

Acegi を使って Java アプリケーションをセキュアにする、 第 1 回: アーキテクチャーの概要とセキュリティ・フィルター

Acegi Security System を使用した URL ベースのセキュリティ

Bilal Siddiqui
CEO
WAP Monster

2007年 3月 27日

この 3 回からなる連載記事で紹介するのは、Java™ エンタープライズ・アプリケーションのための非常に優れたオープン・ソース・セキュリティ・フレームワーク、AcegiSecurity System です。第 1 回目のこの記事では、Bilal Siddiqui が Acegi のアーキテクチャーとコンポーネントを取り上げ、Acegiを使用して単純な Java エンタープライズ・アプリケーションをセキュアにする方法を紹介します。

Acegi Security System は、Java エンタープライズ・アプリケーションのセキュリティ・コードを延々と作成する代わりとなる、非常に優れた使いやすい手段です。Acegiは Spring フレームワークで作成されたアプリケーション向けとなっていますが、これをいろいろなタイプの Java アプリケーションに適用できない理由はありません。この 3 回からなる連載記事では Acegi を基礎から紹介し、単純なエンタープライズ・アプリケーションから複雑なエンタープライズ・アプリケーションまで、Acegiを使用してセキュアにする方法を説明します。

連載第 1 回目ではまず、エンタープライズ・アプリケーションの一般的なセキュリティ問題を取り上げ、この問題を Acegi がどのように解決するかを紹介します。Acegiのアーキテクチャー・モデルとセキュリティ・フィルターについての説明では、このモデルとフィルターによってアプリケーションをセキュアにするための機能がほとんど実現されることがわかるはずです。また、エンタープライズ・セキュリティを実装する過程で、各フィルターがどのように動作し、どのように組み合わせられるか、そしてフィルター・チェーンがどのように機能するかを説明します。最後に、URLベースのセキュリティ・システムの Acegi 実装をデモするサンプル・アプリケーションで記事を締めくくります。連載の今後の 2 回の記事では、Acegiのより高度な使用例として、アクセス制御ポリシーを設計してホストし、そのポリシーを使用するように Acegi を構成する方法を詳しく説明する予定です。

まずは [Acegi をダウンロード](#) してください。サンプル・コードをコンパイルして、この連載で説明するサンプル・アプリケーションを実行するために必要です。また、Tomcat サーバーがワークステーションの一部として動作している必要もあります。

エンタープライズ・アプリケーションのセキュリティ

エンタープライズ・コンテンツ・マネージメント (ECM) アプリケーションが管理するのは、さまざまなタイプのデータ・ソース (ファイル・システム、リレーショナル・データベース、ディレクトリー・サービスなど) に保存されているエンタープライズ・コンテンツのオーサリングと処理です。そのため、ECM のセキュリティではこれらのデータ・ソースへのアクセスを制御しなければなりません。ECM アプリケーションでは、例えば製造企業が持つ設計、マーケティング、生産、および品質管理に関連するデータの読み取り、編集、または削除に対するユーザーのアクセス権を管理する場合が考えられます。

ECM のセキュリティ・シナリオでは、エンタープライズ・リソースのロケーター (ネットワーク・アドレス) にセキュリティを適用することによってアクセス制御を行うのが一般的です。このような単純なセキュリティ・モデルは、URL (Universal Resource Locator) セキュリティと呼ばれます。この記事 (そしてこの連載) でこれから説明するように、Acegi は URL セキュリティを実装するための包括的機能を提供します。

しかし多くのエンタープライズ・シナリオでは、URL セキュリティの適用だけでは十分ではありません。例えば、特定の製造会社が製造する特定の製品に関するデータが含まれる PDF 文書があるとします。この文書の一部には、会社の設計部門で編集および更新されることになっている設計データが含まれます。別の部分には生産データが含まれていて、これは生産管理者が使用することになります。このようなシナリオでは、よりきめの細かいセキュリティを実装して、文書のそれぞれの部分に異なるアクセス権を適用しなければなりません。

この記事では、まず URL セキュリティを実装するための Acegi の機能を紹介します。エンタープライズ・データへのアクセスをきめ細かく制御できる Acegi のメソッド・ベースのセキュリティについては、次の記事で説明します。

Acegi Security System

Acegi Security System では [Acegi のセキュリティ・フィルター](#) を使用して、エンタープライズ・アプリケーションに認証およびアクセス許可サービスを提供します。このフレームワークにはさまざまなタイプのフィルターがあり、アプリケーションの要件に応じて構成できます。各種のセキュリティ・フィルターについての説明はひとまず置いて、以下に Acegi のセキュリティ・フィルターを構成して実現できるタスクを挙げます。

1. ユーザーがセキュア・リソースにアクセスする前に、ユーザーにログインを求めるプロンプトを出す。
2. パスワードなどのセキュリティ・トークンをチェックしてユーザーを認証する。
3. 認証済みユーザーにセキュア・リソースに対するアクセス特権があるかどうかをチェックする。
4. 認証が成功してアクセス許可されたユーザーを、要求するセキュア・リソースにリダイレクトする。

