHTML($system.Status.GetErrorText(sc)),!
    }
} else {
    write "Error Getting Access Token="_$system.Status.GetErrorText(sc),!
}

Quit $$$OK
```

4.4 トークンの検証

アクセス・トークンを受け取ったリソース・サーバは、追加のチェックを実行して、要求されたリソースの使用に必要な権限をユーザが保有しているかどうかを確認します。この検証を実行するために、クライアントは、ここで説明するメソッドを使用して追加情報を取得することができます。

ValidateJWT()

場所:このメソッドはクラス %SYS.OAuth2.Validation にあります。

```
ClassMethod ValidateJWT(applicationName As %String,
                        accessToken As %String,
                        scope As %String,
                        aud As %String,
                        Output jsonObject As %RegisteredObject,
                        Output securityParameters As %String,
                        Output sc As %Status) As %Boolean
```

アクセス・トークンが (opaque トークンではなく) JWT である場合にのみこのメソッドを使用します。

このメソッドは JWT の復号化 (必要な場合) と検証を行い、オブジェクト (jsonObject) を作成して JWT プロパティを格納します。JWT を検証するために、このメソッドは、対象者 (aud が指定されている場合)、発行者エンドポイント (サーバ記述で指定されているものとの一致が必要)、スコープ (scope が指定されている場合) を確認します。このメソッドは、アクセス・トークンの有効期限が切れていないことも確認します。署名された JWT と署名されていない JWT の両方が許可されます。JWT が署名されている場合は、そのシグニチャを確認します。

このメソッドは JWT が有効な場合は 1 を返し、それ以外の場合は 0 を返します。さらに、複数の引数を出力として返します。

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- accessToken は検証される JWT です。
- scope は、スペースで区切られたスコープのリストです。例えば、"scope1 scope2 scope3" のような形式をとります。

scope が指定されている場合は、このスコープを含むスコープ・クレームが JWT に含まれている必要があります。

- aud は、このトークンを使用する対象者を指定します。トークンに関連付けられた aud プロパティがある場合 (通常、対象者はトークンの要求時に指定されているため)、aud はトークンの対象者と一致します。aud が指定されていない場合、対象者のチェックは行われません。
- 出力として返される jsonObject は、JWT のクレームを含むダイナミック・オブジェクトです。このダイナミック・オブジェクトには aud、exp、iss などのプロパティが含まれます。ダイナミック・オブジェクトの詳細は、"JSON の使用" を参照してください。
- 出力として返される securityParameters は、ヘッダから取得されたセキュリティ情報を含む多次元配列です。このセキュリティ情報は、シグニチャや復号化の検証で必要に応じて追加使用されます。

  この配列には以下のノードがあります。

ノード	値
securityParameters("sigalg")	シグニチャまたは MAC アルゴリズム。JWT が署名されている場合にのみ設定します。
securityParameters("keyalg")	キー管理アルゴリズム
securityParameters("encalg")	コンテンツ暗号化アルゴリズム

  keyalg と encalg のノードは、両方が指定されるか、両方が Null になります。

- 出力として返される sc には、このメソッドが設定したステータス・コードが含まれます。

このメソッドが成功 (1) を返す場合は、jsonObject を検証し、必要に応じて含まれているクレームを使用して、要求されたリソースへのアクセスを許可するかどうかを決定します。 必要に応じて securityParameters を使用します。

Oauth 仕様では、アプリケーションは署名された JWT と署名されていない JWT の両方を受け入れることができるため、ValidateJWT メソッドは署名されていない JWT を拒否しません。ただし、多くの場合、署名されていない JWT を拒否して、アプリケーションにより厳しいセキュリティを実装することを強くお勧めします。メソッドから返された securityParameters 配列を調べることで、ValidateJWT に渡されたトークンが署名されていないかどうかを確認できます。securityParameters("sigalg") が設定されていない場合、トークンは署名されていません。例えば、以下のコードは、トークンが署名されていないかどうかを判断し、署名されていない場合は拒否します。

```
Set tInitialValidationPassed = ##class(%SYS.OAuth2.Validation).ValidateJWT(tClientName,
tAccessToken, "", "", .tJsonObj,.tSecurityParams, .tValidateStatus)
// the “sigalg” subscript is set only if the JWT was signed
Set tIsTokenSigned = $Data(tSecurityParams("sigalg"))#2
If 'tIsTokenSigned {
   $$$ThrowStatus($System.Status.Error($$$AccessDenied))
}
```

GetIntrospection()

場所:このメソッドはクラス %SYS.OAuth2.AccessToken にあります。

