Java EE 8 Security API 入門, 第 3 回: IdentityStore を使用してユーザー資格情報へのアクセスをセキュリティーで保護する

新しい IdentityStore API を使用したユーザーの認証・許可

Alex Theedom

2018年 10月 18日

新たに導入された IdentityStore インターフェースを使用して、Java Web アプリケーション内で RDBMS または LDAP アイデンティティー・ストレージをセットアップして構成する方法を学んでください。

このシリーズについて
新しくリリースされた待望の Java EE Security API (JSR 375) によって、クラウドとマイクロサービス・コンピューティング時代に初めて Java エンタープライズ・セキュリティーが導入されます。このシリーズでは、この新しいセキュリティー・メカニズムがあらゆる Java EE コンテナーの実装でセキュリティー処理を単純化および標準化する仕組みを紹介した上で、クラウド対応のプロジェクト内で実際にこのセキュリティー・メカニズムを活用する方法を説明します。

このシリーズの最初の記事では、新しくリリースされた Java EE Security API (JSR 375) の基本的な機能とコンポーネントについて、新たに導入された `IdentityStore` インターフェースを含め、大まかに説明しました。今回の記事では、この `IdentityStore` を使用して Java Web アプリケーション内にユーザー資格情報データを安全に保管し、ユーザー資格情報へのアクセスをセキュリティーで保護する方法を説明します。

`IdentityStore` というアイデンティティー・ストアの抽象化は、Java EE Security API 仕様のリリースで最も大きな話題となっている 3 つの機能のうちの 1 つです。アイデンティティー・ストアとは、ユーザーのアイデンティティー・データ (ユーザー名、グループ・メンバーシップなど、呼び出し側の資格情報を検証するために使用する情報) を保管するデータベースのことです。`IdentityStore` はあらゆる認証メカニズムで使用されるように意図されていますが、第 2 回で紹介した Java EE 8 の `HttpAuthenticationMechanism` に統合すると、とりわけ効果を発揮します。

コードを入手する

Soteria をインストールする

`IdentityStore` について探るために、Java EE 8 Security API のリファレンス実装となっている Soteria を使用します。Soteria を取り込むには、以下に説明する 2 つの方法があります。

1. POM 内で明示的に Soteria を指定する

POM 内で以下の Maven コーディネートを使用して Soteria を指定します。

リスト 1. Soteria プロジェクトの Maven コーディネート

```
<dependency>
  <groupId>org.glassfish.soteria</groupId>
  <artifactId>javax.security.enterprise</artifactId>
  <version>1.0</version>
</dependency>
```

2. 組み込み Java EE 8 コーディネートを使用する

Java EE 8 対応のサーバーでは、新しい Java EE 8 Security API の独自の実装を使用するか、Sotoria の実装に依存することになります。いずれにしても、必要となるのは以下の Java EE 8 コーディネートだけです。

リスト 2. Java EE 8 の Maven コーディネート

```
<dependency>
 <groupId>javax</groupId>
 <artifactId>javaee-api</artifactId>
 <version>8.0</version>
 <scope>provided</scope>
</dependency>
```

`IdentityStore` に関連するインターフェース、クラス、アノテーションは、`javax.security.enterprise.identitystore` パッケージに含まれています。

IdentityStore の仕組み

JAAS LoginModule インターフェースと同様に、`IdentityStore` はアイデンティティー・ストアとやり取りしてユーザーを認証し、グループ・メンバーシップを取得するために使用する抽象化です。`IdentityStore` は `HttpAuthenticationMechanism` と上手く連動するように設計されていますが、必要に応じて任意の認証メカニズムを使用できます。また、コンテナーを使用するかどうかも任意ですが、ほとんどのシナリオでは `IdentityStore` メカニズムでコンテナーを使用することが推奨されます。`IdentityStore` をコンテナーと組み合わせることで、移植可能な標準的な方法でアイデンティティー・ストアを制御できます。

IdentityStoreHandler

`IdentityStore` のインスタンスを管理するには、`IdentityStoreHandler` を使用します。このハンドラーが、使用可能なすべてのアイデンティティー・ストアに対してクエリーを実行するためのメカニズムを提供します。ハンドラー型のインスタンスは、CDI を使用して注入できるようになっています (リスト 3 を参照)。認証を行う必要がある場合は、常に CDI を使用することになります (Java EE Security API での CDI の概要については、第 1 回を参照してください)。