5. セキュア・リソースに対するアクセス特権がないユーザーにアクセス拒否ページを表示する。
6. 認証が成功したユーザーをサーバーに記憶し、セキュア cookie をユーザーのクライアントに設定する。次の認証は、ユーザーにログインを要求せずにそのcookie を使用して実行できるようにする。
7. サーバー側のセッション・オブジェクトに認証情報を保存し、リソースに対するその後の要求にセキュアに対応できるようにする。
8. サーバー側のオブジェクトにセキュリティー情報のキャッシュを作成して保持し、パフォーマンスを最適化する。
9. ユーザーがサインアウトしたときに、ユーザーのセキュア・セッションに対して保持されているサーバー側のオブジェクトを破棄する。
10. ユーザーのセキュリティー情報と ECM のアクセス制御ポリシーを保存するために使用されるバックエンドの各種データ・ストレージ・サービス (ディレクトリー・サービスやリレーショナル・データベースなど) と通信する。

上記のリストからわかるように、エンタープライズ・アプリケーションをセキュアにするために必要となるタスクはほとんどすべて、Acegi のセキュリティー・フィルターを使って実行することができます。

アーキテクチャーとコンポーネント

Acegi を理解すればするほど、Acegi を簡単に操作できるようになります。まずはこのセクションで Acegi コンポーネントについて説明してから、このフレームワークがコンポーネントを結合し、その依存関係を表現するためにどのように制御の反転 (IoC) と XML 構成ファイルを使用するかを説明することにします。

4 大コンポーネント

Acegi Security System を構成する主なコンポーネントは、以下に説明するフィルター、マネージャー、プロバイダー、そしてハンドラーの4つのタイプに分けられます。

フィルター

最上位レベルのコンポーネントで、認証処理、セッションの処理、ログアウトなどの共通セキュリティー・サービスを提供します。フィルターについては[後ほど詳しく説明](#)します。

マネージャー

フィルターはセキュリティー関連の機能を上位レベルで抽象化したものでしかありませんが、マネージャーとプロバイダーは実際の認証処理とログアウト・サービスを実装するために使用されます。マネージャーは、さまざまなプロバイダーが提供する下位レベルのセキュリティー・サービスを管理するコンポーネントです。

プロバイダー

ディレクトリー・サービス、リレーショナル・データベース、あるいは単純なメモリー内オブジェクトなどの各種データ・ストレージ・サービスとの通信には、さまざまなプロバイダーを使用できます。つまり、ユーザー・ベースおよびアクセス制御ポリシーは、これらのデータ・ストレージ・サービスのどれにでも保存できるということです。実行時にはAcegiのマネージャーが該当するプロバイダーを選択します。

ハンドラー

場合によっては、タスクが複数のステップに分かれていて、それぞれのステップが特定のハンドラーで実行されることがあります。例えば、Acegi のログアウト・フィルターは2 つのハンドラーを使用して HTTP クライアントのサインアウトを行います。一方はユーザーの HTTP セッションを無効にするハンドラー、そしてもう一方はユーザーの cookie を破棄するハンドラーです。複数のハンドラーがあると、アプリケーションの要件に応じて機能するように Acegi を構成する一方で、柔軟性がもたらされ、アプリケーションをセキュアにするために必要なステップを実行するハンドラーを、意のままに選択することができます。

制御の反転

Acegi のコンポーネントは、互いに依存しあってエンタープライズ・アプリケーションをセキュアにします。例えば、認証処理フィルターには、適切な認証プロバイダーを選択するために認証マネージャーが必要です。つまり、Acegi を使うには、そのコンポーネントの依存関係を表現し、管理する能力が必要になります。

Java コンポーネントの依存関係の管理には、一般的に IoC 実装が使用されます。IoC は、重要な 2 つの機能を提供します。

1. アプリケーションに必要なコンポーネントと、コンポーネント間の依存関係を表現する構文を提供する。
2. 必要なコンポーネントを実行時に確実に使用できるようにする。

XML 構成ファイル

Acegi がコンポーネントの管理に使用するのは、Spring フレームワーク ([「参考文献」](#)を参照) に付属している人気の高いオープン・ソース IoC 実装です。Spring では、XML 構成ファイルを使ってコンポーネントの依存関係を表現します(リスト 1 を参照)。

リスト 1. Spring 構成ファイルの構造

```
<beans>
  <bean id="filterChainProxy" class="org.acegisecurity.util.filterChainProxy">
    <property name="filterInvocationDefinitionSource">
      <value> value here </value>
    </property>
  </bean>

  <bean id="AuthenticationProcessingFilter"
    class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">
    <property name="AuthenticationManager" ref="authManager"/>
    <!-- Other properties -->
  </bean>

  <bean id="authManager"
    class="org.acegisecurity.providers.ProviderManager">
    <property name="providers">
      <!-- List of providers here -->
    </property>
  </bean>
  <!-- Other bean tags -->
</beans>
```

ご覧のように、Acegi が使用する Spring の XML 構成ファイルには `<beans>` タグが 1 つしかありません。このタグが、他の多数の `<bean>` タグをラップします。すべての Acegi コンポーネント

(つまり、フィルター、マネージャー、プロバイダーなど)は、実際にはJavaBeansです。XML 構成ファイル内のそれぞれの `<bean>` タグが、1 つの Acegi コンポーネントを表します。

