Java Web アプリケーションのための OpenID: 第 2 回

シングル・サインオン認証のための OpenID プロバイダを作成する

J Steven Perry
Principal Consultant
Makoto Consulting Group, Inc.

2010年 3月 16日

OpenID を使用して、Java™ Web アプリケーションのリソースを、認証されていないユーザーから保護する方法を学んでください。Steve Perry が OpenID 認証の仕様について紹介する連載の後半では、シングル・サインオンのアプリケーションというシナリオで、openid4java ライブラリーを使用して OpenID プロバイダを作成する方法を説明します。「閉ループ」アーキテクチャーのアプリケーションのうちの 1 つを OpenID プロバイダとして確立すれば、エンド・ユーザーが一度サインインするだけで複数のアプリケーションにアクセスできるようになります。さらに、OpenID リライングパーティーとプロバイダの間でのカスタム・データの交換を可能にする OpenID AX (Attribute Exchange) 拡張の使い方についても学んでください。

このシリーズの他の記事を見る

エンド・ユーザーが 1 つの共通して認識されるユーザー ID を使って複数の Web サイトや他のオンライン・リソースにアクセスできるようにする OpenID は、信頼性の高い ID 管理兼認証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。連載第 1 回では OpenID 認証の仕様について紹介し、openid4java ライブラリー実装を使用して OpenID を Java Web アプリケーションに統合する方法を説明しました。

第 1 回ではその大半を、OpenID リライングパーティー (RP) に重点を置いて説明を進めました。RP とは、OpenID を利用して登録および認証を行う Web サイトや MP3 などのオンライン・リソースのことです。この RP と並んで OpenID 認証の仕様を構成するのが、OpenID プロバイダ (OP) です。OP はユーザーが OpenID を主張できるよう支援し、OpenID 対応の Web リソースにログインしようとするユーザーを認証します。

第 1 回で取り上げた myOpenID (Java Web アプリケーション登録システムの OP) をはじめ、すでに多数の OpenID プロバイダが存在することから、OpenID プロバイダをわざわざ一から作成しなければならない場合はほとんどありません。

一方、独自の OpenID プロバイダを作成することに意味があるケースは、信頼できるネットワーク内で複数のアプリケーションがリソースを共有するようなアプリケーション・クラスターがある場合です。この場合、セキュアな「閉ループ」システムを作成するとよいかもしれませんが、

ユーザーが個々のアプリケーションにいちいちサインインするのではなく、一度にすべてのアプリケーションにサインインできるようにすると便利です。それにはクラスター内の 1 つのアプリケーションに OP としての役割を持たせることで、すべてのアプリケーションを対象としたシングル・サインオン認証をセットアップすることができます。

今回の記事で焦点とするのは、1 つの閉ループ・アーキテクチャーで多数のアプリケーションを保護するための OpenID プロバイダを作成する方法です。まずはシングル・サインオン認証の利点と構造を詳しく見ていき、その後クラスター・アーキテクチャーに対応した単純な OpenID プロバイダをコーディングする手順に移ります。この手順でも、前回と同じく openid4java ライブラリーを使用して、認証システムのコアとなるランタイム機能を提供するとともに、OpenID 認証の仕様に確実に準拠する OpenID プロバイダを作成します。

シングル・サインオン認証

企業のシナリオによっては、すべての機能を単一のアプリケーションに組み込もうとするよりも、機能やコア・コンピテンシーの異なる複数のアプリケーションを組み合わせたほうが理にかなっているという場合があります。このようなアプリケーション・クラスターが B2B の中核となることはよくあります。その場合、それぞれのビジネス・パートナーが機能を持ち寄ることによって、全体としての事業案の価値が向上することになります。

アプリケーション・クラスターを開発する上で難題となるのは、認証です。アプリケーションのそれぞれにエンド・ユーザーを認証させるのでは、少なくともエンド・ユーザーの観点からは、アプリケーション・クラスターは有効に機能しているとは言えません。