```
ClassMethod GetIntrospection(applicationName As %String,
                             accessToken As %String,
                             Output jsonObject As %RegisteredObject) As %Status
```

このメソッドはアクセス・トークンをイントロスペクション・エンドポイントへ送信し、クレームを含む応答を受信し、このエンドポイントが返したクレームを含むオブジェクト (jsonObject) を作成します。

要求は、client_id と client_secret が applicationName に関連付けられている、基本的な承認 HTTP ヘッダによって承認されます。

引数は以下のとおりです。

- applicationName はクライアント・アプリケーションの名前です。
- accessToken はアクセス・トークンです。
- 出力として返される jsonObject は、イントロスペクション・エンドポイントが返したクレームを含むダイナミック・オブジェクトです。ダイナミック・オブジェクトの詳細は、"JSON の使用" を参照してください。

サーバでイントロスペクション・エンドポイントが指定されていない場合や、[**クライアント秘密鍵**] が指定されていない場合は、このメソッドを使用できません。

このメソッドが成功 (1) を返す場合は、jsonObject を検証し、必要に応じて含まれているクレームを使用して、要求されたリソースへのアクセスを許可するかどうかを決定します。

4.5 バリエーション

このページでは、InterSystems IRIS リソース・サーバが承認サーバのイントロスペクション・エンドポイントを使用するシナリオについて主に説明します。このセクションでは、使用可能ないくつかのバリエーションについて説明します。

4.5.1 バリエーション : リソース・サーバによる Userinfo エンドポイントの呼び出し

リソース・サーバは Userinfo エンドポイントを呼び出すこともできます。そのためには、OAuth クライアントの場合と同様に、リソース・サーバのコードで GetUserinfo() メソッドを使用する必要があります。

4.5.2 バリエーション : リソース・サーバでエンドポイントを呼び出さない

InterSystems IRIS リソース・サーバが承認サーバのエンドポイントを使用しない場合は、このマシンで OAuth 2.0 構成を作成する必要はありません。

さらに、リソース・サーバが GetAccessTokenFromRequest() を使用する必要はありません。代わりに、リソース・サーバは HTTP 承認ヘッダからアクセス・トークンを直接取得し、必要に応じて使用することができます。

5

OAuth 2.0 承認サーバとしての InterSystems IRIS の使用法

このページでは、OAuth 2.0 承認サーバとして InterSystems IRIS® インスタンスを使用する方法について説明します。

クライアント定義を作成するユーザとサーバを設定するユーザは異なる可能性が高いためです。また、クライアント定義の継続的な作成が必要になる場合があるためです。そのため、クライアント定義の作成タスクは、独立したセクションとしてこの記事の最後に収録しています。

5.1 InterSystems IRIS 承認サーバの構成要件

OAuth 2.0 承認サーバとして InterSystems IRIS インスタンスを使用するには、次の構成タスクを実行します。

- InterSystems IRIS にサービスを提供している Web サーバでは、その Web サーバが SSL を使用するように構成します。SSL を使用するように Web サーバを構成する方法は、このドキュメントの対象外であるため、ここでは取り上げません。
- サーバが使用する InterSystems IRIS の SSL 構成を作成します。

  これはクライアント SSL 構成です。証明書は不要です。この構成は Web サーバへの接続に使用されます。この接続を通じて、承認サーバは、`request_uri` パラメータが指定する要求オブジェクトにアクセスします。この接続を通じて、承認サーバは、クライアントの JWKS を更新するときに `jwks_uri` にもアクセスします。クライアントが `request_uri` パラメータを使用して要求を送信せず、承認サーバが `jwks_uri` パラメータでクライアントの JWKS を更新しない場合、承認サーバは SSL 構成を必要としません。

  SSL 構成の作成の詳細は、インターシステムズの "TLS ガイド" を参照してください。

  それぞれの SSL 構成には一意の名前が付いています。参照用に、このドキュメントではこの SSL 構成を `sslconfig` と呼びますが、一意の名前を任意に付けることができます。
- 以下のサブセクションの説明に従ってサーバ構成を作成します。
- その後、必要に応じてクライアント定義を作成します。前のセクションを参照してください。

5.1.1 承認サーバの構成

このタスクを実行するには、%Admin_Secure リソースの USE 権限を持つユーザとしてログインする必要があります。

1. 管理ポータルで **[システム管理]→[セキュリティ]→[OAuth 2.0]→[サーバ構成]** を選択します。

2. [一般] タブで、以下の詳細を指定します。

   - [説明] – 必要に応じて説明を入力します。
   - [発行者エンドポイント] – 承認サーバのエンドポイント URL を指定します。この URL を指定するには、次のオプションの値を入力します。
     - [ホスト名] – 承認サーバのホスト名または IP アドレスを指定します。
     - [ポート] – Web ゲートウェイ構成の変更に対応するために必要な場合は、この値を指定します。
     - [接頭語] – Web ゲートウェイ構成の変更に対応するために必要な場合は、この値を指定します。

     指定した発行者エンドポイントは以下の形式になります。

     ```
     https://hostname:port/prefix/oauth2
     ```

     [ポート] の指定を省略すると、コロンが省略されます。同様に、[接頭語] の指定を省略すると、`hostname:port` と `oauth2` の間のスラッシュが 1 つだけになります。
   - [対象者は必須] – 承認サーバが承認コードおよび暗黙の要求で `aud` パラメータを必要とするかどうかを指定します。このチェック・ボックスのチェックを外すと、`aud` は必須でなくなります。
   - [ユーザ・セッションのサポート] – 承認サーバがユーザ・セッションをサポートするかどうかを指定します。InterSystems IRIS 承認サーバは、(Web セッションではない) ユーザ・セッションをサポートできます。その場合、InterSystems IRIS はセッション維持クラス ([セッション維持クラス]) を使用します (既定値が提供されます)。この章で後述する "コードのカスタマイズ・オプションと全体的な手順" を参照してください。このチェック・ボックスのチェックを外すと、セッションはサポートされません。
   - [パブリッククライアント更新を許可] – サーバでクライアントのリフレッシュ・トークンを処理できるようにするかどうかを指定します。このチェック・ボックスにチェックを付けると、サーバはリフレッシュ・トークンを処理するためにクライアントの秘密鍵を必要としなくなります。
   - [Enforce Proof Key for Code Exchange (PKCE) for public clients] – 承認サーバで承認コード付与タイプがサポートされる場合、このチェック・ボックスにチェックを付けると、パブリック・クライアントは承認コードを要求してコードをアクセス・トークンに交換する際に、PCKE シークレット値を提供する必要があります。
   - [Enforce Proof Key for Code Exchange (PKCE) for confidential clients] – 承認サーバで承認コード付与タイプがサポートされる場合、このチェック・ボックスにチェックを付けると、機密クライアントは承認コードを要求してコードをアクセス・トークンに交換する際に、PCKE シークレット値を提供する必要があります。
   - [HTTPベースのフロントチャネルログアウトをサポート] – フロント・チャネル・ログアウトを有効にするか無効にするかを指定します。既定では、フロント・チャネル・ログアウトが有効になっています。このチェック・ボックスのチェックを外すと、ユーザがサーバからログアウトしたときに、そのユーザはサーバに登録されているクライアントからもログアウトされます。

     注釈　InterSystems IRIS 承認サーバでフロント・チャネル・ログアウトがサポートされるようにするには、`/oauth2` Web アプリケーションの [ユーザ Cookie スコープ] を `[Lax]` に設定する必要があります。アプリケーションの設定の構成の詳細は、"アプリケーションの作成および編集" を参照してください。
   - [フロントチャネルログアウトURLとともに sid (セッションID) クレームの送信をサポート] – フロント・チャネル・ログアウト URL と共にセッション ID クレームを送信できるようにするかどうかを指定します。既定では、セッション ID が送信されるようになっています。このチェック・ボックスのチェックを外すと、サーバがフロント・チャネル・ログアウト URL を呼び出した場合、その URL には `iss` (発行者) と `sid` (セッション ID) の各クエリ・パラメータが付加されません。
   - [リフレッシュ・トークンの返却] – アクセス・トークンと共にリフレッシュ・トークンが返される条件を指定します。それぞれのビジネス・ケースに適したオプションを選択します。

- [サポートされている付与タイプ] ― この承認サーバがアクセス・トークンを作成するために使用を許可する付与タイプを指定します。少なくとも 1 つ選択します。
- [OpenID プロバイダ・ドキュメント] ― OpenID プロバイダによって提供される URL を以下のように指定します。
  - [サービス・ドキュメント URL] ― OpenID プロバイダの使用時に開発者が理解しておく必要があると思われる、人が読める情報を提供する Web ページの URL です。
  - [ポリシー URL] ― Relying Party がプロバイダによって提供されるデータを使用する方法について、OpenID プロバイダのポリシーを記述する Web ページの URL です。
  - [サービス条件 URL] ― OpenID プロバイダのサービス条件を記述する Web ページの URL です。
- [SSL/TLS 構成] ― 承認サーバ用に作成した SSL/TLS 構成を選択します (例:`sslconfig`)。

3. [スコープ] タブで、以下の詳細を指定します。
   - [スコープ] と [説明] の列を持つテーブル ― この承認サーバがサポートしているスコープをすべて指定します。
   - [サポートされていないスコープを許可] ― サポートしていないスコープ値を承認サーバが無視するかどうかを指定します。このチェック・ボックスのチェックを外すと、サポートされていないスコープ値が要求に含まれる場合、承認サーバはエラーを返します。この場合、要求は処理されません。このチェック・ボックスにチェックを付けると、承認サーバはスコープを無視して要求を処理します。
   - [既定のスコープ] ― アクセス・トークンの要求やクライアント構成でスコープが指定されていない場合に使用する、アクセス・トークンの既定のスコープを指定します。
4. [間隔] タブで、以下の詳細を指定します。
   - [アクセス・トークン間隔] ― このサーバが発行したアクセス・トークンの有効期限が切れる秒数を指定します。既定値は 3,600 秒です。
   - [承認コード間隔] ― このサーバが発行した承認コードの有効期限が切れる秒数を指定します。既定値は 60 秒です。
   - [リフレッシュ・トークン間隔] ― このサーバが発行したリフレッシュ・トークンの有効期限が切れる秒数を指定します。既定値は 24 時間 (86,400 秒) です。
   - [セッション終了間隔] ― ユーザ・セッションが自動的に終了する秒数を指定します。値 0 はセッションが自動的に終了しないことを意味します。既定値は 24 時間 (86,400 秒) です。
   - [クライアント秘密鍵の有効期間] ― このサーバが発行したクライアント秘密鍵の有効期限が切れる秒数を指定します。既定値 (0) は、クライアント秘密鍵が期限切れにならないことを示します。
5. [JWT 設定] タブで、以下の詳細を指定します。
   - [X509 資格情報から JWT 設定を作成] ― 証明書に関連付けた秘密鍵を署名と暗号化で使用する場合は、このオプションを選択します。この場合は、"証明書と JWT (JSON Web Token)" の "OAuth 2.0 承認サーバの証明書の使用" も参照してください。

     注釈　インターシステムズでは、[X509 資格情報から JWT 設定を作成] オプションが使用されることはまれで、次に説明する既定の動作が代わりに使用されると考えています。

     このオプションのチェックを外したままにすると、システムは JWKS (JSON Web Key Set) のペアを生成します。一方の JWKS は秘密鍵で、(アルゴリズムあたり) 必要なすべての秘密鍵だけでなく、対称鍵として使用するクライアント秘密鍵も含んでいます。この JWKS は共有されません。他方の JWKS には、対応する公開鍵が含まれていて、公開されて利用できます。また、InterSystems IRIS は、公開の JWKS をクライアントにコピーして、クライアントが承認サーバから JWT の暗号化およびシグニチャの検証を実行できるようにします。

- **[署名アルゴリズム]** – JWT を署名するときに使用するアルゴリズムを選択します。JWT を署名しない場合は空白のままにします。
- **[キー管理アルゴリズム]** – JWT を暗号化するときにキー管理で使用するアルゴリズムを選択します。

  この選択は、コンテンツの暗号化アルゴリズムを選択する場合にのみ行います。
- **[コンテンツ暗号化アルゴリズム]** – JWT を暗号化するときに使用するアルゴリズムを選択します。JWT を暗号化しない場合は空白のままにします。アルゴリズムを選択する場合は、キー管理のアルゴリズムも選択する必要があります。

6. **[カスタマイズ]** タブで、"コードのカスタマイズ・オプション" に従って詳細を指定します。
7. **[保存]** を選択します。

この構成を保存すると、システムは承認サーバが使用する Web アプリケーション (`/oauth2`) を作成します。この Web アプリケーションは変更しないでください。

5.2 コードのカスタマイズ・オプションと全体的な手順

このセクションでは、承認サーバの構成オプションの**[カスタマイズ・オプション]** セクションにある項目について説明します。サブセクションでは、全体的な手順と既定のクラスについて説明します。

- **[認証クラス]** – **%OAuth2.Server.Authenticate** (既定値) またはそのクラスのカスタム・サブクラスを使用します。

  カスタム・サブクラスを定義する場合は、必要に応じて以下のメソッドの一部またはすべてを実装します。
  - BeforeAuthenticate() – 必要に応じてこのメソッドを実装し、認証の前にカスタム処理を実行します。
  - DisplayLogin() – 必要に応じてこのメソッドを実装し、ユーザを識別するログイン・ページを表示します。(一般的でない代替メソッドについては、"DirectLogin() の実装" を参照してください。)
  - DisplayPermissions() – 必要に応じてこのメソッドを実装し、要求された許可をユーザに表示します。
  - AfterAuthenticate() – 必要に応じてこのメソッドを実装し、認証の後にカスタム処理を実行します。
- **[ユーザ検証クラス]** – **%OAuth2.Server.Validate** (既定値) を使用するか、次のメソッドを定義するカスタム・クラスを使用します。
  - ValidateUser() (クライアント資格情報を除くすべての付与タイプで使用)
  - ValidateClient() (クライアント資格情報付与タイプで使用)

  カスタム・クラスを定義して使用することを強くお勧めします。**%OAuth2.Server.Validate** クラスは実例で説明するためのものであり、実稼働環境での使用に適していない可能性があります。
- **[セッション維持クラス]** – 既定のセッション維持クラス **OAuth2.Server.Session** は、HTTP 専用 Cookie を介してユーザ・セッションを維持します。この既定のクラスを拡張してカスタム・ロジックを実装することはできません。代わりに、`%OAuth2.Server.CookieSession` または `%OAuth2.Server.AbstractSession` を要件に応じて拡張できます。既定のクラスのロジックの大部分は `%OAuth2.Server.CookieSession` が元になっているため、既存のコードを活用するカスタムのセッション維持クラスを実装する場合は (opaque ブラウザ Cookie の使用を含む)、%OAuth2.Server.CookieSession を拡張してカスタム・クラスにします。または、インターシステムズの既定のロジックに依存しない、完全にカスタムのセッション維持クラスを実装することもできます。このカスタマイズ方法を使用するには、%OAuth2.Server.AbstractSession を拡張して抽象メソッドを実装します。

- **[トークン生成クラス]** – **%OAuth2.Server.Generate** (既定値)、**%OAuth2.Server.JWT**、またはメソッド GenerateAccessToken() を定義するカスタム・クラスを使用します。カスタム・クラスを作成する場合は、ここに記載したクラスの 1 つをサブクラス化すると、使用するメソッドが得られるため便利です。
- **[カスタマイズ・ネームスペース]** – カスタマイズ・コードを実行するネームスペースを指定します。
- **[カスタマイズ・ロール]** – カスタマイズ・コードを実行するときに使用するロールを 1 つ以上指定します。

カスタム・サブクラスを使用する場合は、"カスタム・メソッドの実装" を参照してください。

5.2.1 InterSystems IRIS 承認サーバが要求を処理する方法

このセクションでは、トークンの承認コード要求または暗黙要求を受け取った InterSystems IRIS 承認サーバが実行する処理について説明します。

1. **[認証クラス]** オプションで指定されたクラスの BeforeAuthenticate() メソッドを呼び出します。このメソッドの目的は、ユーザの識別を開始する前に要求に変更を加えることです。

   既定のクラスでは、このメソッドはスタブです。

2. 次に、付与タイプが承認コードまたは暗黙付与である場合、InterSystems IRIS は以下の手順を実行します。

   a. **[認証クラス]** オプションで指定されたクラスの DisplayLogin() を呼び出します。("DirectLogin() の実装" も参照してください。)

      既定のクラスで、DisplayLogin() は単純な HTML ログイン・ページを表示します。

   b. ユーザ名が Null でない場合は、**[ユーザ検証クラス]** オプションで指定されたクラスの ValidateUser() メソッドを呼び出します。このメソッドの目的は、ユーザを検証して、(properties 配列の変更によって) トークン・エンドポイント、Userinfo エンドポイント、トークン・イントロスペクション・エンドポイントによって返されるクレームを準備することです。

      既定のクラスでは、このメソッドは例にすぎず、実稼働環境での使用には適していない可能性があります。

   c. ユーザが検証されると、**[認証クラス]** オプションで指定されたクラスの DisplayPermissions() メソッドを呼び出します。このメソッドの目的は、要求された許可の一覧を示すページをユーザに表示することです。

      既定のクラスで、このメソッドは許可の一覧を示す単純な HTML ページを表示します。

   または、付与タイプがパスワード資格情報である場合、InterSystems IRIS は **[ユーザ検証クラス]** オプションで指定されたクラスの ValidateUser() メソッドを呼び出します。

   または、付与タイプがクライアント資格情報である場合、InterSystems IRIS は **[ユーザ検証クラス]** オプションで指定されたクラスの ValidateClient() メソッドを呼び出します。

3. ユーザが許可を受け入れると、**[認証クラス]** オプションで指定されたクラスの AfterAuthenticate() メソッドを呼び出します。このメソッドの目的は、アクセス・トークンを生成する前にカスタム処理を実行することです。

   既定のクラスでは、このメソッドはスタブです。

4. **[トークン生成クラス]** オプションで指定されたクラスの GenerateAccessToken() メソッドを呼び出します。このメソッドの目的は、ユーザに返すアクセス・トークンを生成することです。

   既定のクラス (**%OAuth2.Server.Generate**) で、このメソッドは opaque 文字列のアクセス・トークンを生成します。さらに InterSystems IRIS は代替クラス (**%OAuth2.Server.JWT**) を提供し、このクラスで GenerateAccessToken() は JWT のアクセス・トークンを生成します。

5.2.2 既定のクラス

このセクションでは、InterSystems IRIS 承認サーバの既定のクラスについて説明します。さらに、[**トークン生成クラス**] の別のオプションとして提供される **%OAuth2.Server.JWT** クラスについても説明します。

5.2.2.1 %OAuth2.Server.Authenticate (認証クラスの既定値)

%OAuth2.Server.Authenticate クラスは、以下のメソッドを定義します (呼び出し順に記載)。

- BeforeAuthenticate() はスタブです。OK ステータスを表示して終了します。
- DisplayLogin() は、[**ログイン**] ボタンと [**キャンセル**] ボタンからなる単純なログイン・ページを作成する HTML を記述します。
- DisplayPermissions() は、要求された許可を表示する単純なページを作成する HTML を記述します。このページには、[**許可**] ボタンと [**キャンセル**] が表示されます。
- AfterAuthenticate() はスタブです。OK ステータスを表示して終了します。

5.2.2.2 %OAuth2.Server.Validate (ユーザ検証クラスの既定値)

%OAuth2.Server.Validate クラスは、[**ユーザ検証クラス**] オプションの既定のクラスです。

注釈　このクラスは実例で説明するためのものであり、実稼働環境での使用には適していない可能性があります。各自のニーズに応じてこのクラスを置き換えるか、サブクラス化することをお勧めします。

このクラスは、以下のサンプル・メソッドを定義します。

- ValidateUser() は以下を実行します。
  1. 指定されたユーザを **IRISSYS** データベースで検索します。
  2. そのユーザのパスワードを検証します。
  3. そのユーザに関する情報を含む多次元配列を取得します。
  4. この配列を使用して追加のクレームを properties オブジェクトに加えます。
- SupportedClaims() は、この承認サーバがサポートしているクレームの $LIST を返します。既定では、このメソッドは OpenID Connect Core で定義されているクレームのリストを特に返します。
- ValidateClient() (クライアント資格情報付与タイプで使用) はすべてのクライアントを受け入れて、プロパティは追加しません。

サブクラス内のこれらのメソッドはすべてオーバーライドすることができます。

5.2.2.3 OAuth2.Server.Session (セッション・クラスの既定値)

%OAuth2.Server.Session クラスは、[**セッション維持クラス**] オプションの既定のクラスです。このクラスは、HTTP 専用 Cookie を介してセッションを維持します。

このクラスで、GetUser() メソッドは現在のセッションにアクセスしようとします。セッションが存在する場合、このメソッドはそのセッションからユーザ名を取得して返します。セッションが存在しない場合、このメソッドはユーザ名を空の文字列で返し、出力としてエラー・ステータスも返します。

このクラスの追加情報は、クラス・リファレンスを参照してください。

5.2.2.4 %OAuth2.Server.Generate (トークン生成クラスの既定値)

%OAuth2.Server.Generate クラスは、**[トークン生成クラス]** オプションの既定のクラスです。このクラスは、以下のメソッドを定義します。

- GenerateAccessToken() は、opaque アクセス・トークンとしてランダムな文字列を生成します。
- IsJWT() は 0 を返します。

5.2.2.5 %OAuth2.Server.JWT (アクセス・トークン生成クラスの別のオプション)

%OAuth2.Server.JWT クラスは、**[トークン生成クラス]** オプションで使用できる (またはサブクラス化できる) 別のクラスです。このクラスは、以下のメソッドを定義します。

- GenerateAccessToken() は JWT を返します。承認サーバの構成の **[JSON Web Token (JWT) の設定]** に従って、InterSystems IRIS は JWT を返す前に JWT の署名か暗号化またはその両方を行います。
- IsJWT() は 1 を返します。
- CreateJWT() はクレームを含む JSON オブジェクトに基づいて JWT を作成し、承認サーバの構成で規定されているように JWT の署名と暗号化を行います。このメソッドは、OAuth 2.0 および OpenID Connect 使用法の仕様に準拠しており、サブクラス内でオーバーライドするべきではありません。
- AddClaims() ー 要求されたクレームを JWT に追加します。このメソッドは、以下のとおりです。

