Seam Social Module

Reference Guide

3.1.0-SNAPSHOT

by Antoine Sabot-Durand

Introduction

Seam Social Provides CDI Beans and extensions for interacting with a number of major social networks. Currently it provides these services and eases mixing them:

- A generic portable REST client API
- A generic API to deal with OAuth 1.0a and 2.0 services
- A generic API to work with JSON serialization and de-serialization
- A generic identification API to retrieve basic user information from a Social Service
- Specific APIs for interacting with Twitter, Facebook and LinkedIn Services
- A multiservices manager API allowing to deal with multiple OAuth applications and sessions in the same application
- An easy way to extend it by creating a new module for a new services

Seam Social is independent of CDI implementation and fully portable between Java EE 6 and Servlet environments enhanced with CDI. It can be also used with CDI in Java SE (desktop application).

Getting Started

If you are using Maven, you need to add the Seam Social core libraries to your `pom.xml`:

```
    <dependency>
        <groupId>org.jboss.seam.social</groupId>
        <artifactId>seam-social-api</artifactId>
        <version> ${seam.social.version}</version>
        <scope>compile</scope>
    </dependency>

    <dependency>
        <groupId>org.jboss.seam.social</groupId>
        <artifactId>seam-social</artifactId>
        <version> ${seam.social.version}</version>
        <scope>runtime</scope>
    </dependency>
```

```
<dependency>
    <groupId>org.jboss.seam.social</groupId>
    <artifactId>seam-social-twitter</artifactId>
    <version> ${seam.social.version}</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.jboss.seam.social</groupId>
    <artifactId>seam-social-facebook</artifactId>
    <version> ${seam.social.version}</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.jboss.seam.social</groupId>
    <artifactId>seam-social-linkedin</artifactId>
    <version> ${seam.social.version}</version>
    <scope>runtime</scope>
</dependency>
```

Seam Social in 5 minutes

The Web example app is quite simple and gives a good idea of the possibilities with Seam Social. The main steps you need to take to use Seam Social are:

- Declare an OAuth configuration
- Inject an `OAuthService` bean
- Request the Authorization URL for the service and get a request token
- Store the verifier in the `OAuthService` bean and initialize the access token
- Use the service

Should you need to fully understand each step, the complete OAuth lifecycle can be found *here* [https://dev.twitter.com/docs/auth/oauth] or *here* [https://developer.linkedin.com/documents/authentication].

2.1. Declaring an OAuth Configuration

To consume an OAuth service you need to declare an application on the service platform (i.e. for Twitter you can do this at *https://dev.twitter.com/apps/new*. The declaration of an application is done with the `@OAuthApplication` annotation which must contain at least:

- An OAuth API public key
- An OAuth API private/secret key

If you don't know what this is about, please refer to the OAuth concepts in your service documentation.

To use an OAuth Service Bean in Seam Social you need to provide the following configuration information by producing the right `OAuthService` bean:

- Via a producer method or subclassing
- Via an XML configuration (Using Solder's bean configuration feature).

2.2. Configuration with a producer method or a bean definition

The simplest way to configure your service is to create a producer method like so:

```
        @OAuthApplication(apiKey = "FQzlQC49UhvbMZoxUIvHTQ", apiSecret =
"VQ5CZHG4qUoAkUUmckPn4iN4yyjBKcORTW0wnok4r1k")
   @Twitter
   @Produces
```

```
    TwitterService twitterServiceProducer(TwitterService ts) {
        return ts;
    }
```

You can also create a bean by subclassing the implementation of the service like this:

```
        @OAuthApplication(apiKey = "FQzlQC49UhvbMZoxUIvHTQ", apiSecret =
 "VQ5CZHG4qUoAkUUmckPn4iN4yyjBKcORTW0wnok4r1k")
    @Twitter
    public class MyTwitterBean extends TwitterServiceJackson {

…
}
```

The API key and API secret are provided by the service you want to consume (here Twitter). You can use the values above since they're coming from the "Seam Social" Twitter application. The callback depends on your application - it's the URL that will collect the OAuth verifier.

2.3. Injecting the Service Bean into your code

You can now inject the bean with the right service qualifier:

```
@Named
@SessionScoped
public class mySessionBean implements Serializable {
    ...
    @Inject
    @Twitter
    TwitterService service;
    ...
}
```

2.4. Request the OAuth authorization URL

You can now ask for the authorization URL for your service:

```
String authURL = service.getAuthorizationUrl();
```

Calling this URL will bring the user on the service connection page and right delegation for the application. If the user gives rights to the application to use the service on their behalf the service

will send you back a special code (verifier) that you must inject into the service to initiate the connection.