認証に OpenID 標準を利用するクラスター・システムでは、各パートナー・アプリケーションが OP に認証を委任します。この場合、それぞれのアプリケーションの機能とリソースへのアクセスが安全であることが保証され、エンド・ユーザーにはセッションごとに一度だけサインインするだけで済むという利便性が与えられます。

以下では、シングル・サインオン認証システムに関与するコンポーネントのそれぞれについて詳しく説明します。以下に説明するアーキテクチャーは、第 1 回で開発したサンプル・アプリケーションをベースにしていることに注意してください。

Open ID リライングパーティー (RP)

前にも説明したように、Open ID リライングパーティーは Web サイトやその他のオンライン・リソースであり、そのコンテンツにはセキュアなアクセスが必要です。RP は OpenID プロバイダ (OP) を利用してユーザーを認証します。さらに、ユーザーの登録やユーザーに関する情報の識別を要求するために、RP が SReg (Simple Registration) 拡張や AX (Attribute Exchange) 拡張 (「参考文献」を参照) を使用する場合もあります。RP は、openid4java ライブラリーの呼び出しによって OP にユーザーの認証を依頼する際に、SReg 要求および AX 要求を行います。

SReg についての詳細

第 1 回のサンプル・アプリケーションでは、RP が OP からユーザー情報を入手できるようにするために OpenID SReg (Simple Registration) 拡張を使用しました。OpenID の SReg 拡張についての詳細は、「Java Web アプリケーションのための OpenID: 第 1 回」を参照してくださ

い。今回の記事に使用するアプリケーションは、より複雑な情報トランザクションを可能にする OpenID AX (Attribute Exchange) 拡張を利用します。

Open ID プロバイダ (OP)

OpenID プロバイダは、すべてのパートナー・アプリケーションに代わって認証を行います。openid4java ライブラリーの呼び出しによるユーザーの認証が成功すると、OP は RP からの SReg および AX 要求を満たします。OP は、この記事で詳しく調べるシングル・サインオン・アーキテクチャーの中心的役割を果たします。

OpenID プロバイダの作成方法

前回の記事では、openid4java を使用して Java Web アプリケーション登録システムのリライングパーティーを作成する手順を説明しました。今回は OpenID プロバイダを対象に、同様の手順に従います。openid4java には OpenID インフラストラクチャーのすべてがあらかじめコーディングされているため、openid4java を使用することで、容易に OpenID 認証の仕様に準拠することができます。

サンプル・アプリケーションについて

このサンプル・アプリケーションの目的は、OpenID RP と OpenID OP が連動して、許可されていないアクセスからリソースを保護する仕組みを説明することです。そのため、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。

1. ユーザーが保護されたリソースにアクセスしようとします。
2. RP が OP にユーザーの認証を依頼します。
3. OP がユーザーを認証します (ユーザーがまだログインしていない場合)。
4. RP が、保護されたリソースへのアクセスがログイン・ユーザーに許可されているかどうかを判断します。

サンプル・アプリケーションには RP と OP 両方のコードが組み込まれているので、この 2 つが連動する仕組みを調べることができます。実際のシナリオでは、2 つのコンポーネントを同じアプリケーションにデプロイすることはありませんが (そんなことをしても無意味です！)、この 2 つを合わせて確認し、どのようなやりとりが行われるかを調べると参考になるはずです。

サンプル・アプリケーションのコード・リスト

以降のセクションに記載するコード・リストには、OP (および RP) が OpenID を使用するために行う openid4java API 呼び出しが示されています。これらのリストを見るとわかると思いますが、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少ししかありません。このように、openid4java は開発者の作業を実に楽にしてくれます。RP が使用するコードは第 1 回に記載したコードと基本的に同じなので、RP の内部構造についての詳細は第 1 回を参照してください。多少の違い (そのほとんどは、第 1 回では説明しなかった AX 関連の違いです) については、手順の中で指摘します。

最初の記事のために作成したアプリケーションと同じく、このサンプル・アプリケーションでも UI には Wicket を使用します。サンプル・アプリケーションでは Wicket のフットプリントを小さくするため、OP が openid4java を呼び出すために使用するコードは、