```
ClassMethod AddClaims(claims As %ArrayOfObjects,
                      properties As %OAuth2.Server.Properties,
                      json As %DynamicObject)
```

  引数は以下のとおりです。

  - claims は **%OAuth2.Server.Claim** インスタンスの配列です。
  - properties は、承認サーバが使用するプロパティとクレームを含む **%OAuth2.Server.Properties** のインスタンスです。
  - json は JWT を表すダイナミック・オブジェクトです。このオブジェクトはこのメソッドによって変更されます。

5.3 InterSystems IRIS 承認サーバのカスタム・メソッドの実装

承認サーバの動作をカスタマイズするには、"コードのカスタマイズ・オプション" の説明に従ってクラスを定義します。次に、このセクションの情報に基づいて、カスタマイズする処理手順に応じて、これらのクラスのメソッドを定義します。

1. 認証前のオプションのカスタム処理
2. ユーザの識別
3. ユーザの検証とクレームの指定
4. 必要に応じたユーザへの許可の表示
5. 認証後のオプションのカスタム処理
6. アクセス・トークンの生成

これらのサブセクションに続く最後のサブセクションでは、このサーバがクライアント資格情報付与タイプをサポートする必要がある場合にクライアントを検証する方法について説明します。クライアント資格情報付与タイプでは、前述のリストの手順 2 ～ 4 は使用しません。

5.3.1 認証前のオプションのカスタム処理

ここに記載されている情報は、すべての付与タイプに当てはまります。

ユーザを認証する前にカスタム処理を実行するには、認証クラスの BeforeAuthenticate() メソッドを実装します。このメソッドには、以下のシグニチャがあります。

```
ClassMethod BeforeAuthenticate(scope As %ArrayOfDataTypes,
                               properties As %OAuth2.Server.Properties) As %Status
```

以下はその説明です。

- scope は、元のクライアント要求に含まれるスコープを含む %ArrayOfDataTypes のインスタンスです。配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- properties は、承認サーバが使用するプロパティとクレームを含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。

必要に応じて、メソッドのこれらの引数のいずれかまたは両方を変更します。どちらの引数もユーザの識別に使用されるメソッドに後で渡されます。このメソッドは %Status を返す必要があります。

通常、このメソッドを実装する必要はありません。ただし、このメソッドの使用事例の1つとして、FHIR® で使用される launch と launch/patient のスコープを実装するのに利用するというようなものがあります。この事例では、特定の患者を含めるようにスコープを調整する必要があります。

5.3.2 ユーザの識別

ここに記載されている情報は、承認コード付与タイプと暗黙付与タイプにのみ当てはまります。

ユーザを識別するには、認証クラスの DisplayLogin() メソッドを実装します。DisplayLogin() メソッドには、以下のシグニチャがあります。

```
ClassMethod DisplayLogin(authorizationCode As %String,
                         scope As %ArrayOfDataTypes,
                         properties As %OAuth2.Server.Properties,
                         loginCount As %Integer = 1) As %Status
```

以下はその説明です。

- authorizationCode
- scope は、元のクライアント要求に含まれるスコープを含む %ArrayOfDataTypes のインスタンスです。このスコープは、BeforeAuthenticate() メソッドによって変更されている可能性があります。配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- properties は、承認サーバが受信し、処理の早い段階にメソッドが変更したプロパティとクレームを含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。
- loginCount は、ログイン試行の回数を示す整数値です。

このメソッドは、ユーザのログイン・フォームを表示する HTML を記述します。ログイン・フォームには、[ユーザ名] フィールド、[パスワード] フィールド、[承認コード] フィールド (非表示) を含める必要があります。既定の DisplayLogin() メソッドは、%CSP.Page の InsertHiddenField() メソッドを使用して、承認コードの非表示フィールドを追加します。

通常、このフォームには Login ボタンと Cancel のボタンがあります。これらのボタンによってフォームが送信されます。ユーザが Login ボタンでフォームを送信すると、このメソッドはユーザ名とパスワードを受け入れます。ユーザが Cancel ボタンでフォームを送信すると、承認プロセスは access_denied のエラーを返して終了します。

実装時に、同じページに許可を表示することも選択できます。その場合、メソッドはスコープを表示し、Accept という名前のボタンを使用してページを送信します。

このメソッドは %Status を返す必要があります。

5.3.2.1 properties.CustomProperties の更新

フォームに名前が p_ で始まる要素が含まれている場合は、その要素に特別な処理を施します。InterSystems IRIS は DisplayLogin() メソッドが返した要素の値を properties.CustomProperties 配列に追加するとき、最初に p_ 接頭語を名前から削除します。例えば、フォームに p_addme という名前の要素が含まれている場合、InterSystems IRIS は addme (および p_addme 要素の値) を properties.CustomProperties 配列に追加します。

必要に応じて properties の他のプロパティをメソッドで直接設定することもできます。

5.3.3 ユーザの検証とクレームの指定

ここに記載されている情報は、クライアント資格情報付与タイプを除くすべての付与タイプに当てはまります。(この付与タイプについては、"クライアントの検証" を参照してください。)

ユーザを検証し、トークン・エンドポイント、Userinfo エンドポイント、トークン・イントロスペクション・エンドポイントによって返されるクレームを指定するには、ユーザ検証クラスの ValidateUser() メソッドを定義します。このメソッドには、以下のシグニチャがあります。