XML 構成ファイルについての追加注意事項

まず初めに目につくのは、`<bean>` タグごとに、コンポーネントが使用するクラスを識別する `class` 属性があることです。また `<bean>` タグには、Acegi コンポーネントとして振る舞うインスタンス (Java オブジェクト) を識別する `id` 属性もあります。

具体的に説明すると、[リスト 1](#) の最初の `<bean>` タグ

は、`org.acegisecurity.util.filterChainProxy` というクラスのコンポーネント・インスタンス、`filterChainProxy` を識別しています。

Bean の依存関係は、`<bean>` タグの子タグで表現されます。最初の `<bean>` タグの `<property>` 子タグがその一例です。この `<property>` 子タグは、`<bean>` タグが依存する値、あるいは他の Bean を定義します。

[リスト 1](#) では、最初の `<bean>` タグの `<property>` 子タグに `name` 属性と `<value>` 子タグがあります。このそれぞれが、Bean が依存するプロパティの名前と値を定義します。

同様に、[リスト 1](#) の 2 番目と 3 番目の `<bean>` タグは、フィルター Bean がマネージャー Bean に依存することを定義しています。2 番目の `<bean>` タグで表わされているのがフィルター Bean で、3 番目の `<bean>` タグで表わされているのがマネージャー Bean です。

フィルターの `<bean>` タグに含まれているのは、`name` と `ref` という 2 つの属性を持つ `<property>` 子タグです。`name` 属性はフィルター Bean のプロパティを定義し、`ref` 属性はマネージャー Bean のインスタンス (インスタンス名) を参照します。

次のセクションで、XML 構成ファイルに Acegi フィルターを構成する方法を説明します。構成したフィルターは、この記事の終わりで紹介する Acegi サンプル・アプリケーションで使用するようになります。

セキュリティー・フィルター

前にも言ったように、Acegi Security System ではセキュリティー・フィルターを使用して、エンタープライズ・アプリケーションに認証サービスとアクセス許可サービスを提供します。使用できるフィルターはさまざまで、アプリケーションの要件に合わせて構成できます。このセクションで紹介するのは、なかでも重要性の高い 5 つの Acegi セキュリティー・フィルターです。

セッション統合フィルター

通常の場合、最初に構成する Acegi フィルターはセッション統合フィルター (SIF) です。SIF はセキュリティー・コンテキスト・オブジェクトを作成するフィルターで、このオブジェクトがセキュリティー関連情報のプレースホルダーとなります。他の Acegi フィルターはこのセキュリティー・コンテキストにセキュリティー情報を保存し、このセキュリティー・コンテキストにある情報を使用します。

SIF はセキュリティー・コンテキストを作成すると、フィルター・チェーンに含まれるその他のフィルターを呼び出します。呼び出されたフィルターはセキュリティー・コンテキストを取得し、変更を加えます。例えば認証処理フィルター(次に説明)は、ユーザー名、パスワード、Eメール・アドレスなどのユーザー情報をセキュリティー・コンテキストに保存します。

すべてのフィルターが処理を終えると、SIF はセキュリティー・コンテキストが更新されているかどうかをチェックします。いずれかのフィルターがセキュリティー・コンテキストに変更を加えた場合、SIFはその変更をサーバー側のセッション・オブジェクトに保存します。セキュリティー・コンテキストに変更が見つからなければ、SIF はそのセキュリティー・コンテキストを破棄します。

SIF は XML 構成ファイルでリスト 2 に示すように構成されます。

リスト 2. SIF の構成

```
<bean id="httpSessionContextIntegrationFilter"
      class="org.acegisecurity.context.HttpSessionContextIntegrationFilter"/>
```

認証処理フィルター

Acegi は認証処理フィルター (APF) を使用して認証を行います。APF が使用する認証 (ログイン) フォームに、ユーザーがユーザー名とパスワードを入力して認証をトリガーします。

クライアント要求からのユーザー名とパスワードの抽出、バックエンド・ユーザー・ベースからのユーザーのパラメーター読み出し、そしてその情報を使用したユーザーの認証など、バックエンドの認証処理タスクはすべて APF によって実行されます。

APF を構成する際に指定しなければならないパラメーターは以下のとおりです。

- 認証マネージャーとして、認証プロバイダーの管理に使用する認証マネージャーを指定します。
- フィルター・プロセス URL として、ログイン・フォームでクライアントが Sign In ボタンを押したときにアクセスする URL を指定します。Acegi はこの URL 要求を受信した時点で APF を呼び出します。
- デフォルト・ターゲット URL として、認証とアクセス許可が成功した場合にユーザーに表示するページを指定します。
- 認証失敗 URL として、認証が失敗した場合にユーザーに表示するページを指定します。

APF は、ユーザーの要求オブジェクトからユーザー名、パスワード、そしてその他の情報を取得し、この情報を認証マネージャーに渡します。認証マネージャーは適切なプロバイダーを使用して、バックエンド・ユーザー・ベースから詳細なユーザー情報(ユーザー名、パスワード、Eメール・アドレス、ユーザーのアクセス権または特権など)を読み取り、ユーザーを認証して情報を `Authentication` オブジェクトに保存します。