(com.makotogroup.sample.model 内にある) OpenIdProviderService という独自の Java クラスに分離しておきました。

OpenIdProviderService.java には、openid4java API の使用方法のそれぞれに対応する以下のメソッドがあります。

- getServerManager(): openid4java の `ServerManager` クラスへの参照を構成して返すメソッドです。
- getOpEndpointUrl(): このメソッドは、OP が RP からの要求を受け取るエンドポイント URL を返します。
- processAssociationRequest(): このメソッドは openid4java を使用して、RP の要求時に OP を関連付けます。
- sendDiscoveryResponse(): ディスカバリ要求を RP に送信するメソッドです。
- createAuthResponse(): このメソッドは、認証要求の後に RP に送信される penid4java の `AuthResponse` メッセージを作成します。
- buildAuthResponse(): OpenID Simple Registration および Attribute Exchange 要求を処理するコア・メソッドです。

サンプル・アプリケーションを起動するには、`Ant [REF]` を実行して WAR ターゲットを作成し、これを Tomcat の webapps ディレクトリーにコピーしてから Tomcat を起動してください。

OpenID 認証: ステップ・バイ・ステップ

ユーザーがリライングパーティー (RP) から保護されたリソースにアクセスしようとすると、RP はそのユーザーがアクセスしようとしている本人であることを確認した上で (認証)、そのユーザーにアクセスを許可するかどうかを決定します (許可)。この記事で焦点とするのは認証なので、ユーザーが OpenID プロバイダ (OP) によって認証されると、サンプル・アプリケーションは保護されたリソースへのアクセスを許可するという設定にします。実際のシナリオでは、RP も何らかの認証を行います。

サンプル・アプリケーションを実行すると、保護されたリソースが記載された画面が表示されます。その後に発生する以下のイベントのシーケンスについては、次のセクションで詳しく説明します。

1. **保護されたリソースへのアクセス要求**: RP の Web サイトで、保護されたリソースへのアクセスをユーザーが試みます。
2. **RP によるディスカバリの実行**: RP は接続を確立し、さらにアソシエーションを確立するために、OP にディスカバリ要求を送信します。
3. **OP によるディスカバリ要求への応答**: OP がディスカバリ要求に正しく応答するには、SReg、AX (Attribute Exchange)、または OpenID Provider AP (Authentication Policy) (この記事では説明しませんので、「参考文献」を参照) のいずれかの拡張を使用して XRDS (eXtensible Resource Descriptor Sequence) を RP に送信する必要があります。XRDS により、OP がユーザーの OpenID サービス・プロバイダであることが確認されます。
4. **RP によるユーザー認証の要求**: RP は OP に確認し、ユーザーを認証できるかどうかを調べます。ログインが成功すると、RP は SReg または AX、あるいはこの両方を使用して、ユーザーに関する特定の情報を要求します。

5. OP によるユーザーの認証: ユーザーがログインしていない場合、または無効なセッションである場合には、ユーザーに対し、ログイン資格情報を入力するよう求めます。認証が成功すると、OP は RP に認証成功の通知と併せ、SReg や AX によって要求されたデータを送信します。
6. RP によるアクセスの許可: ユーザーに、保護されたリソースへのアクセスを許可。ただし実際のシナリオでは、ほとんどの RP はアクセスを許可する前に、ユーザーに与えられているアクセス権を確認します。