```
ClassMethod ValidateUser(username As %String,
                         password As %String,
                         scope As %ArrayOfDataTypes,
                         properties As %OAuth2.Server.Properties,
                         Output sc As %Status) As %Boolean
```

以下はその説明です。

- username はユーザが提供したユーザ名です。
- password はユーザが提供したパスワードです。ユーザが既にログインしている場合、InterSystems IRIS は空の文字列の password でこのメソッドを呼び出します。この場合、メソッドは password が空の文字列であることを検知し、パスワードの確認を試みません。
- scope は、元のクライアント要求に含まれるスコープを含む %ArrayOfDataTypes のインスタンスです。このスコープは、BeforeAuthenticate() メソッドによって変更されている可能性があります。配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- properties は、承認サーバが受信し、処理の早い段階にメソッドが変更したプロパティとクレームを含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。
- sc はこのメソッドが設定したステータス・コードです。このコードを使用してエラーの詳細を伝えます。

このメソッドは、以下を実行します。

- パスワードが指定されたユーザ名に対応していることを確認します。
- 業務のニーズに応じて scope および properties 引数を適宜使用します。
- 必要に応じて properties オブジェクトを変更してクレーム値を指定するか、新しいクレームを追加します。以下に例を示します。

```
// Setup claims for profile and email OpenID Connect scopes.
Do properties.SetClaimValue("sub",username)
Do properties.SetClaimValue("preferred_username",username)
Do properties.SetClaimValue("email",email)
Do properties.SetClaimValue("email_verified",0,"boolean")
Do properties.SetClaimValue("name",fullname)
```

- エラーの場合には sc 変数を設定します。
- ユーザを有効と見なす場合は 1 を返し、それ以外の場合は 0 を返します。

ValidateUser() が値を返すと、承認サーバは properties オブジェクトの以下の値を自動的に設定します (これらの値が欠落している場合)。

- properties.ClaimValues:
  - iss — 承認サーバの URL
  - sub — client_id
  - exp — 1840 年 12 月 31 日からの秒単位による有効期限
- properties.CustomProperties:
  - client_id — 要求しているクライアントの client_id

5.3.4 許可の表示

ここに記載されている情報は、承認コード付与タイプと暗黙付与タイプにのみ当てはまります。

ユーザの検証後に許可を表示するには、認証クラスの DisplayPermissions() メソッドを実装します。このメソッドには、以下のシグニチャがあります。

```
ClassMethod DisplayPermissions(authorizationCode As %String,
                               scopeArray As %ArrayOfDataTypes,
                               currentScopeArray As %ArrayOfDataTypes,
                               properties As %OAuth2.Server.Properties) As %Status
```

以下はその説明です。

- authorizationCode は承認コードです。
- scopeArray は、ユーザがまだ許可を与えられていない、新たに要求されたスコープです。この引数は、%ArrayOfDataTypes のインスタンスです。

  配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- currentScopeArray は、ユーザが以前に許可を与えられているスコープです。この引数は、%ArrayOfDataTypes のインスタンスです。

  配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- properties は、承認サーバが受信し、処理の早い段階にメソッドが変更したプロパティとクレームを含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。

このフォームには Accept ボタンと Cancel のボタンが必要です。これらのボタンによってフォームが送信されます。ユーザが Accept ボタンでフォームを送信すると、このメソッドは承認を続行します。ユーザが Cancel ボタンでフォームを送信すると、承認プロセスは終了します。

5.3.5 認証後のオプションのカスタム処理

ここに記載されている情報は、すべての付与タイプに当てはまります。

認証後にカスタム処理を実行するには、認証クラスの AfterAuthenticate() メソッドを実装します。このメソッドには、以下のシグニチャがあります。

```
ClassMethod AfterAuthenticate(scope As %ArrayOfDataTypes, properties As %OAuth2.Server.Properties) As
 %Status
```

以下はその説明です。

- scope は、承認要求が設定したスコープとこのメソッドが呼び出される前のすべての処理を含む %ArrayOfDataTypes のインスタンスです。配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- properties は、承認要求が設定したプロパティおよびクレームとこのメソッドが呼び出される前のすべての処理を含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。

必要に応じて、メソッドのこれらの引数のいずれかまたは両方を変更します。具体的には、プロパティを認証 HTTP 応答に追加することもできます。その場合は、プロパティを properties.ResponseProperties に追加します。

通常、このメソッドを実装する必要はありません。ただし、このメソッドの使用事例の1つとして、FHIR® で使用される launch と launch/patient のスコープを実装するのに利用するというようなものがあります。この事例では、特定の患者を含めるようにスコープを調整する必要があります。

5.3.6 アクセス・トークンの生成

ここに記載されている情報は、すべての付与タイプに当てはまります。

アクセス・トークンを生成するには、トークン生成クラスの GenerateAccessToken() メソッドを実装します。このメソッドには、以下のシグニチャがあります。

```
ClassMethod GenerateAccessToken(properties As %OAuth2.Server.Properties, Output sc As %Status) As
%String
```

以下はその説明です。

- properties は、承認サーバが受信し、処理の早い段階にメソッドが変更したプロパティとクレームを含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。
- 出力として返される sc は、このメソッドが設定したステータス・コードです。この変数を設定してエラーの詳細を伝えます。

このメソッドはアクセス・トークンを返します。このアクセス・トークンは properties 引数に基づきます。このメソッドで、クレームを JSON 応答オブジェクトに追加することもできます。そのためには、properties オブジェクトの ResponseProperties 配列プロパティを設定します。

5.3.7 クライアントの検証

ここに記載されている情報は、クライアント資格情報付与タイプにのみ当てはまります。

クライアント資格情報を検証し、トークン・エンドポイント、Userinfo エンドポイント、トークン・イントロスペクション・エンドポイントによって返されるクレームを指定するには、ユーザ検証クラスの ValidateClient() メソッドを定義します。このメソッドには、以下のシグニチャがあります。

```
ClassMethod ValidateClient(clientId As %String,
                           clientSecret As %String,
                           scope As %ArrayOfDataTypes,
                           Output properties As %OAuth2.Server.Properties,
                           Output sc As %Status) As %Boolean
```

以下はその説明です。

- clientId はクライアント ID です。
- clientSecret はクライアント秘密鍵です。
- scope は、元のクライアント要求に含まれるスコープを含む %ArrayOfDataTypes のインスタンスです。このスコープは、BeforeAuthenticate() メソッドによって変更されている可能性があります。配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- properties は、承認サーバが受信し、処理の早い段階にメソッドが変更したプロパティとクレームを含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。
- sc はこのメソッドが設定したステータス・コードです。このコードを使用してエラーの詳細を伝えます。

このメソッドは、以下を実行します。

- クライアント秘密鍵が指定されたクライアント ID に対応していることを確認します。
- 業務のニーズに応じて scope および properties 引数を適宜使用します。
- 必要に応じて、properties オブジェクトを変更してクレーム値を指定します。以下に例を示します。

```
// Setup claims for profile and email OpenID Connect scopes.
Do properties.SetClaimValue("sub",username)
Do properties.SetClaimValue("preferred_username",username)
Do properties.SetClaimValue("email",email)
Do properties.SetClaimValue("email_verified",0,"boolean")
Do properties.SetClaimValue("name",fullname)
```

- エラーの場合には sc 変数を設定します。
- ユーザを有効と見なす場合は 1 を返し、それ以外の場合は 0 を返します。

ValidateClient() が値を返すと、承認サーバは properties オブジェクトの以下の値を自動的に設定します (これらの値が欠落している場合)。

- properties.ClaimValues:
  - iss — 承認サーバの URL
  - sub — client_id
  - exp — 1840 年 12 月 31 日からの秒単位による有効期限
- properties.CustomProperties:
  - client_id — 要求しているクライアントの client_id

5.4 %OAuth2.Server.Properties オブジェクトの詳細

前のセクションで説明したメソッドでは、properties 引数を使用していますが、この引数は%OAuth2.Server.Properties のインスタンスです。%OAuth2.Server.Properties クラスの目的は、承認サーバ・コード内のメソッド間で受け渡す必要がある情報を保持することです。このセクションでは、このクラスの基本プロパティとクレーム関連のプロパティについて説明します。このクラスには、クレームを操作するメソッドもあります。これらのメソッドについては、最後のサブセクションで説明します。

5.4.1 基本プロパティ

%OAuth2.Server.Properties クラスには以下の基本プロパティがあります。これらのプロパティは、カスタム・コードの内部処理の情報の伝達に使用されます。

RequestProperties

```
Property RequestProperties as array of %String (MAXLEN="");
```

承認要求のクエリ・パラメータが含まれています。

このプロパティは配列であるため、通常の配列インタフェースを使用して操作します。(これは、このクラスの他のプロパティでも同様です。)例えば、クエリ・パラメータの値を取得するには、RequestProperties.GetAt(parmname) を使用します。ここで、parmname はクエリ・パラメータの名前です。

ResponseProperties

```
Property ResponseProperties as array of %String (MAXLEN="");
```

トークン要求への JSON 応答オブジェクトに追加されるプロパティがすべて含まれます。このプロパティは必要に応じて設定します。

CustomProperties

```
Property CustomProperties as array of %String (MAXLEN="");
```

さまざまなカスタマイズ・コード間の通信に使用されるカスタム・プロパティが含まれます。"properties.CustomProperties の更新" を参照してください。

ServerProperties

```
Property ServerProperties as array of %String (MAXLEN="");
```

承認サーバがカスタマイズ・コードと共有するプロパティが含まれます。認証クラスで使用するために、クライアント・プロパティの `logo_uri`、`client_uri`、`policy_uri`、`tos_uri` がこの方法で共有されます。

5.4.2 クレーム関連のプロパティ

%OAuth2.Server.Properties クラスには、IntrospectionClaims、IDTokenClaims、UserinfoClaims、JWTClaims プロパティがあります。これらのプロパティは、必須のクレーム、特にカスタム・クレームに関する情報を伝達します。

このクラスには、実際のクレーム値を伝達する ClaimValues プロパティもあります。カスタマイズ・コードでクレームの値を設定する必要があります (通常は ValidateUser クラス内で)。

以下のリストで、これらのプロパティについて説明します。

IntrospectionClaims