最後に APF は、この `Authentication` オブジェクトを SIF が事前に作成したセキュリティー・コンテキストに保存します。セキュリティー・コンテキストに保存された `Authentication` オブジェクトは、後でアクセス許可を決定する際に使用されることになります。

APF はリスト 3 に示すように構成します。

リスト 3. APF の構成

```
<bean id="authenticationProcessingFilter"
      class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">
  <property name="authenticationManager"
    ref="authenticationManager" />
  <property name="filterProcessesUrl"
    value="/j_acegi_security_check" />
  <property name="defaultTargetUrl"
    value="/protected/protected1.jsp" />
  <property name="authenticationFailureUrl"
    value="/login.jsp?login_error=1" />
</bean>
```

このコードを見ると、APF は上記で説明した 4 つのパラメーターに依存していることがわかります。それぞれのパラメーターは、リスト 3 の `<property>` タグとして構成されます。

ログアウト処理フィルター

ログアウト処理フィルター (LPF) は、Acegi がログアウト処理を管理するために使用するフィルターです。LPF は、クライアントからログアウト要求が出された時点で動作します。ログアウト要求はクライアントが呼び出した URL から識別します。

LPF はリスト 4 に示すように構成します。

リスト 4. LPF の構成

```
<bean id="logoutFilter" class="org.acegisecurity.ui.logout.LogoutFilter">
  <constructor-arg value="/logoutSuccess.jsp"/>
  <constructor-arg>
    <list>
      <bean class="org.acegisecurity.ui.logout.SecurityContextLogoutHandler"/>
    </list>
  </constructor-arg>
</bean>
```

上記を見ると、LPF のコンストラクターは 2 つのパラメーター、ログアウト成功 URL (`/logoutSuccess.jsp`) とハンドラーのリストを取ることがわかります。ログアウト成功 URL は、ログアウト・プロセスが完了した後にクライアントをリダイレクトするために使用されます。ハンドラーは実際のログアウト・プロセスを実行します(ここでは 1 つのハンドラーしか構成していませんが、HTTP セッションを無効にするには十分です)。ハンドラーについては、連載第 2 回で詳しく説明します。

例外変換フィルター

例外変換フィルター (ETF) は、認証およびアクセス許可プロシーチャーで発生した例外 (許可失敗など) を処理するフィルターです。例外が発生した場合、ETF がその処理内容を決定します。

例えば、認証されていないユーザーが保護されているリソースにアクセスしようとする、ユーザーに認証を促すログイン・ページを表示するようにしたり、同様にアクセス許可で失敗した場合にはアクセス拒否ページを表示するように ETF を構成できます。

ETF はリスト 5 に示すように構成します。

リスト 5. ETF の構成

```
<bean id="exceptionTranslationFilter"
  class="org.acegisecurity.ui.ExceptionTranslationFilter">
  <property name="authenticationEntryPoint">
    <bean
      class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilterEntryPoint">
      <property name="loginFormUrl" value="/login.jsp" />
    </bean>
  </property>
  <property name="accessDeniedHandler">
    <bean class="org.acegisecurity.ui.accessDeniedHandlerImpl">
      <property name="errorPage" value="/accessDenied.jsp" />
    </bean>
  </property>
</bean>
```

リスト 5 では、ETF は `AuthenticationEntryPoint` と `accessDeniedHandler` という 2 つのパラメーターを使用しています。`AuthenticationEntryPoint` はログイン・ページを指定し、`accessDeniedHandler` はアクセス拒否ページを指定するプロパティです。

インターセプター・フィルター

Acegi は、インターセプター・フィルターを使ってアクセスを許可するかどうかを決定します。インターセプター・フィルターの構成で必要なのは、APF による認証が成功した後に動作するようにすることです。インターセプターはアプリケーションのアクセス制御ポリシーを使用してアクセス許可を決定します。

アクセス制御ポリシーを設計してディレクトリー・サービスでホストする方法、そしてそのアクセス制御ポリシーを読み取るように Acegi を構成する方法については次回の記事で説明するので、ここでは Acegi を使って単純なアクセス制御ポリシーを構成する方法に焦点を絞ります。後ほどサンプル・アプリケーションをビルドする際に、この単純なアクセス制御ポリシーを使用します。

単純なアクセス制御ポリシーの構成手順は、以下の 2 つのステップに分けられます。

1. アクセス制御ポリシーを作成する。
2. アクセス制御ポリシーに従って Acegi のインターセプター・フィルターを構成する。

ステップ 1. 単純なアクセス制御ポリシーを作成する

まず [リスト 6](#) を見てください。このリストで、ユーザーとユーザーの役割を定義する方法がわかります。

リスト 6. ユーザーに対する単純なアクセス制御ポリシーの定義

```
alice=123,ROLE_HEAD_OF_ENGINEERING
```

リスト 6 のアクセス制御ポリシーでは、名前が `alice`、パスワードが `123`、役割が `ROLE_HEAD_OF_ENGINEERING` のユーザーを定義しています (次のセクションでは、任意の数のユー