ここからは、上記のステップごとに詳しく説明していきます。

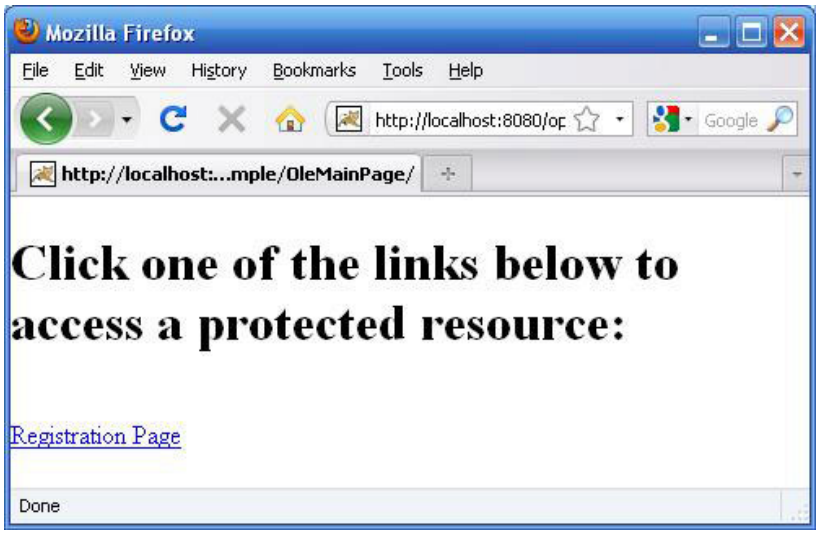
AX 拡張を使用する理由とは

お気付きだと思いますが、サンプル・アプリケーションは OpenID のSReg 拡張と AX 拡張 (「参考文献」を参照) の両方を使用して、OP と RP との間でユーザー情報を受け渡します。基本的に、この 2 つの拡張は OP と RP との間での効率的な通信を可能にするためのものです。SReg では交換できる属性が限られていますが、AX を使用すると、RP と OP の両方でその情報が属性として定義されている限り、事実上あらゆる情報を交換することができます。この記事で話題にしているようなクラスターのシナリオでは、信頼できるアプリケーション (RP) ごとに独自のカスタム「ベンダー拡張」が定義されている場合もあります。それもまた、OP と RP との間の通信を効率化する方法です。AX 拡張については、記事の後のほうで詳しく説明します。

保護されたリソースへのアクセス要求

サンプル・アプリケーションに含まれる保護されたリソースは 1 つだけです。アプリケーションを起動して RP の URL である http://localhost:8080/openid-provider-sample-app/ にアクセスすると、以下の図に示すページがロードされます。

図 1. サンプル・アプリケーションのメイン・ページ



ユーザーがページ上のリンクをクリックすると、リスト 1 のコードが実行されます。

リスト 1. 保護されたリソースを表示するアプリケーションのメイン・ページ

```
package com.makotogroup.sample.wicket;
. . .
public class OleMainPage extends WebPage {
  public OleMainPage() {
    add(new OleMainForm("form"));
  }
  public class OleMainForm extends Form {
    public OleMainForm(String id) {
      super(id);
      add(new PageLink("openIdRegistrationPage", new IPageLink() {
        public Page getPage() {
          return new OpenIdRegistrationPage();
        }
        public Class<? extends WebPage> getPageIdentity() {
          return OpenIdRegistrationPage.class;
        }
      }));
    }
  }
}
```

リスト 1 の太字で示されたコードに注目してください。ユーザーが図 1 に示されたリンクをクリックすると、Wicket はユーザーを `OpenIdRegistrationPage` (リソース) にリダイレクトします。すると、リンクの宛先が呼び出され、`OpenIdRegistrationPage` クラスのコンストラクターが実行されます。このクラスでは、以下の 2 つの動作を行います。

- 初期呼び出しのエントリー・ポイントとしての動作
- 認証成功に続いて OP が行う「コールバック」動作

このページにアクセスするための初期呼び出しでは、`Wicket` の `PageParameters` が渡されません。これにより、RP はユーザーを認証するためには OP に確認しなければならないと認識するというわけです。

RP によるディスカバリの実行

RP と OP が通信するには、RP が OP でディスカバリを実行する必要があります。OP でのディスカバリはコーディングの観点からすると単純ですが (これもまた、openid4java のおかげです)、これは重要なステップであることから、ここで特に取り上げることにしました。