```
Property IntrospectionClaims as array of %OAuth2.Server.Claim;
```

イントロスペクション・エンドポイントが返すクレームを指定します (必須クレームの基本セットの範囲を超えて指定)。承認サーバは `scope`、`client_id`、`username`、`token_type`、`exp`、`iat`、`nbf`、`sub`、`aud`、`iss`、`jti` クレームを (このプロパティに含まれていない場合でも) 返します。

ほとんどの場合、このプロパティの値には空の文字列を使用できます。このプロパティは、`claims` 要求パラメータをサポートするために指定されます (詳細は、OpenID Connect Core のセクション 5.5 を参照してください)。

形式的にはこのプロパティは配列であり、その配列キーは (ClaimValues プロパティの名前と一致する) クレーム名であり、その配列値は %OAuth2.Server.Claim のインスタンスです。%OAuth2.Server.Claim クラスには以下のプロパティがあります。

- Essential

  ```
  property Essential as %Boolean [ InitialExpression = 0 ];
  ```

  クレームが必須、任意選択のいずれであるかを指定します。値 1 は必須を意味し、値 0 は任意選択を意味します。

- Values

  ```
  property Values as list of %String(MAXLEN=2048);
  ```

  このクレームの許容値のリストを指定します。

通常、クレームの値は ValidateUser クラスで設定されます。

IDTokenClaims

```
Property IDTokenClaims as array of %OAuth2.Server.Claim;
```

承認サーバが IDToken で必要とするクレームを指定します (必須クレームの基本セットの範囲を超えて指定)。承認サーバは、iss、sub、exp、aud、azp クレームを (このプロパティに含まれていない場合でも) 必要とします。

このプロパティはオブジェクトの配列です。詳細は、IntrospectionClaims プロパティの項目を参照してください。

ほとんどの場合、このプロパティの値には空の文字列を使用できます。このプロパティは、claims 要求パラメータをサポートするために指定されます (詳細は、OpenID Connect Core のセクション 5.5 を参照してください)。

UserinfoClaims

```
Property UserinfoClaims as array of %OAuth2.Server.Claim;
```

Userinfo エンドポイントが返すクレームを指定します (必須クレームの基本セットの範囲を超えて指定)。承認サーバは sub クレームを (このプロパティに含まれていない場合でも) 返します。

ほとんどの場合、このプロパティの値には空の文字列を使用できます。このプロパティは、OpenID Connect Core のセクション 5.5 をサポートするために提供されます。

このプロパティはオブジェクトの配列です。詳細は、IntrospectionClaims プロパティの項目を参照してください。

クレームは、スコープと要求のクレーム・パラメータに基づいて定義されます。クレームの戻り値は、ClaimValues プロパティの同じキーを持ちます。通常、クレームの値は ValidateUser クラスで設定されます。

JWTClaims

```
Property JWTClaims as array of %OAuth2.Server.Claim;
```

JWT ベースの既定のアクセス・トークン・クラス (%OAuth2.Server.JWT) が返す JWT アクセス・トークンで必要とされるクレームを必須クレームの基本セットの範囲を超えて指定します。承認サーバは、iss、sub、exp、aud、jti クレームを (このプロパティに含まれていない場合でも) 返します。

このプロパティはオブジェクトの配列です。詳細は、IntrospectionClaims プロパティの項目を参照してください。

このクレームはカスタマイズ・コードによって定義されます。通常、クレームの値は ValidateUser クラスで設定されます。

ClaimValues

```
property ClaimValues as array of %String(MAXLEN=1024);
```

実際のクレーム値とその型を指定します。このプロパティを操作するには、次のセクションのメソッドを使用します。

このプロパティを直接操作する必要がある場合は、このプロパティが以下の特徴を持つ配列であることに注意してください。

- 配列キーはクレーム名です。
- 配列値は $LISTBUILD(type,value) の形式をとります。ここで、type は値のタイプを保持し、value は実際の値を保持します。type には "string"、"boolean"、"number"、または "object" を指定できます。type が "object" の場合、value は文字列としてシリアル化された JSON オブジェクトとなります。

  value は $LIST 構造となります。この場合、クレーム値はシリアル化されるときに JSON 配列としてシリアル化され、それぞれの配列項目は指定された type となります。

5.4.3 クレームを操作するメソッド

%OAuth2.Server.Properties クラスは、ClaimValues プロパティの操作を簡素化するために使用できるインスタンス・メソッドも提供します。

SetClaimValue()

```
Method SetClaimValue(name As %String, value As %String, type As %String = "string")
```

name 引数で指定したクレームの値を設定することで、ClaimValues プロパティを更新します。type 引数はクレームの型を示します。型には "string" (既定値)、"boolean"、"number"、"object" があります。type が "object" の場合、value は文字列としてシリアル化された JSON オブジェクトでなければなりません。

value は $LIST 構造にすることもできます。この場合、クレーム値はシリアル化されるときに JSON 配列としてシリアル化され、それぞれの配列項目は指定された type となります。

RemoveClaimValue()

```
Method RemoveClaimValue(name As %String)
```

name 引数で指定したクレームを削除することで、ClaimValues プロパティを更新します。

GetClaimValue()

```
Method GetClaimValue(name As %String, output type) As %String
```

ClaimValues プロパティを調べて、name 引数で指定されたクレームの値を返します。出力として返される type 引数はクレームの型を示します。SetClaimValue() を参照してください。

NextClaimValue()