ザーとその役割をファイルに定義し、Acegi インターセプター・フィルターがそのファイルを使用するように構成する方法を説明します)。

ステップ 2. Acegi のインターセプター・フィルターを構成する

インターセプター・フィルターはリスト 7 で構成しているように、3 つのコンポーネントを使用してアクセス許可を決定します。

リスト 7. インターセプター・フィルターの構成

```
<bean id="filterInvocationInterceptor"
      class="org.acegisecurity.intercept.web.FilterSecurityInterceptor">
  <property name="authenticationManager" ref="authenticationManager" />
  <property name="accessDecisionManager" ref="accessDecisionManager" />
  <property name="objectDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
      /protected/**=ROLE_HEAD_OF_ENGINEERING
      /**=IS_AUTHENTICATED_ANONYMOUSLY
    </value>
  </property>
  <!-- More properties of the interceptor filter -->
</bean>
```

リスト 7 に示されているように、構成する必要がある 3 つのコンポーネントは AuthenticationManager、accessDecisionManager、objectDefinitionSource です。

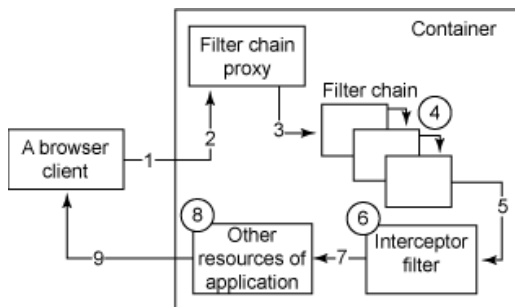
- AuthenticationManager コンポーネントは、[認証処理フィルター](#)を紹介したときに説明した認証マネージャーに相当します。インターセプター・フィルターはこの AuthenticationManager を使用して、アクセス許可のプロセス中にクライアントを再認証します。
- accessDecisionManager コンポーネントはアクセス許可のプロセスを管理します。これについては、次の記事で詳しく説明します。
- objectDefinitionSource コンポーネントには、アクセス制御定義が含まれます。この定義に従ってアクセス許可が行われることになります。[リスト 7](#)では、objectDefinitionSource プロパティの値に 2 つの URL (/protected/* および /*) が含まれています。つまりこの値は、この 2 つの URL の役割を定義します。ここでは /protected/* URL の役割は ROLE_HEAD_OF_ENGINEERING となっていますが、アプリケーションの要件に応じて任意の役割を定義できます。
- [リスト 6](#)では、ユーザー alice に対して ROLE_HEAD_OF_ENGINEERING を定義しました。したがって、alice は /protected/* URL にアクセスできます。

フィルター機能の仕組み

すでに説明したように、Acegi のコンポーネントは互いに依存しあってアプリケーションをセキュアにします。今後の記事で、セキュリティー・フィルターを特定の順序で適用するように Acegi を構成する方法を説明しますが、これによって作成されるのがフィルター・チェーンです。このフィルター・チェーンを目的として、Acegi はアプリケーションをセキュアにするために構成したすべてのフィルターをラップするフィルター・チェーン・オブジェクトを維持管理します。図 1 は、Acegi フィルター・チェーンのライフ・サイクルです。フィルター・チェーンは、クライアン

トが HTTP 要求をアプリケーションに送信すると開始されます(図 1 には、ブラウザー・クライアントにサービス提供するコンテナを示しています)。

図 1. Acegi のフィルター・チェーンをホストしてブラウザー・クライアントにセキュアなサービスを提供するコンテナ



以下のステップで、フィルター・チェーンのライフ・サイクルを説明します。

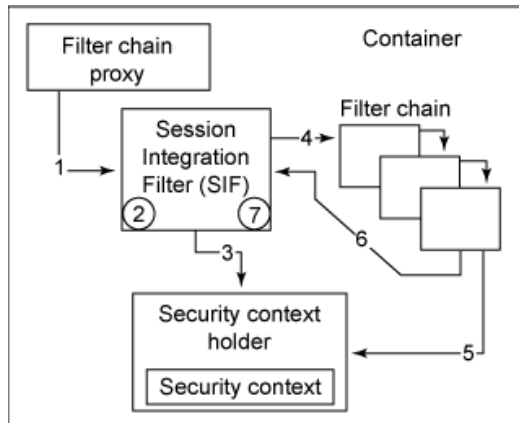
1. ブラウザー・クライアントがアプリケーションに HTTP 要求を送信します。
2. コンテナは HTTP 要求を受信すると、その HTTP 要求に含まれる情報をラップする要求オブジェクトを作成します。コンテナは、異なるフィルターが共通して処理できる応答オブジェクトも作成します。フィルターはこの応答オブジェクトを処理して要求側クライアントに対する HTTP 応答を準備します。次にコンテナは Acegi のフィルター・チェーン・プロキシ (プロキシ・フィルター) を呼び出します。このプロキシが持っている情報は、実際のフィルターの適用シーケンスです。コンテナはプロキシを呼び出すときに、要求、応答およびフィルター・チェーン・オブジェクトを渡します。
3. プロキシ・フィルターがフィルター・チェーン内の最初のフィルターを呼び出し、要求、応答およびフィルター・チェーン・オブジェクトを渡します。
4. フィルター・チェーンのフィルターが順次それぞれの処理を行います。フィルターはチェーン内の次のフィルターを呼び出すことによって、任意の時点で処理を終了できます。フィルターが一切の処理を実行しないことを選択する場合があります(例えば APF は、送られてきた要求に認証の必要がないことがわかった時点で、その処理を終了します)。
5. 認証フィルターは処理を終了すると、アプリケーション内に構成されたインターセプター・フィルターに要求オブジェクトと応答オブジェクトを渡します。
6. インターセプターは、要求側クライアントに、要求する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。
7. インターセプターがアプリケーション (例えば、認証とアクセス許可に成功した場合に、クライアントが要求する JSP ページ) に制御を移します。
8. アプリケーションが応答オブジェクトにコンテンツを上書きします。
9. これで応答オブジェクトの準備は完了です。コンテナは応答オブジェクトを HTTP 応答に変換して要求側クライアントに送信します。