以下に示すのは、RP がディスカバリ要求を送信するために (`OpenIdRegistrationPage` のコンストラクターから) 使用するコードです。

```
  DiscoveryInformation discoveryInformation =
    RegistrationService.performDiscoveryOnUserSuppliedIdentifier(
          OpenIdProviderService.getOpEndpointUrl());
```

このコードで、RP は以下の 2 つの操作を行います。

1. OP のエンドポイント URL でディスカバリを実行します。
2. RP 自体と OP とのアソシエーションを確立します (Diffie-Hellman 鍵交換方式の詳細と、アソシエーション中に行われる巧妙な操作については、第 1 回を参照してください)。

この後、RP のディスカバリ要求を処理するかどうかは OP に一任されます。

OP によるディスカバリ要求への応答

openid4java はサンプル・アプリケーションの RP 側と OP 側の両方で実行されていることを思い出してください。OP でのディスカバリ・プロセスの一環として、RP 側の openid4java は空の要求を OP のエンドポイント URL に送信します。このエンドポイント URL は、OP に接続するための URL となります。つまり OP は、ここで RP からのすべての要求を受け取ります。OP はこの要求を処理するようにセットアップされていなければなりません。`OpenIdProviderService.getOpEndpointUrl()` の内容を見てみると、エンドポイント URL は http://localhost:8080/openid-provider-sample-app/sample/OpenIdLoginPage となっていることがわかります。

RP が空の要求を OP に送信すると、Wicket は `OpenIdLoginPage` を組み立て、そのコンストラクターを実行します (リスト 2 を参照)。

リスト 2. OP エントリー・ポイント

```
 public OpenIdLoginPage(PageParameters parameters) throws IOException {
    super(parameters);
    if (parameters.isEmpty()) {
      // Empty request. Assume discovery request...
OpenIdProviderService.sendDiscoveryResponse (getResponse());
  . . .
```

OP は空の要求を受け取ると、それがディスカバリ要求であると見なし、XRDS 文書を作成して要求側に送信します。

リスト 3 に、`sendDiscoveryRequest()` のコードを記載します。

リスト 3. ディスカバリ要求に対する応答を送信する

```
  public static void sendDiscoveryResponse (Response response) throws IOException {
    //
    response.setContentType("application/xrds+xml");
    OutputStream outputStream = response.getOutputStream();
    String xrdsResponse = OpenIdProviderService.createXrdsResponse();
    //
    outputStream.write(xrdsResponse.getBytes());
    outputStream.close();
  }
```

XRDS 文書は、RP 側の openid4java が正しく機能するために不可欠です。簡潔にするためにこの記事では XRDS 文書の詳細を省きます。詳細については**サンプル・アプリケーションのソース・コード**をダウンロードして参照してください。

RP は OP から XRDS 文書を受け取ることで、当該ユーザーにとって適切な OP に接続していることを認識します。この認識を得た上で、RP は認証要求を作成し、OP に送信します。

RP によるユーザー認証の要求

RP はユーザーを認証できるかどうかを調べるため、OP に確認します。リスト 4 に、そのための (コンストラクターからの) 一連の呼び出しを記載します。

リスト 4. OP に認証を委任するための RP のコード

```
DiscoveryInformation discoveryInformation =
  RegistrationService.performDiscoveryOnUserSuppliedIdentifier(
        OpenIdProviderService.getOpEndpointUrl());
MakotoOpenIdAwareSession session =
  (MakotoOpenIdAwareSession)getSession();
session.setDiscoveryInformation(discoveryInformation, true);
AuthRequest authRequest =
  RegistrationService.createOpenIdAuthRequest(
        discoveryInformation,
        RegistrationService.getReturnToUrl());
getRequestCycle().setRedirect(false);
getResponse().redirect(authRequest.getDestinationUrl(true));
```

まず初めに、RP は OP のエンドポイント URL で OP に接続します。この呼び出しは奇異に見えるかもしれませんが、このシナリオで焦点としているのは、信頼できるパートナーを OP として使用したアプリケーションのクラスターであることを思い出してください。RP の観点からすると、ユーザーが提供した ID を認証するということは、OP の所在を見つけて、openid4java に以降の対話を円滑にするために必要なオブジェクトを作成させるだけのことです。実質的な認証は、OP が処理します。

