Java Web アプリケーションのための OpenID: 第 1 回

Java Web アプリケーションを OpenID 認証に対応させる

J Steven Perry
Principal Consultant
Makoto Consulting Group, Inc.

2010年 3月 16日
(初版 2010年 1月 27日)

OpenID は、さまざまな Java™ Web アプリケーションのリソースにユーザーが簡単にアクセスできるようにする分散型認証プロトコルです。この 2 回連載の前半では、OpenID 認証の仕様について概説し、OpenID を Java アプリケーションに組み込む方法をサンプル Java アプリケーションを用いて説明します。記事の著者である J. Steven Perry は、OpenID 認証の仕様を手作業で実装する手間を省くために OpenID4java ライブラリーを使用しています。このライブラリーと知名度の高い myOpenID という OpenID プロバイダを利用して、Wicket で作成した Java アプリケーションのための安全かつ信頼性に優れた登録プロセスを作成します。

このシリーズの他の記事を見る

OpenID は、分散型認証メカニズムです。OpenID を使用すれば、例えば http://OpenID.jstevenperry.com/steve という URI の所有者が自分であることを証明し、その ID を使用して、Google や Slashdot、そして Wordpress の他、OpenID をサポートしているあらゆるサイトで自分の身元を証明することができます。OpenID がエンド・ユーザーにとって素晴らしい認証メカニズムであることは確かです。けれども OpenID を使用しているうちに、「私が顧客のために作成する Java ベースの Web アプリケーションに対しても、OpenID を使用して標準的な信頼性に優れた認証システムを作成できないものだろうか」という考えが浮かんできました。

この 2 回の連載記事では、OpenID4java ライブラリーと知名度の高い OpenID プロバイダである myOpenID を利用して、Java ベースの Web アプリケーション用の認証システムを作成する方法を紹介します。また、SReg (OpenID Simple Registration Extension) を使ってユーザー情報を入手する方法についても説明します。

連載第 1 回では、まず OpenID の概要を紹介し、自分個人の OpenID を取得する方法を説明します。次に OpenID 認証が機能する仕組みを概説した後、OpenID4java を使って OpenID 認証を行うために必要なステップを順に説明します。続く第 2 回では、独自の OpenID プロバイダを作成する方法を取り上げます。

連載全体をとおして使用するアプリケーションは、私がこの記事のために作成した Wicket ベースの Java Web アプリケーションです。このアプリケーションのソース・コードは随時ダウンロー

ドすることができます。また、OpenID4java ライブラリーを参照してみることもお勧めします (「参考文献」を参照)。

注: この記事で焦点とするのは Java Web アプリケーションに OpenID を使用する方法ですが、OpenID はあらゆるソフトウェア・アーキテクチャーのシナリオで機能します。

OpenID の紹介

OpenID は、ユーザーが ID を持っていることを証明するための仕様です。とりあえず、ID とはユーザーを一意に識別する `String` のことだと思ってください。皆さんも私と同様にたくさんの ID やユーザー ID を持っているのではないでしょうか。私は Facebook のユーザー ID と Twitter のユーザー ID の他、インターネットで使用している数十のサイトでの ID を持っています。使用するユーザー ID を統一したいとは思っているのですが、その ID を新しくサインアップするすべてのサイトで使用できるとは限りません。そのため、ユーザー ID とそれに対応する Web サイトを頭のなかでマッピングしているわけですが、面倒なことに、「パスワードを忘れた場合」の機能を盛んに使用するはめになってしまいます。ID を 1 つだけ作成し、その ID をどこででも使える方法があったとしたら言うことありません。

まさにこの問題を解決するのが、OpenID です。私は OpenID を使用して ID を主張し、このプロトコルを採用しているサイトや Web リソースのすべてでこの ID を使用しています。最新の統計によると (OpenID Web サイトでの報告)、Facebook、Yahoo!、Google、Twitterを含め、50,000 を超える Web サイトが OpenID をサポートしています。

OpenID 認証

OpenID 認証は OpenID の要です。この認証は、以下の 3 つの主要なコンポーネントで構成されます。

- OpenID 識別子: このテキスト・ストリングによってユーザーを一意に識別します。
- OpenID RP (リライングパーティー): このオンライン・リソースは、アクセスを許可するユーザーを識別する際に OpenID を使用します (大抵は Web サイトを表しますが、ファイルや画像をはじめ、アクセスを制御する必要のあるほとんどすべてのものがリソースとなり得ます)。
- OP (OpenID プロバイダ): このサイトでは、ユーザーが OpenID を主張し、続いてサインインすることで、ユーザーはあらゆる RP を利用できるようになります。