```
Method NextClaimValue(name As %String) As %String
```

指定されたクレームの後にある (ClaimValues プロパティの) 次のクレームの名前を返します。

5.5 承認サーバ・エンドポイントの場所

OAuth 2.0 承認サーバとして InterSystems IRIS インスタンスを使用する場合、承認エンドポイントの URL は次のようになります。

エンドポイント	URL
発行者エンドポイント	`https://serveraddress/oauth2`
承認エンドポイント	`https://serveraddress/oauth2/authorize`
トークン・エンドポイント	`https://serveraddress/oauth2/token`
Userinfo エンドポイント	`https://serveraddress/oauth2/userinfo`
トークン・イントロスペクション・エンドポイント	`https://serveraddress/oauth2/introspection`
トークン取り消しエンドポイント	`https://serveraddress/oauth2/revocation`

いずれの場合も、serveraddress は InterSystems IRIS インスタンスを実行しているサーバの IP アドレスまたはホスト名になります。

5.6 InterSystems IRIS OAuth 2.0 承認サーバでのクライアント定義の作成

このセクションでは、クライアントを動的に登録していない場合に、InterSystems IRIS OAuth 2.0 承認サーバでクライアント定義を作成する方法について説明します。まず、このページの前述の説明に従って InterSystems IRIS OAuth 2.0 承認サーバを設定します。管理ポータルで以下の手順を実行します。

1. **[システム管理]**→**[セキュリティ]**→**[OAuth2.0]**→**[サーバ構成]** を選択します。
2. **[クライアント構成]** ボタンをクリックしてクライアント記述を表示します。このテーブルは最初は空です。
3. **[一般]** タブで、以下の詳細を指定します。
   - **[名前]** – このクライアントの一意の名前を指定します。
   - **[説明]** – 必要に応じて説明を入力します。
   - **[クライアント・タイプ]** – このクライアントのタイプを指定します。選択項目には **[パブリック]** (パブリック・クライアント、RFC 6749 に準拠)、**[機密]** (機密クライアント、RFC 6749 に準拠)、**[リソース]** (専用のリソース・サーバ) があります。
   - **[リダイレクト URL]** – このクライアントで予期される 1 つ以上のリダイレクト URL。
   - **[サポートされている付与タイプ]** – このクライアントがアクセス・トークンを作成するために使用できる付与タイプを指定します。少なくとも 1 つ選択します。
   - **[サポートされるレスポンス・タイプ]** – クライアントが限定使用するOAuth 2.0 response_type値を選択します。
   - **[認証タイプ]** – 承認サーバへの HTTP 要求で使用する認証タイプを選択します (RFC 6749 または OpenID Connect Core のセクション 9 で規定)。以下のいずれかを選択します。
     - なし

  - **基本**
  - **フォームエンコードされた本文**
  - **クライアント秘密鍵 JWT**
  - **秘密鍵 JWT**

- **[認証署名アルゴリズム]** – トークン・エンドポイントでこのクライアントを認証するために使用される JWT の署名に必要なアルゴリズムを選択します ([認証タイプ] が **[クライアント秘密鍵 JWT]** または **[秘密鍵 JWT]** である場合)。オプションを選択しない場合、OpenID プロバイダおよび Relying Party でサポートされる任意のアルゴリズムが使用されます。

4. 必要な場合は、**[クライアント資格情報]** タブを選択し、次の詳細を表示します。
   - **[クライアント ID]** – RFC 6749 で規定されているクライアント ID。InterSystems IRIS がこの文字列を生成します。
   - **[クライアント秘密鍵]** – RFC 6749 で規定されているクライアント秘密鍵。InterSystems IRIS がこの文字列を生成します。
5. **[クライアント情報]** タブで、以下の詳細を指定します。
   - **[起動 URL]** – このクライアントの起動に使用する URL を指定します。状況によっては、この値をクライアントの識別に使用することも、aud クレームの値として使用することもできます。
   - **[承認の表示]** セクション:
     - **[クライアント名]** – エンド・ユーザに表示するクライアントの名前を指定します。
     - **[ロゴ URL]** – クライアント・アプリケーションのロゴを参照する URL を指定します。このオプションを指定すると、承認サーバは承認中にこのイメージをエンド・ユーザに表示します。このフィールドの値は、有効なイメージ・ファイルを指している必要があります。
     - **[クライアント・ホーム・ページ]** – クライアントのホーム・ページの URL を指定します。このフィールドの値は、有効な Web ページを指している必要があります。このオプションを指定すると、承認サーバは、この URL を分かりやすい形でエンド・ユーザに示します。
     - **[ポリシー URL]** – プロファイル・データの取り扱い方法を確認できるよう Relying Party Client エンド・ユーザに提供する URL を指定します。このフィールドの値は、有効な Web ページを指している必要があります。
     - **[サービス条件の URL]** – Relying Party のサービス条件を確認できるように Relying Party Client がエンド・ユーザに提供する URL を指定します。このフィールドの値は、有効な Web ページを指している必要があります。
   - **[連絡先電子メール]** – クライアント・アプリケーションの責任者への連絡に使用する適切な電子メール・アドレスをコンマで区切ったリストです。
   - **[既定の最長経過時間]** – 既定の最長認証経過時間 (秒単位) を指定します。このオプションを指定した場合、最長認証経過時間に達したとき、エンド・ユーザをアクティブに再認証する必要があります。max_age 要求パラメータは、この既定値をオーバーライドします。このオプションを省略した場合に、既定で最長認証経過時間は設定されません。
   - **[既定のスコープ]** – アクセス・トークン要求の既定のスコープを空白で区切ったリストで指定します。
6. **[JWT 設定]** タブで、以下の詳細を指定します。
   - **[JSON Web Token (JWT) の設定]** – 承認サーバから JWT のシグニチャを検証し、承認サーバに送信される JWT を暗号化するためにクライアントが使用する公開鍵のソースを指定します。

既定では、ダイナミック登録プロセスは JWKS (JSON Web Key Set) のペアを生成します。一方の JWKS は秘密鍵で、(アルゴリズムあたり) 必要なすべての秘密鍵だけでなく、対称鍵として使用するクライアント秘密鍵も含んでいます。この JWKS は共有されません。他方の JWKS には、対応する公開鍵が含まれていて、公開されて利用できます。ダイナミック登録プロセスは、公開の JWKS のクライアントへのコピーも実行します。

その他のオプションは以下のとおりです。

- **[URL から JWKS]** ― 公開の JWKS を指す URL を指定した後、InterSystems IRIS に JWKS をロードします。
- **[ファイルから JWKS]** ― 公開の JWKS を含むファイルを選択し、そのファイルを InterSystems IRIS にロードします。
- **[X509 証明書]** ― 詳細は、"証明書と JWT (JSON Web Token)" の "OAuth 2.0 承認サーバの証明書の使用" を参照してください。

これらのオプションのいずれかにアクセスするには、まず **[ダイナミック登録以外のソース]** を選択します。

7. **[保存]** を選択します。

5.7 JWT で使用するキーの回転

ほとんどの場合、新しい公開/秘密鍵ペアを承認サーバに生成させることができます。これは、非対称の RS256、RS384、および RS512 の各アルゴリズムに使用する RSA 鍵にのみ適用されます。(例外は、**[X509 証明書]** として、**[ダイナミック登録以外のソース]** を指定する場合です。この場合、新しいキーは生成できません。)

新しい公開/秘密鍵ペアの生成は、キーの回転と呼ばれています。このプロセスは、新しい秘密 RSA 鍵および関連する公開 RSA 鍵を秘密と公開の JWKS に追加します。

承認サーバでキーの回転を実行すると、承認サーバは新しい秘密 RSA 鍵を使用して、クライアントに送信する JWT に署名します。同様に、承認サーバは新しい公開 RSA 鍵を使用して、クライアントに送信する JWT を暗号化します。承認サーバは、クライアントから受信した JWT を解読するために新しい RSA 鍵を使用しますが、これが失敗したら古い RSA 鍵を使用するため、古い公開 RSA 鍵を使用して作成された JWT を解読できます。

最後に、クライアントから受信した署名済み JWT を承認サーバが検証できない場合、承認サーバにクライアントの公開の JWKS の URL があると、承認サーバは新しい公開の JWKS を取得し、署名の検証を再試行します。(ダイナミック Discovery を使用した場合や、構成で **[URL から JWKS]** オプションが指定された場合、承認サーバはクライアントの公開の JWKS の URL を持っています。それ以外の場合には承認サーバはこの URL を持っていません。)

承認サーバのキーを回転するには、以下の手順を実行します。

1. **[システム管理]**→**[セキュリティ]**→**[OAuth 2.0]**→**[サーバ構成]** を選択します。

   承認サーバの構成が表示されます。

2. **[キーの回転]** ボタンを選択します。

注釈　対称の HS256、HS384、および HS512 の各アルゴリズムは、常に対称鍵としてクライアントの秘密鍵を使用します。

5.7.1 承認サーバのキーの回転の API

承認サーバでキーをプログラムによって回転させるには、**OAuth2.Server.Configuration** の RotateKeys() メソッドを呼び出します。

新しいクライアントの JWKS を取得するには、OAuth2.Server.Client の UpdateJWKS() メソッドを呼び出します。

これらのメソッドの詳細は、クラス・リファレンスを参照してください。

5.8 クライアントからの新しい公開の JWKS の取得

ほとんどの場合、クライアントは JWKS の公開/秘密鍵ペアを生成します。公開の JWKS を承認サーバが受信する方法はさまざまです。1 つの方法は、公開の JWKS をクライアントが URL で指定する方法です。"InterSystems IRIS OAuth 2.0 承認サーバでのクライアント定義の作成" の [URL から JWKS] オプションを参照してください。

クライアントが [URL から JWKS] で定義されていて、クライアントが JWKS の新しいペアを生成する場合、同じ URL から新しい公開の JWKS を承認サーバに取得させることができます。そのためには、以下の操作を実行します。

1. 管理ポータルで [**システム管理**]→[**セキュリティ**]→[OAuth 2.0]→[**サーバ構成**] を選択します。

   承認サーバの構成が表示されます。

2. [**JWKS の更新**] ボタンを選択します。

クライアントが [URL から JWKS] で定義されておらず、クライアントが JWKS の新しいペアを生成する場合、公開の JWKS を取得し、これを承認サーバに送信し、ファイルからロードする必要があります。

A
プログラムによる構成項目の作成方法

管理ポータルを使用して OAuth 2.0 のクライアント、リソース・サーバ、承認サーバを構成する方法については他の記事で説明しています。これらの構成をプログラムで作成することもできます。以下のサブセクションでは、クライアント (リソース・サーバを含む) と承認サーバの作成方法について詳しく説明します。

A.1 プログラムによるクライアント構成項目の作成方法

OAuth 2.0 クライアントまたは OAuth 2.0 リソース・サーバの構成項目をプログラムで作成するには、以下の手順を実行します。

1. サーバ記述を作成します。
2. 関連付けられたクライアント構成を作成します。

A.1.1 サーバ記述の作成

サーバ記述は、OAuth2.ServerDefinition のインスタンスです。サーバ記述を作成するには次の手順に従います。

1. %SYS ネームスペースに切り替えます。
2. 承認サーバが Discovery をサポートする場合は、%SYS.OAuth2.Registration の Discover() メソッドを呼び出します。このメソッドは、以下のとおりです。

```
ClassMethod Discover(issuerEndpoint As %String,
                     sslConfiguration As %String,
                     Output server As OAuth2.ServerDefinition) As %Status
```

   引数は以下のとおりです。

   - issuerEndpoint は、承認サーバの識別に使用するエンドポイントの URL を指定します。
   - sslConfiguration は、Discover() メソッドの呼び出しに使用する InterSystems IRIS SSL/TLS 構成のエイリアスを指定します。
   - 出力として返される server は、OAuth2.ServerDefinition のインスタンスです。

3. 次に、返された OAuth2.ServerDefinition インスタンスを保存します。

また、承認サーバが Discovery をサポートしない場合、次の手順を実行します。

1. %SYS ネームスペースに切り替えます。

2. OAuth2.ServerDefinition のインスタンスを作成します。
3. インスタンスのプロパティを設定します。ほとんどの場合、プロパティの名前は管理ポータルに表示されるラベルと一致します (スペースと大文字化を除く)。詳細は、"サーバ記述の手動による作成" を参照してください。これらのプロパティは、以下のとおりです。
   - IssuerEndpoint
   - SSLConfiguration
   - InitialAccessToken。[登録アクセス・トークン] フィールドに対応します。
   - Metadata。OAuth2.Server.Metadata のインスタンスで、多くのプロパティが含まれます。OpenID Provider Metadata (https://openid.net/specs/openid-connect-discovery-1_0.html) を参照してください。

   ServerCredentials の詳細は、"証明書と JWT (JSON Web Token)" の "OAuth 2.0 承認サーバの証明書の使用" を参照してください。
4. インスタンスを保存します。

A.1.2 クライアント構成の生成

クライアント構成は、OAuth2.Client のインスタンスです。クライアント構成を作成するには、次の手順を実行します。

1. %SYS ネームスペースに切り替えます。
2. OAuth2.Client のインスタンスを作成します。
3. インスタンスのプロパティを設定します。ほとんどの場合、プロパティの名前は管理ポータルに表示されるラベルと一致します (スペースと大文字化を除く)。詳細は、"クライアントの構成および動的な登録" を参照してください。これらのプロパティは、以下のとおりです。
   - ApplicationName
   - ClientId。クライアントを動的に登録する場合、手動で設定する必要はありません。
   - ClientSecret。クライアントを動的に登録する場合、手動で設定する必要はありません。
   - DefaultScope
   - Description
   - Enabled
   - JWTInterval
   - Metadata。OAuth2.Client.Metadata のインスタンスで、多くのプロパティが含まれます。詳細は、Client Metadata (http://openid.net/specs/openid-connect-registration-1_0-19.html) を参照してください。
   - RedirectionEndpoint。[応答を受信するために承認サーバに対して指定されるクライアント URL] オプションに対応します。このプロパティのタイプは %OAuth2.Endpoint です。OAuth2.Endpoint クラスは、UseSSL、Host、Port、Prefix のプロパティを持つシリアル・クラスです。
   - SSLConfiguration
   - ServerDefinition。以前に作成した OAuth2.ServerDefinition のインスタンスであることが必要です。

   ClientCredentials の詳細は、"証明書と JWT (JSON Web Token)" の "OAuth 2.0 クライアントの証明書の使用" を参照してください。

4. 承認サーバがダイナミック・クライアント登録をサポートする場合は、%SYS.OAuth2.Registration の RegisterClient() メソッドを呼び出します。このメソッドは、以下のとおりです。