次に現行の Wicket の `Session` が取得されて、そこに、openid4java から取得された `DiscoveryInformation` を後で使用できるように保管します。ここでは、`Session` に openid4java オブジェクトを保管しやすくするために、`MakotoOpenIdAwareSession` という特殊な `Session` サブクラスを作成しました。

その後、openid4java から取得された `DiscoveryInformation` オブジェクトを使用して認証要求を作成します。このオブジェクトは、認証の呼び出しを行うためのリダイレクト先を Wicket に通知するために使用されます。

上記のステップはすべて、第 1 回で説明した内容と同様ですが、今回これらのステップを再度説明している理由は、この記事で使用するサンプル・アプリケーション・コードのアーキテクチャーが第 1 回のコードとはかなり異なるためです。また、OP 側での API 呼び出しを記載することで、皆さんに全体像を把握してもらいたいという目的もあります。

この時点で、RP は OP からの認証に対する応答を待つことになります。次のステップに進む前に、ユーザーを認証する上で Attribute Exchange が果たす役割を詳しく説明しておきます。

OpenID Attribute Exchange 拡張

この記事の前半で簡単に説明したように、SReg (Simple Registration) 拡張は、RP と OP の間で (SReg 仕様で定義された) 特定の情報一式を交換することを可能にします。この記事のサンプル・アプリケーションで `createOpenIdAuthRequest()` メソッドを調べると、RP は OP から情報を取得するために別の拡張を使用していることがわかります。それが、OpenID AX (Attribute Exchange) です。

SReg 拡張と同じく、OpenID AX (Attribute Exchange) は RP と OP との間で一貫した標準的な方法で情報を交換するために使用されます。しかし SReg とは異なり、AX では RP と OP の両方が AX 拡張をサポートする限り、OpenID リライングパーティーとプロバイダとが交換する情報に制限を設けません。

簡単に言えば、RP がメッセージを使って OP に特定の情報を要求すると、OP がその情報をメッセージに含めて送信するという仕組みです。これらのメッセージは、ブラウザーのリダイレクト先となる URL にエンコードされますが、openid4java はオブジェクトを使用して、メッセージ内の情報を使用できるようにします。

RP が AX 要求を行うために使用するクラスは、`FetchRequest` です。RP はこのメッセージ・オブジェクトへの参照を作成した後、OP に返させる属性を追加します (リスト 5 を参照)。

リスト 5. 属性を追加した RP の FetchRequest

```
AuthRequest ret = obtainSomehow();
// Create AX request to get favorite color
FetchRequest fetchRequest = FetchRequest.createFetchRequest();
fetchRequest.addAttribute("favoriteColor",
       "http://makotogroup.com/schema/1.0/favoriteColor",
        false);
ret.addExtension(fetchRequest);
```

OP が要求された情報を RP に送信するときに使用する構造体は、RP が AX 要求を行うために使用する構造体と同じです (リスト 6 を参照)。

リスト 6. 要求された属性を送信する OP

```
if (authRequest.hasExtension(AxMessage.OPENID_NS_AX)) {
  MessageExtension extensionRequestObject =
     authRequest.getExtension(AxMessage.OPENID_NS_AX);
  FetchResponse fetchResponse = null;
Map<String, String> axData = new HashMap<String, String>();
  if (extensionRequestObject instanceof FetchRequest) {
FetchRequest axRequest = (FetchRequest)extensionRequestObject;
    ParameterList parameters = axRequest.getParameters();
    fetchResponse = FetchResponse.createFetchResponse(
        axRequest, axData);
    if (parameters.hasParameter("type.favoriteColor")) {
axData.put("favoriteColor", registrationModel.getFavoriteColor());
      fetchResponse.addAttribute("favoriteColor",
          "http://makotogroup.com/schema/1.0/favoriteColor",
          registrationModel.getFavoriteColor());
    }
authResponse.addExtension(fetchResponse);
  } else {
    // ERROR
  }
}
```