OpenID Foundation は、OpenID 仕様によるオープンソース ID 管理の促進に関心を寄せるメンバーからなるコンソーシアムです。

OpenID が機能する仕組み

例えば、あるユーザーが RP の Web サイトの一部となっているリソースへのアクセスを試行するとします。この RP は OpenID を使用しているため、ユーザーがリソースにアクセスするには、OpenID であると認識できる (正規化された) 形で自分の OpenID を提示する必要があります。OpenID は OP のロケーションを使用してエンコードされるので、RP がユーザーの ID を受け取ると、該当する OP にユーザーをリダイレクトします。ユーザーはこの OP で、提示した ID の所有者であることを証明することになります。

ここで、OpenID 仕様の各コンポーネントと認証プロセスでのそれぞれの役割を簡単に説明しておきます。

OpenID 識別子

OpenID の中心となるのは当然、OpenID 識別子です。OpenID 識別子 (あるいは単に「識別子」とも言います) は人間が読める文字のストリングで、ユーザーを一意に識別します。あるユーザーの OpenID と別のユーザーの OpenID が同じということは決してありません。これこそが、OpenID が機能する仕組みです。OpenID 認証のバージョン 2.0 の仕様では、OpenID RP はユーザーの認証方法を調べるために ID をデコード (「正規化」) できることが規定されています。OpenID を扱うことのできる世界、つまり私たち開発者がコードを作成する世界では、以下の 2 つの ID が興味の対象となります。

- ユーザ入力識別子
- 主張識別子

その名前からわかるように、ユーザ入力識別子はユーザーが RP に提供する ID です。ユーザ入力識別子は主張識別子に正規化されなければなりません。主張識別子とは凝った呼び方をしているだけで、ユーザーから提供された ID を標準形式に変換した ID にすぎません。この変換後の主張識別子はディスカバリと呼ばれるプロセスで使用されます。このプロセスによって OP のロケーションを見つけた後、OP がユーザーを認証します。

OpenID RP (リライングパーティー)

通常、ユーザ入力識別子は RP に提示され、RP によって主張識別子に正規化されます。RP はユーザーのブラウザー (「User Agent」) を OP にリダイレクトし、ユーザーはその OP で自分のパスワードを提供することによって認証されます。

RP は主張識別子がどのように認証されるかについて、その詳細を知りもしなければ、知ろうともしません。RP が必要とする情報は、OP がユーザーを正常に認証したかどうかだけです。ユーザーが認証されると、User Agent (ここでも大抵はユーザーのブラウザー) は、ユーザーがアクセスしようとしたセキュアなリソースにリダイレクトされます。ユーザーを認証できない場合は、RPがアクセスを拒否します。

OP (OpenID プロバイダ)

OP (OpenID プロバイダ) の役目は、ID を発行すること、そしてユーザーを認証することです。さらに、OP は Web ベースの OpenID 管理も行います。OP はユーザーごとに以下の基本情報を収集して保持します。

- E メール・アドレス
- フルネーム
- 誕生日
- 郵便番号
- 国
- 主要言語

OP は主張識別子の認証要求を受けて、ユーザーのブラウザーをサインイン・ページにリダイレクトします。ユーザーはこのページで、パスワードの入力を求められます。OP に制御が移るのは、この時点です。ユーザーが正常に認証されると、OP はブラウザーを RP が指定したロケーション (特殊な「戻り先」の URL) にリダイレクトします。ユーザーの認証が失敗した場合、ユーザーはおそらく OP から認証試行の失敗メッセージを受け取ることになります (少なくとも、よく使用されている 2 つの OpenID プロバイダである ClaimID と myOpenID はこのメッセージを提供します)。

OpenID RP になる方法

OpenID の主要なコンポーネントと、これらのコンポーネントがどのように連動するかがわかったところで、ここから後は、オープンソースの OpenID4java ライブラリーを使って OpenID RP (リライングパーティー) を作成する手順に焦点を絞ります。

OpenID を使用するための最初のステップは、ID を取得することです。これは簡単なことで、まず myOpenID にアクセスして「SIGN UP FOR AN OPENID」ボタンをクリックします。OpenID には redneckyogi や jstevenperry のようなユーザー ID を選んでください (ちなみに、両方とも私のユーザー ID です)。ユーザー ID を入力すると、サインアップ・フォームにその名前がすでに使用されているかどうかが示されます。使用されていなければ、フォームの指示に従ってパスワードと E メール・アドレス、そして JCaptcha スタイルのテキスト・ボックスに入力します (ロボットでないことを確認するため)。サインアップはこれで完了です。

数分後、フォームに入力したアドレスに E メールが届きます。E メールにはリンクが記載されているので、このリンクをクリックして E メール・アドレスを確認してください。おめでとうございます！これで OpenID を取得できました。

当然のことながら、素晴らしい技術の例に漏れず OpenID プロバイダにも数々の選択肢があります (全プロバイダのリストについては、「参考文献」を参照)。

いかに素早く簡単に OpenID を取得できるかと言うと、私は myOpenID、Verisign、ClaimID のアカウントにサインアップしたところ、約 30 分で ID を取得できました。しかもこの時間には、詳細情報を入力した時間と写真をアップロードした時間も含まれています。