2.5. Set the verifier and initiate connection

As the verifier comes back to the application after an action of the final user, you have to set up a servlet or a JSF page (the URL of which is the callback URL you configured when you set up the service) to catch it and add it to the current session. Here is an example with JSF:

```
<f:metadata>
   <f:viewParam name="#{mySessionBean.twitterService.verifierParamName}"
      value="#{mySessionBean.twitterService.verifier}"
            required="true"
            requiredMessage="Error with Twitter. Retry later"/>
   <f:event type="preRenderView"
      listener="#{mySessionBean.twitterService.initAccessToken()}"/>
</f:metadata>
```

The service is now connected - you have an access token.

2.6. Send request to the service

You can now use the service with your rights:

```
TwitterProfile user = twitter.getMyProfile();
String fullName = user.getFullName();
```

Seam Social Qualifiers and Beans

3.1. Service Qualifiers

Each OAuth Application needs a specific qualifier bearing the `@ServiceRelated` Meta annotation. Out of the box Seam Social provides one default OAuth application qualifier for each social services provides (`@Twitter`, `@LinkedIn`, `@Facebook`). Should you need to support more than one application for a given service, you'd have to create a new service related qualifier (`@TwitterMyApp` for instance). Please refer to the "Extending Seam Social" section to learn more about this. Those qualifiers will be used on all the bean attached to a given OAuth application. They are useful to avoid ambiguous injection in the generic part of the API :

```
@Inject
@Twitter
OAuthService twiterService;
```

and

```
@Inject
@Facebook
OAuthService fbService;
```

Inject two different beans implementing the same interface (OAuthService).

3.2. Basic JSON Beans

JSON exchanges are managed by two beans:

- `JsonMapper` which deals with the implementation of JSON parser (Right now Jackson)
- `JsonService` which has a higher function and uses `JsonMapper` to provide decoupling from the Json parser.

3.3. Beans created by @OAuthApplication

Seam social uses the Generic bean functionality provided by Solder. Thus when you write :

```
@OAuthApplication(apiKey = "FQzlQC49UhvbMZoxUIvHTQ",
      apiSecret = "VQ5CZHG4qUoAkUUmckPn4iN4yyjBKcORTW0wnok4r1k")
@Twitter
```

```
@Produces
TwitterService twitterServiceProducer(TwitterService ts) {
   return ts;
}
```

The Generic extension creates the followings CDI beans with the same qualifier as the produced services (`@Twitter` in the example above) for you :

- OAuthService : the producer creates this first bean. It's the center of all OAuth exchange. Calling basic function in OAuthProvider it uses Settings to initiate connection and then uses Session information and JSon Mapper bean to transform exchange to and from object.
- OAuthServiceSettings : this bean has the same scope than the service and contains all the configuration of the OAuth Application plus the Service Related Qualifier of the service.
- OAuthProvider : this bean has the same scope than the service. It contains all the basic function to deal with OAuth exchange like the creation of tokens and request signatures. Its main purpose is to deal with implementation of OAuth management library (right now Scribe).
- OAuthSession : this session scoped bean contains all the OAuth Information needed to interact with remote service. Mainly the AccessToken.

Seam Social Advanced Usage

4.1. Working with Multi Service Manager

Seam Social provides a `MultiServicesManager` bean that can help you to manage multiple services and sessions for one user. Without this bean you'll be able to have multiple services but only one session for each service. The web app example application is a good starting point to learn how to use `MultiServicesManager` bean.

4.2. Provided Modules

Right now Seam Social comes with 3 basic service modules : Twitter, Facebook and LinkedIn. For this first Seam Social release (3.1.0), our main goal was to create a good core API for identification so provided modules have very basic functionalities. Check the JavaDoc to learn about them. We'll provide more functionalities and modules for the next release.

4.3. Extending Seam Social

To extend Seam Social by supporting a new service you'll have to provide the following class or ressources :

- A Qualifier for your service bearing the `@SocialRelated` meta annotation and the corresponding literal. You'll also need to create a properties file having the same name than your Qualifier. All of these should reside in the `org.jboss.seam.social` package.
- An implementation of the `OAuthService` interface. Extending the `OAuthServiceBase` is probably the easiest way to get it. It's good practice to create an interface for this bean to have an easy way to switch implementations if needed.
- A configuration bean implementing `ServiceConfiguration` having the service qualifier. Implements the `getServiceClass()` method by returning the class you created in step 2.
- A model class extending UserProfile abstract class.

The Facebook module is a good example of such an extension.