定義された属性のそれぞれに、単純な名前とそれに関連付けられた URI が設定されます。上記の場合、属性の単純な名前は `FavoriteColor` で、その URI は http://makotogroup.com/schema/1.0/favoriteColor. です。

さらに、属性はストリングにすることが可能でなければなりません (日付フィールドをストリングで送信する例については、サンプル・アプリケーションを参照してください)。RP と OP の間で交換する属性を定義する際には、この両方が交換対象の属性に同意してなければなりませんが、それ以降は交換する情報に何の制限もありません。

それでは、先ほどアプリケーションのやりとりで中断したところから説明を続けます。

OP によるユーザーの認証

認証要求が OP のエンドポイント URL に到達したところに話を戻すと、OP が認証要求を受け取ると、その要求を解読して次に行うべき処理を判断します。OP が要求を開いて取得する mode は、アソシエーションであることも、認証であることもあります。

リスト 7. OP によるアソシエーション要求の処理

```
//From (OpenIdLoginPage's constructor):

public OpenIdLoginPage(PageParameters parameters) throws IOException {
    super(parameters);
    . . .
    if ("associate".equals(mode)) {
        OpenIdProviderService.processAssociationRequest(getResponse(), requestParameters);
      }
    . . .
}

//From (OpenIdProviderService):

  public static void processAssociationRequest(Response response, ParameterList request)
       throws IOException {
    Message message = getServerManager().associationResponse(request);
    sendPlainTextResponse(response, message);
  }
  private static void sendPlainTextResponse(Response response, Message message)
       throws IOException {
    response.setContentType("text/plain");
    OutputStream os = response.getOutputStream();
    os.write(message.keyValueFormEncoding().getBytes());
    os.close();
  }
```

リスト 7 では、`OpenIdLoginPage` のコンストラクター (サンプル・アプリケーションでの OP のエントリー・ポイント) が最初に要求を解読します。このリストの mode はアソシエーション要求を示しているため、コンストラクターはアソシエーションの具体的な処理を openid4java に委任します。そのコードは、OpenIdProviderService.java にラップされています。その後、アソシエーション要求は RP に送り返されます。

アソシエーションが確立されたと RP が判断すると (実際には openid4java が判断すると)、RP は OP に対して別の呼び出しを行います。この場合も OP が要求を解読して、その要求を処理します。大抵、この要求は `checkid_authentication` 要求となります。

リスト 8 に、`OpenIdLoginPage` のコンストラクターのコードを記載します。

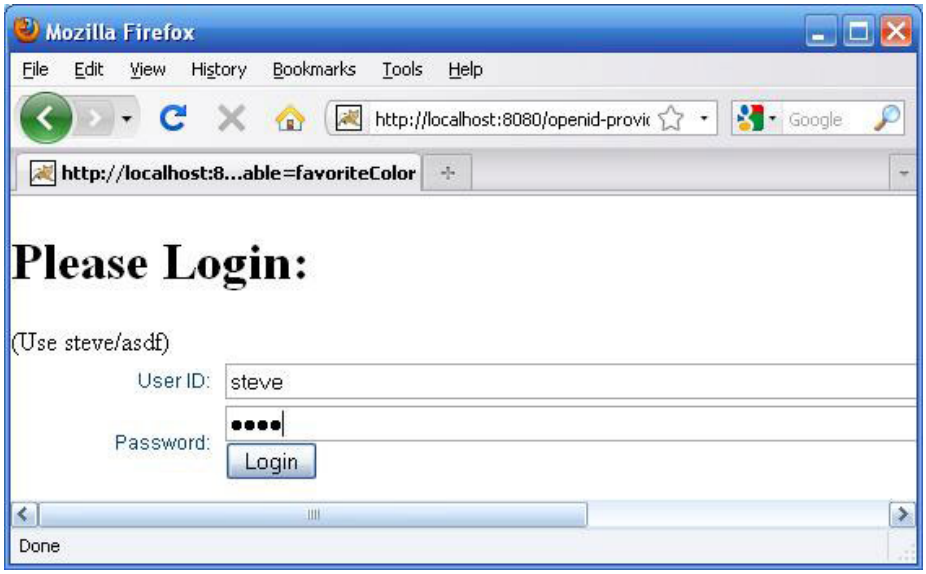
リスト 8. openid4java による checkid_authentication 要求の解読

```
  public OpenIdLoginPage(PageParameters parameters) throws IOException {
    super(parameters);
     . . .
      else if ("checkid_immediate".equals(mode)
        ||
         "checkid_setup".equals(mode)
        ||
         "check_authentication".equals(mode)) {
        if (((MakotoOpenIdAwareSession)getSession()).isLoggedIn()) {
          // Create AuthResponse from session variables...
sendSuccessfulResponse();
        }
    add(new OpenIdLoginForm("form"));
    . .
  }
```