Acegi フィルターをさらに詳しく理解できるよう、セッション統合フィルター (SIF) と認証処理フィルター (APF) の 2 つに絞って、その動作を詳しく説明することにします。

SIF がセキュリティ・コンテキストを作成する方法

図 2 に、SIF によるセキュリティ・コンテキストの作成に伴うステップを示します。

図 2. SIF によるセキュリティ・コンテキストの作成



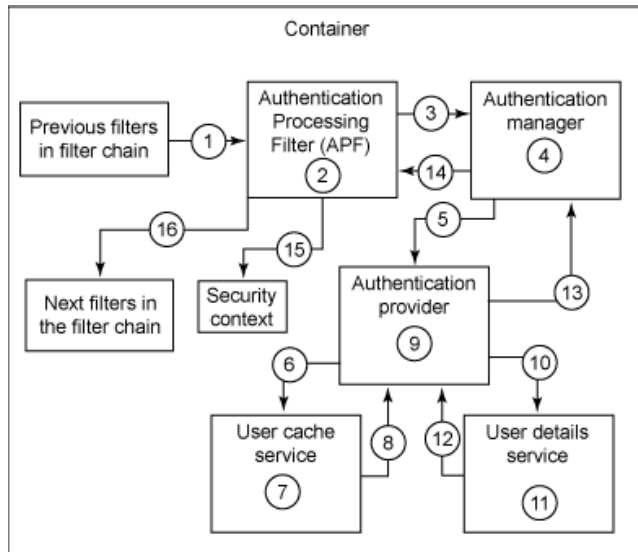
上記のステップを以下に詳しく説明します。

1. Acegi のフィルター・チェーン・プロキシーが SIF を呼び出し、要求、応答およびフィルター・チェーン・オブジェクトを渡します。通常、SIF はフィルター・チェーンの最初のフィルターとして構成されること注意してください。
2. SIF は、この Web 要求を以前に処理したことがあるかどうかをチェックします。処理したことがあるとわかった場合、以降の処理は行わずにフィルター・チェーン内の次のフィルターに制御を移します(以下のステップ 4 を参照)。特定の Web 要求が行われている間に初めて呼び出されたことがわかった場合、SIF はフラグを設定します。次の呼び出しの際には、このフラグで、SIF がすでに呼び出されたことが示されます。
3. SIF は、セッション・オブジェクトが存在し、そこにセキュリティ・コンテキストが含まれているかどうかをチェックします。セッション・オブジェクトがある場合、そこからセキュリティ・コンテキストを取得し、セキュリティ・コンテキスト・ホルダーと呼ばれる一時プレースホルダーにそのセキュリティ・コンテキストを配置します。セッション・オブジェクトがなければ、SIF はセキュリティ・コンテキストを新規に作成してセキュリティ・コンテキスト・ホルダーに書き込みます。セキュリティ・コンテキスト・ホルダーはアプリケーションのスコープ内に存在するため、他のセキュリティ・フィルターからもアクセスすることが可能です。
4. SIF がフィルター・チェーン内の次のフィルターを呼び出します。
5. その他のフィルターは、必要に応じてセキュリティ・コンテキストを編集します。
6. フィルター・チェーンの処理が完了すると、SIF に制御が移ります。
7. SIF は、フィルター・チェーンの処理中に他のいずれかのフィルターによってセキュリティ・コンテキストが変更されているかどうかをチェックします(例えば、APF がセキュリティ・コンテキストにユーザー詳細を保存している場合があります)。変更されていれば、セッション・オブジェクト内のセキュリティ・コンテキストを更新します。つまり、フィルター・チェーンの処理中に行われたセキュリティ・コンテキストの変更はすべて、セッション・オブジェクト内に含まれるようになるということです。