```
ClassMethod RegisterClient(applicationName As %String) As %Status
```

applicationName はクライアント・アプリケーションの名前です。

このメソッドはクライアントを登録し、(クライアント ID およびクライアント秘密鍵を含む) クライアント・メタデータを取得した後、OAuth2.Client のインスタンスを更新します。

A.2 プログラムによるサーバ構成項目の作成方法

OAuth 2.0 承認サーバの構成項目をプログラムで作成するには、以下の手順を実行します。

1. 承認サーバ構成を作成します。

   1 つの InterSystems IRIS インスタンス上に複数の承認サーバ構成を定義できないことに注意してください。また、この構成を作成するには、%Admin_Secure リソースの USE 権限を持つユーザとしてログインする必要があります。

2. 関連付けられたクライアント記述を作成します。

A.2.1 承認サーバ構成の作成

認証サーバ構成は、OAuth2.Server.Configuration のインスタンスです。承認サーバ構成を作成するには、次の手順を実行します。

1. %SYS ネームスペースに切り替えます。
2. OAuth2.Server.Configuration のインスタンスを作成します。
3. インスタンスのプロパティを設定します。 ほとんどの場合、プロパティの名前は管理ポータルに表示されるラベルと一致します (スペースと大文字化を除く)。詳細は、"承認サーバの構成" を参照してください。これらのプロパティは、以下のとおりです。
   - AccessTokenInterval
   - AllowUnsupportedScope
   - AudRequired。[対象者は必須] オプションに対応します
   - AuthenticateClass
   - AuthorizationCodeInterval
   - ClientSecretInterval
   - CustomizationNamespace
   - CustomizationRoles
   - DefaultScope
   - Description
   - EncryptionAlgorithm
   - GenerateTokenClass
   - IssuerEndpoint。[発行者エンドポイント] オプションに対応します。OAuth2.Endpoint タイプです。OAuth2.Endpoint クラスは、UseSSL、Host、Port、Prefix のプロパティを持つシリアル・クラスです。

- JWKSFromCredentials
- KeyAlgorithm
- Metadata。OAuth2.Server.Metadata のインスタンスで、多くのプロパティが含まれます。OpenID Provider Metadata (https://openid.net/specs/openid-connect-discovery-1_0.html) を参照してください。
- RefreshTokenInterval
- ReturnRefreshToken
- SSLConfiguration
- SessionClass
- SessionInterval。[**セッション終了間隔**] オプションに対応します
- SigningAlgorithm
- SupportSession。[**ユーザ・セッションのサポート**] オプションに対応します
- SupportedScopes。[**スコープ**] と [**説明**] の列を持つテーブルに対応します。このプロパティは文字列の配列であるため、SetAt()、GetAt() などの通常の配列インタフェースを使用します。
- ValidateUserClass

署名、キー管理、暗号化のアルゴリズムに使用できる値については、%OAuth2.JWT のクラス・リファレンスを参照してください。

ServerCredentials および ServerPassword の詳細は、"証明書と JWT (JSON Web Token)" の "OAuth 2.0 承認サーバの証明書の使用" を参照してください。

4. `OAuth2.Server.Configuration.Save()` メソッドを使用してインスタンスを保存します。`%Save()` メソッドではなく `Save()` メソッドを使用する必要があります。このメソッドには、Web アプリケーションの作成などの追加の機能が用意されているからです。

InterSystems IRIS では、このクラスの複数インスタンスの使用はサポートされていません。

また、このインスタンスを保存するには、%Admin_Secure リソースの USE 権限を持つユーザとしてログインする必要があります。

A.2.2 クライアント記述の作成

クライアント記述は、OAuth2.Server.Client のインスタンスです。クライアント記述を作成するには、次の手順を実行します。

1. %SYS ネームスペースに切り替えます。
2. OAuth2.Server.Client のインスタンスを作成します。
3. インスタンスのプロパティを設定します。ほとんどの場合、プロパティの名前は管理ポータルに表示されるラベルと一致します (スペースと大文字化を除く)。詳細は、"クライアント記述の作成" を参照してください。これらのプロパティは、以下のとおりです。
   - ClientCredentials
   - ClientType
   - DefaultScope
   - Description
   - LaunchURL

- **Metadata**。OAuth2.Client.Metadata のインスタンスで、多くのプロパティが含まれます。詳細は、Client Metadata (http://openid.net/specs/openid-connect-registration-1_0-19.html) を参照してください。
- **Name**
- **RedirectURL**。[**リダイレクト URL**] オプションに対応します。このプロパティは文字列の配列であるため、SetAt()、GetAt() などの通常の配列インタフェースを使用します。

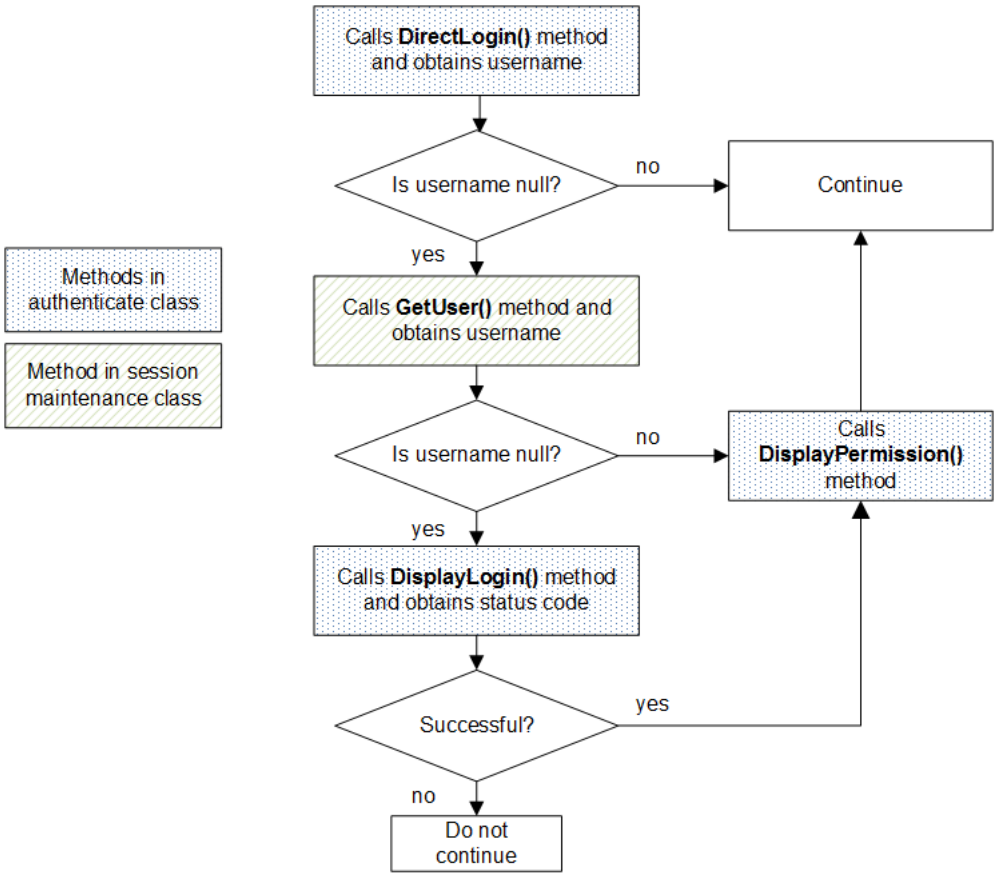
4. インスタンスを保存します。

   システムは、**ClientId** および **ClientSecret** プロパティの値を生成します。

B

DirectLogin() の実装

OAuth 2.0 承認サーバとして InterSystems IRIS® を使用するときは、通常、認証クラスの DisplayLogin() メソッドを実装します。このメソッドは、ユーザがユーザ名とパスワードを入力してログインするページを表示します。サーバによる認証時にログイン・フォームを表示せず、現在のセッションも使用されないようにする場合は、認証クラスの DirectLogin() メソッドを実装します。以下のフローチャートは、アクセス・トークンの要求を処理するときに InterSystems IRIS 承認サーバがユーザを識別する仕組みを示しています。



既定では、GetUser() メソッドは、前のログインで入力されたユーザ名を取得します。

DirectLogin() を実装すると、DisplayPermissions() は呼び出されません。DirectLogin() は許可を表示する役割を果たすからです。

DirectLogin() メソッドには、以下のシグニチャがあります。