すでに OpenID をお持ちかもしれません

OpenID.net によると、Google や Wordpress の他、人気のあるサイトでは OpenID をサポートしています。これらのサイトのいずれかにサインアップ済みであれば、OpenID をすでに持っている可能性があります。

例えば Yahoo! アカウントを持っている場合、おそらく OpenID も使用できるはずです (知りませんでしたが、私もそうでした)。サインインするときに Yahoo! ID を使用するだけで、Yahoo があなたの OpenID プロバイダになります。つまり、ユーザーが Yahoo ベースの OpenID を xxxx@yahoo.com という形で入力すると、RP が Yahoo にそのユーザーを認証するように求めるというわけです (この動作を実際に確かめるには、この記事に付属のサンプル・アプリケーションを実行してください)。

サンプル・アプリケーションについて

記事の冒頭で述べたように、私はこの記事のために、OpenID4java を使って単純な OpenID RP を作成する Java Web アプリケーションを作成しました。このサンプル・アプリケーション自体も単

純なもので、WAR としてビルドして Tomcat にデプロイし、ローカル・マシンから実行することができます。サンプル・アプリケーションの焦点は、以下のプロセスに絞り込まれています。

- ユーザーが登録ページで OpenID を入力します。
- アプリケーションが (ユーザーを OP にリダイレクトし、そこでサインインさせることによって) ユーザーの ID を検証します。
- 認証に成功すると、アプリケーションは OP からユーザーのプロファイル情報を取得し、ユーザーを「Save (保存)」ページにリダイレクトします。このページでは、ユーザーが自分のプロファイル情報を再度確認し、保存することができます。
- 「Save (保存)」ページに表示する情報は、OP から入手可能な情報のなかから引き出されます。

このアプリケーションは Wicket で作成しました。その理由は単純に、Wicket は私の大のお気に入りだからです。それでも、OpenID RP の作成方法を学ぶことに集中できるよう、Wicket に特有の事柄はできるだけ最小限に抑えてあります。

サンプル・アプリケーションのアーキテクチャーは、その役割によって以下の 2 つの領域に分けられます。

- Wicket で作成されたユーザー・インターフェース
- OpenID 認証 (OpenID4java ライブラリーを使用)

もちろん、この 2 つの分野には重なる部分もありますが、前述のとおり、Wicket の詳細に気をそらされることなく OpenID の手順に従えるよう、重複を最小限にするように努めました。

OpenID4java およびサンプル・アプリケーション・コードについて

OpenID 認証の仕様は複雑です。仕様を年中実装しているとしたら、独自の実装を作成するのはお手の物かもしれませんが、私は怠け者なので、目の前の問題を解決するために必要な最小限の作業しかしたくありません。そこで登場するのが、OpenID4java ライブラリーです。OpenID4java は OpenID 認証の仕様の実装で、プログラムによって OpenID を極めて簡単に使用できるようになっています。

以降のコード・リストに、RP が OpenID を使用するために呼び出す OpenID4java API を記載します。これらのリストを見るとわかると思いますが、OpenID を使用するためにサンプル・アプリケーションに実際に必要となるコードはほんのわずかです。OpenID4java は開発者の作業を実に楽にしてくれます。

サンプル・アプリケーションでは Wicket に特有の事柄を少なくするため、OpenID4java を呼び出すコードは、(`com.makotogroup.sample.model` 内にある) `RegistrationService` という独自の Java クラスに分離しておきました。このクラスには、OpenID4java API の使用方法それぞれに対応する以下の 5 つのメソッドがあります。

- getReturnToUrl()。このメソッドは、認証が成功したときにブラウザーをリダイレクトする宛先 URL を返します。
- getConsumerManager()。このメソッドを使用して、メイン OpenID4java API クラスのインスタンスを取得します。このクラスが、サンプル RP アプリケーションが認証を行うために必要なすべてのコードを処理します。

- performDiscoveryOnUserSuppliedIdentifier()。このメソッドの目的は、その名前に示されているとおり、ディスカバリ・プロセスで発生する可能性のある問題を処理することです。
- createOpenIDAuthRequest()。認証を行うために必要な AuthRequest 構成体を作成するメソッドです。
- processReturn()。このメソッドは、認証要求の結果を処理します。

RP を作成する

認証の本質は、ユーザーがその身元を証明することにあります。それによって、好ましくないユーザーや悪意のあるユーザーが Web リソースにアクセスするのを防ぐためです。ユーザーの身元が証明された上で、そのユーザーにリソースへのアクセスを許可するかどうかが決定されます (リソースへのアクセス許可については、この記事の範囲外です)。