リスト 8 の太字で示した 2 つの行に注意してください。最初の太字の行では、OpenIdLoginPageが要求成功の応答を送信します。`OpenIdLoginPage` はまず `Session` オブジェクトを使用して、ユーザーがログインしているかどうかを判断します (ログイン・ユーザーは再度ログインする必要がないためです)。ユーザーがログインしていれば、認証成功のメッセージを返します。これが、サンプル・アプリケーションでシングル・サインオンを行う仕組みです。

ユーザーがログインしていない場合、Wicket はユーザーが資格情報を入力できるようにログイン・フォームを作成します (図 2 を参照)。

図 2. 認証されていないユーザーに対して OP が表示するログイン画面



ユーザーの認証が成功すると、ユーザーには認証成功の応答が送信され、ある特定の情報 (具体的には、`DiscoveryInformation` オブジェクト) が `Session` に保管されます。これらの要素もまた、シングル・サインオン認証の基礎となる構造です。

この時点で、ブラウザーは認証成功の応答と一緒に RP の「return-to」に記されている URL にリダイレクトされます

RP によるアクセスの許可

ユーザーが正常にログインすると、OP は認証成功を示す `AuthResponse` メッセージで応答します。ここからは、ユーザーにアクセスを許可するかどうかは RP 次第です。サンプル・アプリケーションでは、OP がユーザーを認証した場合には、そのユーザーに自動的にアクセスを許可するようになっています。加えて、OP は RP が要求したユーザー情報をすべて送信します。図 3 は、RP が登録画面に表示するユーザー情報です。

図 3. OP から取得した情報を表示するサンプル・アプリケーションの登録ページ

まとめ

この記事では、OpenID 認証の仕様に従って、パートナー・アプリケーションからなるクラスターを対象としたシングル・サインオンのユーザー認証をセットアップする方法を説明しました。シングル・サインオン・アーキテクチャーは、パートナー・アプリケーション間で信頼が確立されていて、そのうち 1 つのアプリケーションが OpenID プロバイダ (OP) として機能する場合に有効です。

認証およびデータ交換に OpenID を使用することによって、すべての関係者が確実に認証および許可の条件に同意することになります。OpenID は広く採用されている標準なので、新しく採用するという場合でも、OpenID 認証の実装に関する学習、そしてトラブルシューティングについては、業界が大々的にサポートしてくれます。

この記事で実装したシングル・サインオン・アーキテクチャーについての詳細を調べるには、サンプル・コードを参照してください。このサンプル・アプリケーションは、WAR を作成して Tomcat にドロップするだけで実行することができます。TRACE ロギング機能を有効にして、ロ

グ出力を観察すると、この記事では説明していないアプリケーションの詳細が見えてくるはずです。

他のあらゆる仕様と同じく、OpenID Authentication は複雑な仕様ですが、openid4java によってとても簡単に使用できるようになります。どうぞ遠慮なく、この記事のソース・コードをダウンロードして、OpenID によるアプリケーションの認証に役立ててください。

ダウンロード

内容	ファイル名	サイズ
Source code for the sample application	openid-provider-sample-app.zip	4.5KB

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。