```
ClassMethod DirectLogin(scope As %ArrayOfDataTypes,
                        properties As %OAuth2.Server.Properties,
                        Output username As %String,
                        Output password As %String) As %Status
```

以下はその説明です。

- scope は、元のクライアント要求に含まれるスコープを含む %ArrayOfDataTypes のインスタンスです。このスコープは、BeforeAuthenticate() メソッドによって変更されている可能性があります。配列キーはスコープ値であり、配列値はスコープ値の対応する表示形式です。
- properties は、承認サーバが受信し、処理の早い段階にメソッドが変更したプロパティとクレームを含む %OAuth2.Server.Properties のインスタンスです。"%OAuth2.Server.Properties オブジェクトの詳細" を参照してください。
- 出力として返される username はユーザ名です。
- 出力として返される password は対応するパスワードです。

実装では、独自のロジックを使用して username および password 引数を設定します。そのためには、必要に応じて scope および properties 引数を使用します。アクセスを拒否するには、メソッドで username 引数を $char(0) に設定します。この場合、承認サーバは `access_denied` のエラーを返します。

メソッドで properties のプロパティを設定することもできます。このオブジェクトは後続の処理で利用できます。

このメソッドは %Status を返す必要があります。

C

証明書と JWT (JSON Web Token)

OAuth 2.0 のそれぞれのパーティが公開/秘密鍵ペアを持っている必要があります。これらの鍵ペアに対して、証明書およびその秘密鍵を使用できます。ただし、これは一般的な手法ではありません。このページでは、次の各シナリオの詳細について説明します。

- OAuth 2.0 クライアント
- OAuth 2.0 リソース・サーバ
- OAuth 2.0 承認サーバ

いずれの場合でも、秘密鍵と対応する証明書の生成では、インターシステムズ公開鍵インフラストラクチャを使用できます。

注釈　InterSystems IRIS は JWKS (JSON Web Key Set) のペアを生成できます。一方の JWKS は秘密鍵で、(アルゴリズムあたり) 必要なすべての秘密鍵だけでなく、対称鍵として使用するクライアント秘密鍵も含んでいます。この JWKS は共有されません。他方の JWKS には、対応する公開鍵が含まれていて、公開されて利用できます。JWKS を生成するオプションを使用する場合は、このページは無視してください。

C.1 OAuth 2.0 クライアントの証明書の使用

OAuth 2.0 クライアントは承認サーバから (暗号化や署名を施した) JWT を受信できます。同様に、このクライアントは承認サーバへ (暗号化や署名を施した) JWT を送信することができます。これらの目的で証明書/秘密鍵のペアを使用する場合、以下の表を参考にして必要になる証明書を特定してください。

シナリオ	クライアント構成の要件
以下のいずれかの場合 : · クライアントは、承認サーバが送信した JWT のシグニチャを検証する必要がある · クライアントは、承認サーバへ送信する JWT を暗号化する必要がある	承認サーバが所有する証明書と、サーバ証明書に署名する CA (認証機関) 証明書を取得します。この証明書の公開鍵は、シグニチャ検証と暗号化で使用されます。
以下のいずれかの場合 : · クライアントは、JWT に署名してから承認サーバへ送信する必要がある · クライアントは、承認サーバが送信した JWT を復号化する必要がある	クライアント用の秘密鍵を取得し、対応する証明書とその証明書に署名する CA 証明書を取得します。秘密鍵を署名と復号化に使用します。

いずれの場合も、そのクライアント Web アプリケーションを含む同じインスタンスで次の手順も実行する必要があります。

- InterSystems IRIS が使用する信頼できる証明書を用意します。信頼された証明書には、クライアントの証明書に署名する証明書と承認サーバの証明書を組み込む必要があります (必要とする証明書に応じていずれかまたは両方)。
- InterSystems IRIS で証明書を使用できるようにするための InterSystems IRIS 資格情報セットを作成します。

  クライアント証明書の場合は、資格情報セットを作成するときに秘密鍵を読み込み、秘密鍵のパスワードを入力します。
- クライアントを構成する場合は、**[X509 資格情報から JWT 設定を作成]** オプションを選択します。また、以下の指定を行います。
  - **[X.509 証明書]** – クライアントの証明書を使用し対応秘密鍵 (`ClientConfig` など) を含む資格情報セットを選択します。
  - **[秘密鍵パスワード]** – この証明書の秘密鍵のパスワードを入力します。

C.2 OAuth 2.0 リソース・サーバの証明書の使用

OAuth 2.0 リソース・サーバは、承認サーバから (暗号化や署名を施した) JWT を受信できます。同様に、リソース・サーバは承認サーバへ (暗号化や署名を施した) JWT を送信することができます。これらの目的で証明書/秘密鍵のペアを使用する場合、以下の表を参考にして必要になる証明書を特定してください。

シナリオ	リソース・サーバ構成の要件
リソース・サーバは、承認サーバが送信した JWT のシグニチャを検証する必要がある	承認サーバが所有する証明書と、サーバ証明書に署名する CA 証明書を取得します。この証明書の公開鍵は、シグニチャ検証と暗号化で使用されます。
リソース・サーバは、承認サーバへ送信する JWT を暗号化する必要がある	
リソース・サーバは、JWT に署名してから承認サーバへ送信する必要がある	リソース・サーバ用の秘密鍵を取得し、対応する証明書とその証明書に署名する CA 証明書を取得します。秘密鍵は署名と復号化に使用します。
リソース・サーバは、承認サーバが送信した JWT を復号化する必要がある	

いずれの場合も、そのリソース・サーバ Web アプリケーションを含む同じインスタンスで次の手順も実行する必要があります。

- InterSystems IRIS が使用する信頼できる証明書を用意します。信頼された証明書には、リソース・サーバの証明書に署名する証明書と承認サーバの証明書を組み込む必要があります (必要とする証明書に応じていずれかまたは両方)。
- InterSystems IRIS で証明書を使用できるようにするための InterSystems IRIS 資格情報セットを作成します。

  リソース・サーバの証明書では、資格情報セットを作成するときに秘密鍵を読み込み、秘密鍵のパスワードを入力します。
- リソース・サーバを構成する場合は、[**X509 資格情報から JWT 設定を作成**] オプションを選択します。また、以下の指定を行います。
  - [**X.509 証明書**] – リソース・サーバの証明書を使用し対応秘密鍵 (`ResourceConfig` など) を含む資格情報セットを選択します。
  - [**秘密鍵パスワード**] – この証明書の秘密鍵のパスワードを入力します。

C.3 OAuth 2.0 承認サーバの証明書の使用

OAuth 2.0 承認サーバは、そのクライアントから (暗号化や署名を施した) JWT を受信できます。同様に、承認サーバはそのクライアントへ (暗号化や署名を施した) JWT を送信することができます。これらの目的で証明書/秘密鍵のペアを使用する場合、以下の表を参考にして必要になる証明書を特定してください。

シナリオ	承認サーバ構成の要件
承認サーバは、クライアントが送信した JWT のシグニチャを検証する必要がある	そのクライアントが所有する証明書と、証明書に署名する CA 証明書を取得します。この証明書の公開鍵は、シグニチャ検証と暗号化で使用されます。
承認サーバは、クライアントへ送信する JWT を暗号化する必要がある	
承認サーバは、JWT に署名してからクライアントへ送信する必要がある	承認サーバ用の秘密鍵を取得し、対応する証明書とその証明書に署名する CA 証明書を取得します。秘密鍵は署名と復号化に使用します。
承認サーバは、クライアントが送信した JWT を復号化する必要がある	

いずれの場合も、その承認サーバを含む同じインスタンスで次の手順も実行する必要があります。

- InterSystems IRIS が使用する信頼できる証明書を用意します。信頼された証明書には、クライアントの証明書に署名する証明書と承認サーバの証明書を組み込む必要があります (必要とする証明書に応じていずれかまたは両方)。
- InterSystems IRIS で証明書を使用できるようにするための InterSystems IRIS 資格情報セットを作成します。

  承認サーバの証明書では、資格情報セットを作成するときに秘密鍵を読み込み、秘密鍵のパスワードを入力します。
- サーバを構成する場合、[**JWT 設定**] タブを選択します。このタブで、[**X509 資格情報から JWT 設定を作成**] オプションを選択します。また、以下の指定を行います。
  - [**X509 証明書**] – 承認サーバの証明書を使用し対応秘密鍵 (`AuthConfig` など) を含む資格情報セットを選択します。
  - [**秘密鍵パスワード**] – この証明書の秘密鍵のパスワードを入力します。
- サーバ上でクライアント定義を作成する場合、[**JWT 設定**] タブを選択します。このタブで、[**ダイナミック登録以外のソース**] では、[**X509 証明書**] を選択します。また、[**クライアント資格情報**] では、クライアント証明書を使用する認証情報セット (`ClientConfig` など)を選択します。

D

JWT ヘッダの操作

ここでは、JWT ヘッダをカスタマイズする方法と、JWT ヘッダ内のカスタム値を処理する方法を説明します。JWT は、承認サーバ、または OAuth フレームワーク外部のカスタム・コードによって生成されます。

D.1 ヘッダ値の追加 (承認サーバ)

承認サーバによって JWT トークンが生成される場合、使用される署名アルゴリズムと暗号化アルゴリズムに基づいて、`alg`、`enc`、`kid`、`typ`、および `cty` の各ヘッダが自動的に生成されます。これらのヘッダをカスタム・コードで直接操作することはできません。ただし、他のヘッダ値は JWTHeaderClaims 配列を使用して JWT ヘッダに追加できます。例えば、`JWTHeaderClaims` を使用して、`jku` および `jwk` のヘッダ・パラメータをトークンに追加できます。`jku` の場合、関連する `jwk_uri` または `JWKS` がサーバによって自動的に JOSE 配列に追加されます。RFC 7515 で定義されているヘッダ・パラメータの一部はサポートされないことに注意してください。`JWTHeaderClaims` 配列を使用して、任意のカスタム値を JWT ヘッダに追加することもできます。

例えば、以下のコードを %OAuth2.Server.Validate のサブクラスに追加すると、2 つの標準ヘッダと 1 つのカスタム・ヘッダ値を JWT ヘッダに追加できます。

```
ClassMethod ValidateUser(username As %String, password As %String, scope As %ArrayOfDataTypes, properties
 As %OAuth2.Server.Properties, Output sc As %Status) As %Boolean
{
  ...
  Do properties.SetClaimValue("co","intersystems")

  Do properties.JWTHeaderClaims.SetAt("","jku")
  Do properties.JWTHeaderClaims.SetAt("","co")
  Do properties.JWTHeaderClaims.SetAt("","iss")
  ...
}
```

D.2 ヘッダ値の追加 (JWT の直接生成)

ObjectToJWT() メソッドを使用して、カスタム・コードによって OAuth フレームワークの外部で JWT を生成できます。このメソッドは、JOSE ヘッダを表す文字列の配列を最初のパラメータとして受け入れます。JWT ヘッダに値を追加するに

は、ObjectToJWT メソッドを呼び出す前に JOSE 配列に値を追加するだけです。例えば、jku ヘッダ・パラメータを JOSE ヘッダに追加するには、以下のように指定します。

```
Set JOSE("jku")=""
Set JOSE("jku_local")=%server.Metadata."jwks_uri"
Set JOSE("jku_remote")=%client.Metadata."jwks_uri"
// set JOSE("sigalg") and/or JOSE("encalg") and JOSE("keyalg")
Set sc=##class(%OAuth2.JWT).ObjectToJWT(.JOSE,json,%server.PrivateJWKS,%client.RemotePublicJWKS,.JWT)
```

標準のヘッダ・パラメータに対応する JOSE 配列ノードの詳細は、クラス・リファレンスを参照してください。

D.2.1 カスタム・ヘッダ・パラメータの追加

ObjectToJWT メソッドに渡される JOSE 配列には、JWT ヘッダに挿入するカスタム・ヘッダ・パラメータを追加できます。カスタム・ヘッダ・パラメータを追加するには、まず、それらのヘッダ・パラメータをキー/値のペアとしてダイナミック・オブジェクト内に定義します。キーはカスタム・パラメータの名前、値はそのパラメータの値です。ダイナミック・オブジェクトを定義したら、custom 添え字を使用してそのダイナミック・オブジェクトを JOSE 配列のノードに追加します。例えば、以下のコードでは、co と prod の 2 つのカスタム・パラメータが JWT ヘッダに挿入されます。

```
Set newParams={"co":"intersystems","prod":"bazbar"}
Set JOSE("custom")=newParams
// set JOSE("sigalg") and/or JOSE("encalg") and JOSE("keyalg")
Set sc=##class(%OAuth2.JWT).ObjectToJWT(.JOSE,json,%server.PrivateJWKS,%client.RemotePublicJWKS,.JWT)
```

JOSE 配列の custom ノードを使用して、RFC 7515 で定義されているヘッダ値をオーバーライドすることはできません。入れ子になった署名または暗号化を行う場合、カスタム・ヘッダは "内部" (署名された) トークンにのみ追加されます。

D.3 カスタム・ヘッダ・パラメータの処理

JWTToObject メソッドを使用すると、JWT を処理してヘッダを返すことができます。これらのヘッダにアクセスする方法は 2 つあります。このメソッドでは、JWT に対する署名/暗号化操作に使用するアルゴリズムが含まれる標準の JOSE ヘッダ・パラメータは、文字列の配列で返されます。また、JWT の "未加工" のヘッダはダイナミック・オブジェクトとして 6 番目のパラメータで返されます。このダイナミック・オブジェクトのキー/値ペアを解析することで、JWT ヘッダに存在するカスタムおよび標準のヘッダ・パラメータを処理できます。トークンが、入れ子になった署名および暗号化を使用して作成されている場合、このメソッドで返される未加工のヘッダは、"内部" (署名された) トークンからのものです。