この記事のサンプル・アプリケーションは、多くの Web サイトに共通する機能を実行します。それは、ユーザー登録です。ここでは、ユーザーがその身元を証明できれば、登録も許可されることを前提とします。これは単純な前提ですが、このアプリケーションで説明するのは、OP との典型的なやりとりはどのように行われ、その際どのように OpenID4java が使われるのかについてです。その基本的なステップは以下のとおりです。

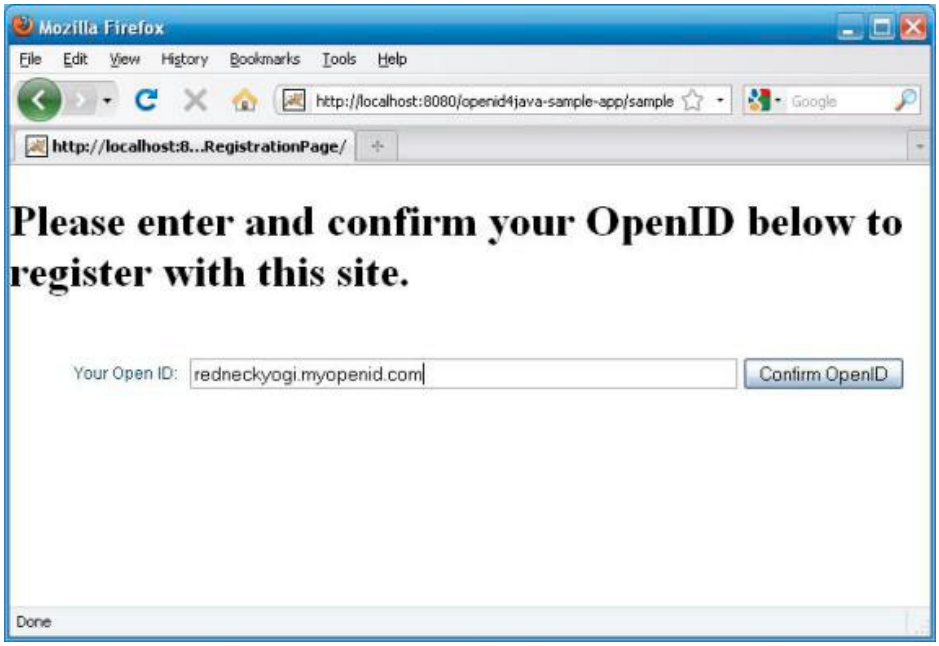
1. **ユーザ入力識別子の取得**: RP がユーザーの OpenID を取得します。
2. **ディスカバリ**: RP はユーザ入力識別子を正規化することによって、認証のために接続する対象となる OP とその OP への接続手段を確定します。
3. **アソシエーション**: これはオプションのステップですが、このステップを実行することを強くお勧めします。このステップによって、RP と OP がセキュアな通信チャネルを確立します。
4. **認証要求**: RP が OP に対し、ユーザーの認証を要求します。
5. **検証**: RP は OP から提供されたユーザー ID の検証を要求することで、通信が改ざんされていないことを確実にします。
6. **アプリケーションの再開**: 認証に続き、RP はユーザーを、そのユーザーが要求したリソースにリダイレクトします。

これから上記のステップのそれぞれを、サンプル・コードと併せて詳しく説明します。以降のセクションでは 1 つの例に沿って OpenID 認証プロセスを最初から最後まで説明します。

ユーザ入力識別子を取得する

ユーザ入力識別子を取得するのは、RP アプリケーションの仕事です。ここで挙げる実例では、サンプル・アプリケーションの `OpenIDRegistrationPage` でユーザー ID を取得します。私がこのページで自分の OpenID を入力し、「Confirm OpenID (OpenID を確認)」ボタンをクリックすると、RP として機能するサンプル・アプリケーションが私のユーザ入力識別子を取得します。図 1 は、実際のサンプル・アプリケーションのスクリーン・ショットです。

図 1. ユーザ入力識別子を取得する



この例でのユーザ入力識別子は、redneckyogi.myOpenID.com です。

UI コードには、2 つの役割があります。1 つはユーザーが「Your OpenID (OpenID)」テキスト・ボックスにテキストを入力したことを確認すること、そしてもう 1 つは、ユーザーが「Confirm OpenID (OpenID を確認)」ボタンをクリックしたら、フォームの実行を依頼することです。フォームの実行が依頼されると、サンプル・アプリケーションは OpenID の入力確認に続き、OpenID 認証の呼び出しシーケンスを開始します。リスト 1 に、フォームの実行が依頼されると、この呼び出しシーケンスを実行する `OpenIDRegistrationPage` のコードを記載します。