APF がユーザーを認証する方法

図 3 に、APF によるユーザー認証に伴うステップを示します。

図 3. APF によるユーザー認証



上記のステップを以下に詳しく説明します。

1. フィルター・チェーン内の前のフィルターが、要求、応答およびフィルター・チェーン・オブジェクトを APF に渡します。
2. APF は、要求オブジェクトから取得したユーザー名、パスワード、およびその他の情報を使用して認証トークンを作成します。
3. APF が認証マネージャーに認証トークンを渡します。
4. 認証マネージャーには、1 つまたは複数の認証プロバイダーが含まれる場合があります。それぞれのプロバイダーがサポートする認証タイプは 1 つだけです。認証マネージャーは、そこに含まれるプロバイダーのうちのどれが、APF から受け取った認証トークンをサポートするかをチェックします。
5. 認証マネージャーが認証に適切なプロバイダーに認証トークンを渡します。
6. 認証プロバイダーは認証トークンからユーザー名を抽出し、これをユーザー・キャッシュ・サービスと呼ばれるサービスに渡します。Acegi は認証済みユーザーのキャッシュを維持しているため、ユーザーが次回サインインしたときに、Acegi はそのユーザーの詳細 (ユーザー名、パスワード、アクセス特権など) をバックエンドのデータ・ストレージから読み取るのではなく、このキャッシュからロードできます。これによってパフォーマンスが向上します。
7. ユーザー・キャッシュ・サービスによって、ユーザーの詳細がキャッシュに含まれているかどうかチェックされます。
8. ユーザー・キャッシュ・サービスが認証プロバイダーにユーザーの詳細を返します。キャッシュにユーザー詳細が含まれていない場合は、ヌルを返します。
9. 認証プロバイダーは、キャッシュ・サービスからユーザー詳細とヌルのどちらが返されたかを確認します。
10. キャッシュがヌルを返した場合、認証プロバイダーはユーザー詳細サービスと呼ばれる別のサービスにユーザー名 (ステップ 6 で抽出) を渡します。
11. ユーザー詳細サービスが、ユーザーの詳細が含まれるバックエンドのデータ・ストレージ (ディレクトリー・サービスなど) と通信します。

12. ユーザー詳細サービスがユーザーの詳細を返します。ユーザーの詳細が見つからなければ、認証例外をスローします。
13. ユーザー・キャッシュ・サービスまたはユーザー詳細サービスのいずれかが有効なユーザー詳細を返すと、認証プロバイダーはユーザーが提供したセキュリティ・トークン(パスワードなど)を、キャッシュまたはユーザー詳細サービスから返されたパスワードと突き合わせます。一致する場合、認証プロバイダーは認証マネージャーにユーザーの詳細を返します。一致しない場合は認証例外をスローします。
14. 認証マネージャーが APF にユーザーの詳細を返します。これで、ユーザーの認証は成功したことになります。
15. APF は、[図 2](#) のステップ 3 で作成されたセキュリティ・コンテキストにユーザー詳細を保存します。
16. APF がフィルター・チェーン内の次のフィルターに制御を移します。

単純な Acegi アプリケーション

Acegi についてかなり理解してもらえたと思うので、説明のまとめとして、今までに学んだ内容で何ができるかを見てみましょう。簡単なデモ用に、サンプル・アプリケーション([「ダウンロード」](#)を参照)を設計し、リソースを保護する Acegi を構成してあります。

このサンプル・アプリケーションには、5 つの JSP ページ (index.jsp、protected1.jsp、protected2.jsp、login.jsp、accessDenied.jsp)が含まれています。

index.jsp は、アプリケーションのウェルカム・ページです。このページは、3 つのハイパーリンクをユーザーに表示しています ([図 4](#) を参照)。

図 4. サンプル・アプリケーションのウェルカム・ページ

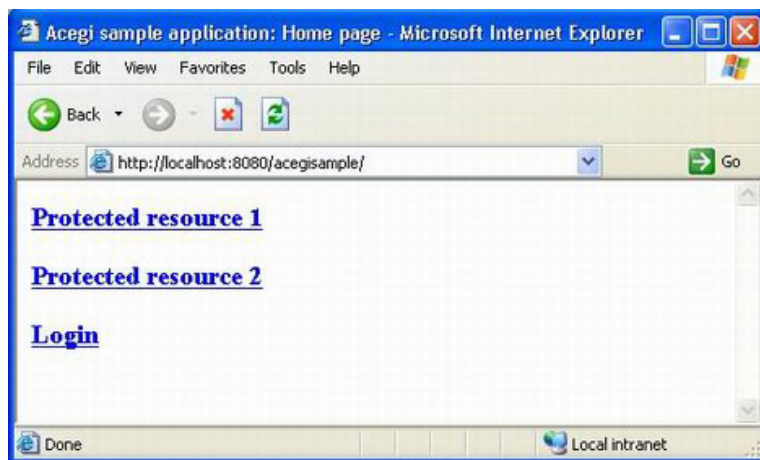


図 4 に示したリンクのうち、2 つは保護リソース '(protected1.jsp および protected2.jsp) へのリンクで、3 つ目はログイン・ページ (login.jsp) へのリンクです。accessDenied.jsp ページは、Acegi がユーザーに保護リソースへのアクセスが許可されていないことを検出した場合にのみ表示されます。

ユーザーが保護リソースのいずれかにアクセスしようとする、このサンプル・アプリケーションはログイン・ページを表示します。このログイン・ページを使ってユーザーがサインインすると、ユーザーは要求する保護リソースに自動的にリダイレクトされます。