リスト 1. RegistrationService.java を使用して OpenID 認証呼び出しシーケンスを実行する Wicket UI コード

```
Button confirmOpenIdButton = new Button("confirmOpenIdButton") {
  public void onSubmit() {
    String userSuppliedIdentifier = formModel.getOpenId();
DiscoveryInformation discoveryInformation =RegistrationService.performDiscoveryOnUserSuppliedIdentifier(
        userSuppliedIdentifier);
    MakotoOpenIdAwareSession session =
     (MakotoOpenIdAwareSession)owningPage.getSession();
    session.setDiscoveryInformation(discoveryInformation, true);
AuthRequest authRequest =
      RegistrationService.createOpenIdAuthRequest(
        discoveryInformation, returnToUrl);
    getRequestCycle().setRedirect(false);
getResponse().redirect(authRequest.getDestinationUrl(true));
    }
};
```

このサンプル・コードが Wicket UI コードにどのように収まっているかについては、あまり気を取られないようにしてください (ただし興味がある場合は遠慮なく、リスト 1 を抜粋した元のコードである `OpenIDRegistrationPage.java` を調べてみてください)。ここで重要な点は、ユーザーがボタンをクリックすると、この UI コードは `RegistrationService` のさまざまなメソッドに委譲して

OpenID4java の API を呼び出し、以下の 3 つの操作を行っていることです (リスト 1 では、それぞれの操作を太字で示してあります)。

1. ユーザ入力識別子でのディスカバリの実行
2. OpenID4java `AuthRequest` オブジェクトの作成 (このオブジェクトは、認証要求を行うために使用されます。)
3. OpenID プロバイダへのブラウザーのリダイレクト

ブラウザーをリダイレクトした時点で UI コードの処理は完了し、制御は OP の手に渡されます。myOpenID.com が ID の一部となっていて、ユーザ入力識別子は整形式の URL ではないことに注目してください。しかしそれでも、この ID には十分な情報がエンコードされているため、OpenID4java は正規化してディスカバリを実行することができます。次のセクションでは、ディスカバリについて説明します。

ディスカバリ

RP はユーザ入力識別子を取得し、どの OP (OpenID プロバイダ) にどういう方法で接続すればよいかを判別するために使用できる形式へと変換します。

ディスカバリのプロセスは、RP が OP への要求を行う方法を判別するために使用します。ここで鍵となるのはユーザ入力識別子ですが、ユーザ入力識別子をディスカバリに使用するにはこれを正規化しなければなりません。まさにこのユーザ入力識別子を正規化するという力仕事を行ってくれるのが、OpenID4java ライブラリーです。したがって、ここで正規化について詳しく説明する必要はありません。

正規化には以下の 2 つの形式があります。

1. XRI: Extensible Resource Identifier
2. URL: Uniform Resource Locator

この記事では、URL の例を取り上げます。図 1 のユーザ入力識別子は、スキームが欠けている URI です。そのため OpenID4java は正規化の一環として「http://」を追加し、http://redneckyogi.myOpenID.com という主張識別子にします。

主張識別子にエンコードされるのは OP の名前です。この例の場合、myOpenID がそれに該当します。主張識別子は URL であるため、OpenID4java はこの OP に接続する手段が http://myOpenID.com であると理解し、この URL にアクセスします。

リスト 2 (サンプル・アプリケーションの `RegistrationService` クラスから抜粋) に、RP が OpenID4java を使用してディスカバリを行う方法を示します。

リスト 2. OpenID4java を使用してディスカバリを実行する

```
public static
   DiscoveryInformation performDiscoveryOnUserSuppliedIdentifier(
      String userSuppliedIdentifier) {

  DiscoveryInformation ret = null;
  ConsumerManager consumerManager = getConsumerManager();
  try {
    // Perform discover on the User-Supplied Identifier
List<DiscoveryInformation> discoveries =
      consumerManager.discover(userSuppliedIdentifier);
    // Pass the discoveries to the associate() method...
    ret = consumerManager.associate(discoveries);
  } catch (DiscoveryException e) {
    String message = "Error occurred during discovery!";
    log.error(message, e);
    throw new RuntimeException(message, e);
  }
  return ret;
}
```

OpenID4java による OpenID 認証方式で中心となるクラスは `ConsumerManager` です。OpenID4java には、このクラスの使用方法に関して厳格なガイドラインがあります。そのため、このクラスは静的クラスのメンバーとして保管し、`getConsumerManager()` メソッドを使用してアクセスします (詳細は、サンプル・アプリケーションに含まれる RegistrationService.java を参照してください)。

OpenID4java はたった 1 行のコード (リスト 2 に太字で示されているコード) で、コードがユーザ入力識別子を正規化してディスカバリを実行できるようにします。ディスカバリによって返されるのは、`DiscoveryInformation` オブジェクトの `java.util.List` です。これらのオブジェクトは opaque オブジェクトとして扱うことができます。ここで注意しなければならないのは、オブジェクトを確実に保持することだけです。これらのオブジェクトは、RP 実装で OP とのアソシエーションを形成するオプションを選択した場合に必要となります (サンプル・アプリケーションではこのオプションを採用しています)。

アソシエーション

アソシエーションとは、RP と OP との間で (Diffie-Hellman 鍵交換方式を使用して) 秘密鍵を共有することで、RP と OP とのやりとりの信頼性とセキュリティーを強化することです。アソシエーションは OpenID 仕様では要件とされていませんが、アソシエーションを行う場合は、RP コードから `ConsumerManager` の `associate()` メソッドを 1 回呼び出します (リスト 3 を参照)。

リスト 3. OpenID4java を使用してアソシエーションを確立する

```
public static
   DiscoveryInformation performDiscoveryOnUserSuppliedIdentifier(
      String userSuppliedIdentifier) {

  DiscoveryInformation ret = null;
  ConsumerManager consumerManager = getConsumerManager();
  try {
    // Perform discover on the User-Supplied Identifier
    List<DiscoveryInformation> discoveries =
      consumerManager.discover(userSuppliedIdentifier);
    // Pass the discoveries to the associate() method...
ret = consumerManager.associate(discoveries);
  } catch (DiscoveryException e) {
    String message = "Error occurred during discovery!";
    log.error(message, e);
    throw new RuntimeException(message, e);
  }
  return ret;
}
```

このメソッドは、ディスカバリの結果を記述する `DiscoveryInformation` オブジェクトを返します (このオブジェクトも opaque オブジェクトとして扱うことができます)。サンプル・アプリケーションでは `DiscoveryInformation` オブジェクトをセッションに保存していますが、それはこの後わかるように、このオブジェクトが後から必要になってくるためです。次のセクションで取り上げる認証要求を行う際にも、このオブジェクトが必要となります。

認証

RP がユーザ入力識別子を使用したディスカバリを正常に完了したら、次はユーザーを認証する番です。認証では、`ConsumerManager` は `AuthRequest` という特殊なオブジェクトが作成するよう求められ、OP はこのオブジェクトを使用して認証要求を処理することになります。

この認証でのやりとりでは、OP は `SimpleRegistration` (略して SReg) という OpenID 拡張を利用するように求められます。この拡張により、RP が OP に対し、応答にユーザー・プロファイルからの特定の属性を返すように要求することが可能になります。`AuthRequest` オブジェクトを作成し、`SReg` を使って属性を要求するコードは、リスト 4 のとおりです。

リスト 4. AuthRequest を作成し、SReg 拡張を使用する

```
public static AuthRequest
createOpenIdAuthRequest(DiscoveryInformation
discoveryInformation, String returnToUrl) {
  AuthRequest ret = null;
  //
  try {
    // Create the AuthRequest object
ret =getConsumerManager().authenticate(discoveryInformation,
                                       returnToUrl);
    // Create the Simple Registration Request
SRegRequest sRegRequest =
SRegRequest.createFetchRequest();
    sRegRequest.addAttribute("email", false);
    sRegRequest.addAttribute("fullname", false);
    sRegRequest.addAttribute("dob", false);
    sRegRequest.addAttribute("postcode", false);
ret.addExtension(sRegRequest);
```

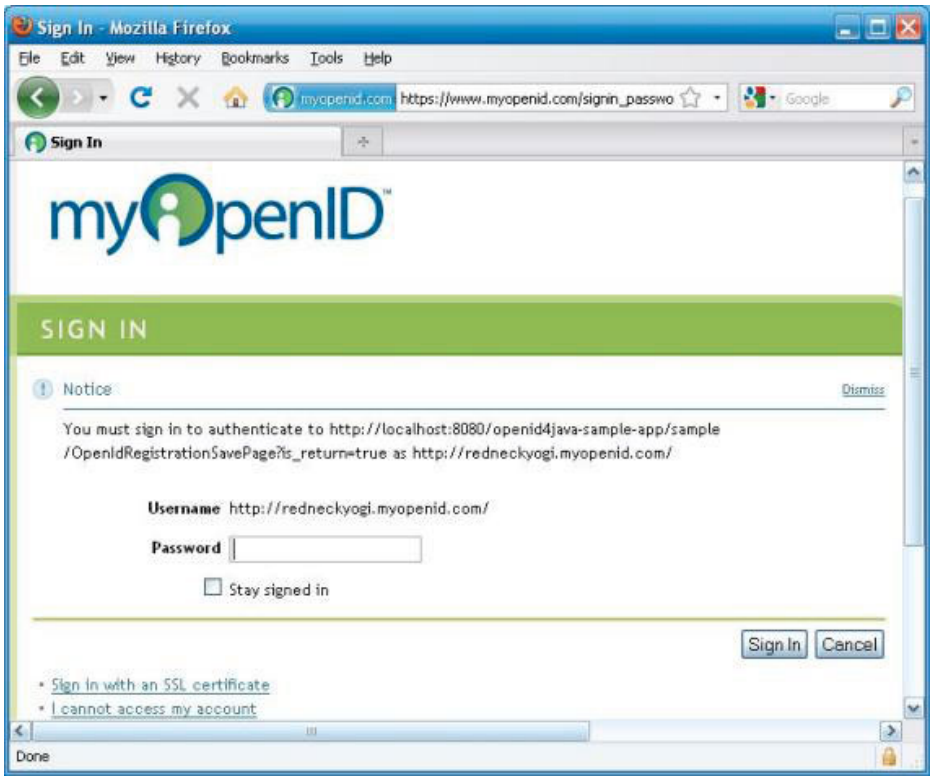
```
  } catch (Exception e) {
    String message = "Exception occurred while building " +
                     "AuthRequest object!";
    log.error(message, e);
    throw new RuntimeException(message, e);
  }
  return ret;
}
```