ログイン・ページを直接要求するには、ウェルカム・ページの 3 つ目のリンク (Login) をクリックします。これによって表示されるログイン・ページで、ユーザーはサインインできます。ユーザーがサインインすると、アプリケーションはユーザーをprotected1.jsp にリダイレクトします。これは、ユーザーが特定の保護リソースを要求しないでサインインしたときに常に表示されるデフォルト・リソースです。

サンプル・アプリケーションの構成

この記事でダウンロードするソース・コードには、acegi-config.xml という名前の XML 構成ファイルが含まれています。Acegi フィルターの構成は、このファイルにあります。[セキュリティ・フィルターについての説明](#)で例を記載したので、この構成はもうお馴染みのはずです。

このサンプル・アプリケーションには web.xml ファイルも作成してあります (リスト 8 を参照)。

リスト 8. サンプル・アプリケーションの web.xml ファイル

```
<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/acegi-config.xml</param-value>
  </context-param>
  <filter>
    <filter-name>Acegi Filter Chain Proxy</filter-name>
    <filter-class>
      org.acegisecurity.util.FilterToBeanProxy
    </filter-class>
    <init-param>
      <param-name>targetClass</param-name>
      <param-value>
        org.acegisecurity.util.FilterChainProxy
      </param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>Acegi Filter Chain Proxy</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
</web-app>
```

この web.xml ファイルが構成する内容は以下のとおりです。

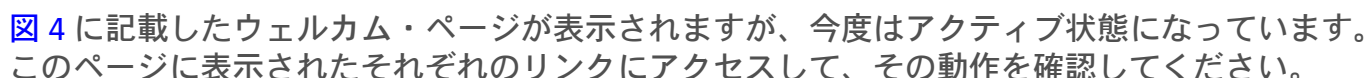
- `<context-param>` タグに含まれる acegi-config.xml ファイルの URL。
- `<filter>` タグに含まれる Acegi フィルター・チェーン・プロキシー・クラスの名前。
- `<filter-mapping>` タグに含まれる、URL と Acegi フィルター・チェーン・プロキシー間のマッピング。Acegi フィルター・チェーン・プロキシーには、アプリケーションのすべての URL (/*) をマッピングすることも可能です。Acegi は、フィルター・チェーン・プロキシーにマッピングされたすべての URL にセキュリティを適用します。
- `<listener>` タグに含まれるアプリケーション・コンテキスト・ローダー。このローダーは Spring の IoC フレームワークをロードします。

アプリケーションのデプロイメントおよび実行手順

サンプル・アプリケーションは至って簡単にデプロイおよび実行できます。必要なステップは以下の2つだけです。

1. このチュートリアルでダウンロードしたソース・コードのうち、acegisample.war ファイルを Tomcat システムの `webapps` ディレクトリーにコピーします。
2. Acegi Security System ホーム・ページから、[Download and unzip acegi-security-1.0.3.zip](#) をダウンロードして解凍します。すると、acegi-security-sample-tutorial.war という名前のサンプル・アプリケーションが見つかるはずです。この war ファイルを展開して、見つかったすべての jar を WEB-INF/lib フォルダーに入れます。WEB-INF/lib フォルダー内のすべての JAR を、acegisample.war アプリケーションの WEB-INF/lib フォルダーにコピーします。

これで、サンプル・アプリケーションを試す準備ができました。Tomcat を起動し、ブラウザーで <http://localhost:8080/acegisample/> を指定してください。

 図 4 に記載したウェルカム・ページが表示されますが、今度はアクティブ状態になっています。このページに表示されたそれぞれのリンクにアクセスして、その動作を確認してください。

まとめ

連載「Acegi による Java アプリケーションのセキュリティ保護」の第 1 回目となるこの記事では、Acegi Security System の機能、アーキテクチャー、そしてコンポーネントを説明しました。なかでも Acegi のセキュリティ・フレームワークに不可欠なセキュリティ・フィルターについては、かなりの知識を得られたはずです。また、XML 構成ファイルを使用してコンポーネントの依存関係を構成する方法を説明し、URL ベースのセキュリティを実装するサンプル・アプリケーションでの実際の Acegi セキュリティ・フィルターも紹介しました。

この記事で説明したセキュリティ手法はかなり単純なもので、この手法を実装するために使用した Acegi 機能も同じく単純なものです。次回の記事で取り組むのはこれより高度な Acegi の使用例で、その手始めとしてアクセス制御ポリシーを作成してディレクトリー・サービスに保存します。さらに、Acegi がディレクトリー・サービスと連携して、作成したアクセス制御ポリシーを実装するように構成する方法についても説明します。

ダウンロード

内容	ファイル名	サイズ
Source code for this article	j-acegi1.zip	10KB

著者について

Bilal Siddiqui

Bilal Siddiqui は XML のコンサルタントです。パキスタン、ラホールの University of Engineering and Technology で電子工学を学んだのち、1995年に産業制御システム向けのソフトウェア・ソリューションの設計を始めました。その後 XML に転向し、C++ のプログラミング経験を生かして Web および WAP ベースの XML 処理ツール、サーバー・サイド構文解析ソリューション、およびサービス・アプリケーションの構築を行うようになりました。

© Copyright IBM Corporation 2007

(www.ibm.com/legal/copytrade.shtml)

商標

(www.ibm.com/developerworks/jp/ibm/trademarks/)