リスト 4 の太字で示された最初の行では、`ConsumerManager.authenticate()` を呼び出します。この行は実際に認証を呼び出すわけではありません。ディスカバリ成功時の OP とのやりとりによって返された `DiscoveryInformation` オブジェクト (リスト 3 を参照) と、認証が成功した後に User Agent (ブラウザー) をリダイレクトする宛先 URL を取得しているだけです。

2 番目の太字の行では、`SRegRequest.createFetchRequest()` に対する静的メソッド呼び出しによって `SReg` 要求を作成します。続いて `SRegRequest` オブジェクトの `addAttribute()` を呼び出すことによって、OP に対し、必要な属性を Simple Registration Extension の一部として返すように要求します。そして最後の太字の行で `addExtension()` を呼び出し、拡張を `AuthRequest` に追加します。

OpenID4java により、この一連のアクションは極めて直観的に理解できるようになっています。この時点でブラウザーはユーザーを認証する役割を持つ OpenID プロバイダにリダイレクトされ、そこでユーザーが自分のパスワードを入力することになります。このリダイレクトを行う Wicket UI コードについては、`OpenIDRegistrationPage.java` を参照してください。図 2 は、要求を認証する myOpenID サーバーのスクリーン・ショットです。

図 2. 認証要求を処理する myOpenID



この時点で確実にしなければならないのは、要求を処理可能なコードが「リダイレクト先」URL として指定した URL で実行されていることです (リスト 4 を参照)。サンプル・アプリケーションでは、リダイレクト先 URL を `RegistrationService.getReturnToUrl()` にハードコーディングしています。`OpenIDRegistrationSavePage` のコンストラクターは Web 要求を解読し、その要求が OP から返された要求であるかどうかを調べます。OP からの要求であれば、検証が必要です。

検証

リスト 5 は、要求が OP から送られたものかどうかを調べるためのコードです。OP からの要求であれば、`is_return` パラメーターがあり、その値は `true` となっています。その場合、OpenID4java を使用して要求 (実際には OP からの応答) を検証し、リスト 4 で要求した属性を取得します。

リスト 5. リダイレクト先 URL を処理する

```
public OpenIdRegistrationSavePage(PageParameters pageParameters) {
  RegistrationModel registrationModel = new RegistrationModel();
  if (!pageParameters.isEmpty()) {
    String isReturn = pageParameters.getString("is_return");
    if (isReturn.equals("true")) {
      MakotoOpenIdAwareSession session =
        MakotoOpenIdAwareSession)getSession();
DiscoveryInformation discoveryInformation =
        session.getDiscoveryInformation();
      registrationModel =
        RegistrationService.processReturn(discoveryInformation,
          pageParameters,
          RegistrationService.getReturnToUrl());
      if (registrationModel == null) {
          error("Open ID Confirmation Failed.");
        }
      }
    }
    add(new OpenIdRegistrationInformationDisplayForm("form",
        registrationModel));
  }
```

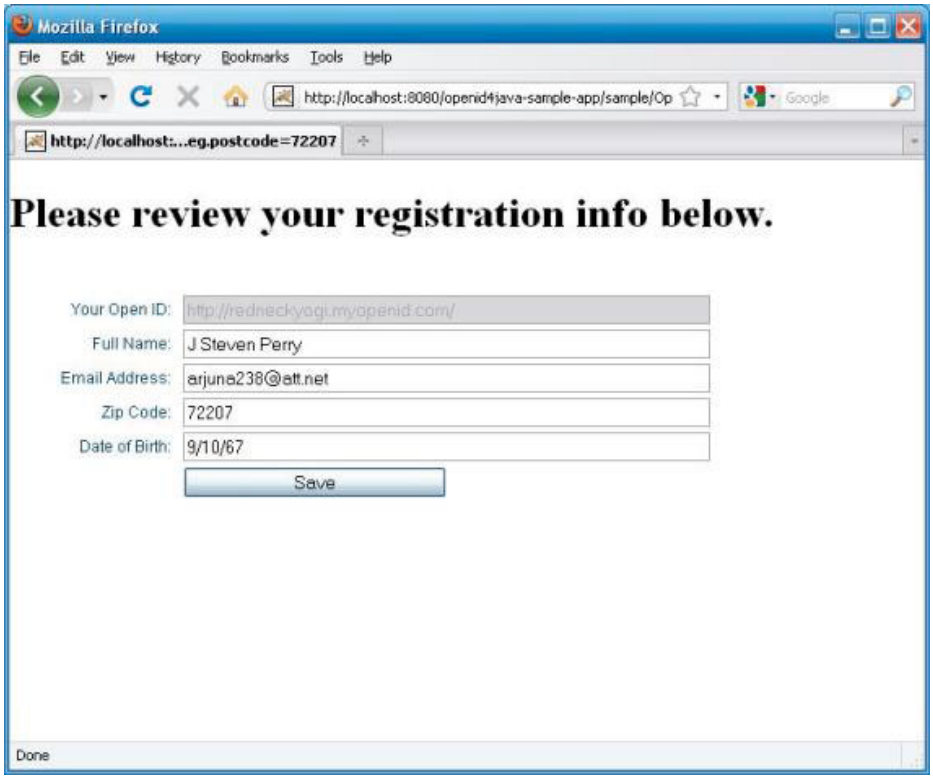
上記のコードでは、最初に Wicket ページのコンストラクターが、その要求が前に行われた認証要求に対する OP からの応答であると判断すると、カスタム `Session` クラス (`MakotoOpenIDAwareSession`) を使用して `DiscoveryInformation` オブジェクトを取得します。このオブジェクトは、OP とのディスカバリ成功時のやりとりの後に保存されたオブジェクトです。`RegistrationService.processReturn()` メソッドは、この `DiscoveryInformation` オブジェクトと要求パラメーター、そしてリダイレクト先 URL を使用して要求を検証します。要求の検証が成功すると、完全にデータが入力された状態の `RegistrationModel` オブジェクトが返されます。このオブジェクトが `OpenIDRegistrationSavePage` の Wicket モデルとして機能することによって、アプリケーションは意図していた機能を再開することができます。

アプリケーションの再開

認証要求に対する応答の検証が正常に完了すると、RP がこれまで OpenID によって保護していたリソースへのアクセス権がユーザーに与えられます。このサンプル・アプリケーションでのリソースは、登録プロセスです。認証済みユーザーには、OP から取得した情報を再度確認し、必要に応じて変更して、保存できる画面が表示されます。サンプル・アプリケーションには実際に登録情報を保存するコードは含まれていませんが、フックはあります。図 3 に示しているのは、私

がサンプル・アプリケーションを実行して自分の OpenID を認証したときに、OP から取得した情報です。

図 3. OP から取得したプロファイル情報を表示するサンプル・アプリケーション



まとめ

インターネット上で使用するいくつもの ID を把握しておかなければならないという面倒を解消する OpenID は、信頼性の高い ID 管理ソリューションとして広く採用されるようになっています。自分用の OpenID を取得するのは簡単で、これまでに取得した人々の数は数百万にのぼります。他のあらゆる仕様と同じく、OpenID 認証の仕様は複雑ですが、これを大幅に単純化するのが OpenID4java です。この記事では、OpenID 認証がどのように機能するかを説明し、OpenID4java を使用することで、OpenID 認証を簡単に Java Web アプリケーションに統合できることを明らかにしました。

連載第 2 回では、OpenID のパズルを仕上げる残りの半分として、OpenID プロバイダを作成する方法に焦点を当てます。今回と同じく、この連載での説明のために作成したサンプル Java Web アプリケーションを使って、コード中心に説明を進める予定です。次回の記事でお目にかかるまでは、Java Web アプリケーションに OpenID 認証を実装するための RegistrationService.java のコードをどうぞ、ご遠慮なくお使いください！

ダウンロード

内容	ファイル名	サイズ
OpenID examples	openid4java-sample-app.zip	4.3 MB

著者について

J Steven Perry

J. Steven Perry は独立したソフトウェア開発コンサルタントで、1991年以来、プロとしてソフトウェアを開発してきました。ソフトウェア開発に情熱を注ぐ彼は、ソフトウェア開発を題材とした執筆活動や、他の開発者たちの指導も楽しんでいます。彼の著作には『Java Management Extensions』(O'Reilly) と『Log4j』(O'Reilly)、そして「Joda-Time」(IBM developerWorks の記事) があります。余暇は、3 人の子供とのひと時、バイク、そしてヨガの指導で過ごしています。彼はアーカンサス州リトルロックにある Makoto Consulting Group のオーナー兼主任コンサルタントです。