リスト 3. アイデンティティー・ストア・ハンドラーの注入

```
@Inject
private IdentityStoreHandler idStoreHandler;
```

`IdentityStoreHandler` インターフェースには、資格情報インスタンスを受け入れる `validate()` という 1 つのメソッドがあります。通常、このメソッドの実装は、1 つ以上の `IdentityStore` 実装に関連付けられた `validate()` メソッドと `getCallerGroups()` メソッドを呼び出し、これらのメソッド呼び出しの結果を集約して返します。この機能については、このチュートリアルの後のほうで詳しく説明します。

ほとんどのシナリオには、Java EE Security API に付属しているデフォルトの `IdentityStoreHandler` インターフェース実装で十分対応できますが、このデフォルトをカスタム実装で置き換えることもできます。

`IdentityStoreHandler` のデフォルト実装では、複数の `IdentityStore` に対して認証を行います。リストに含まれるストアを繰り返し処理し、`CredentialValidationResult` インスタンスという形に集約した結果を返すという仕組みです。このオブジェクトは非常に単純なものになる場合も、複雑なものにする場合もあります。最も単純な形では、`NOT_VALIDATED`、`INVALID`、または `VALID` のステータス値を返します。ただし多くの場合は、以下の値をさらに組み合わせる必要があります。

- `CallerPrincipal` (呼び出し側のグループを使用する場合もそうでない場合もあります)
- 呼び出し側の名前または LDAP で識別される名前
- アイデンティティー・ストアで一意となっている、呼び出し側の名前

今のところはデフォルトに焦点を絞りますが、この記事の後のほうで、`IdentityStore` インターフェースを実装して独自の軽量のアイデンティティー・ストアをセットアップする方法を説明します。

組み込みアイデンティティー・ストア

Java EE Security API には、それぞれ LDAP と RDBMS を対象とした組み込みの `IdentityStore` 実装が同梱されています。新しい Security API に備わっている他の機能と同じく、組み込みの IdentityStore 実装はアノテーションを使用して簡単に呼び出すことができます。

組み込み RDBMS 統合を呼び出す

外部データベースには、JNDI にバインドされた `DataSource` を使用してアクセスできます。外部データベースをアクティブにするには、`@DataBaseIdentityStoreDefinition` アノテーションを使用します。外部データベースをアクティブにした後、接続の詳細を構成するために、このアノテーションに値を渡します。

組み込み LDAP 統合を呼び出す

LDAP `IdentityStore` Bean を呼び出して構成するには、`@LdapIdentityStoreDefinition` アノテーションを使用します。Bean を呼び出した後、外部 LDAP サーバーに接続するために必要な構成の詳細を渡すことができます。

これらの実装はアプリケーションをスコープにした CDI Bean であり、Java EE 7 ですでに使用可能になっている `@DataStoreDefinition` アノテーションに基づいていることに注意してください。

組み込み RDBMS アイデンティティー・ストアを構成する方法

最も単純な組み込みアイデンティティー・ストアは、データベース・ストアです。データベース・ストアを構成するには、`@DataBaseIdentityStoreDefinition` アノテーションを使用します。リスト 4 に、組み込みデータベース・ストアの構成例を記載します。

リスト 4. RDBMS アイデンティティー・ストアの構成

```
@DatabaseIdentityStoreDefinition(
   dataSourceLookup = "${'java:global/permissions_db'}",
   callerQuery = "#{'select password from caller where name = ?'}",
   groupsQuery = "select group_name from caller_groups where caller_name = ?",
   hashAlgorithm = PasswordHash.class,
   priority = 10
)
@ApplicationScoped
@Named
public class ApplicationConfig { ... }
```

データベース定義を構成したことがあれば、リスト 4 に記載されている構成オプションはお馴染みのはずです。注意すべき 1 つの点として、priority は 10 に設定されている点があります。この値は、複数のアイデンティティー・ストアが実装されている場合に使用します。この設定によって、反復処理の順序が決まります (追って詳細を説明します)。

データベースを構成するために使用できるパラメーターは 9 個あります。これらのパラメーターについては、このリンク先の `DatabaseIdentityStoreDefinition` の Javadoc を参照してください。

組み込み LDAP アイデンティティー・ストアを構成する方法

LDAP の構成には、RDBMS よりも遥かに多くの構成オプションがあります。LDAP 構成のセマンティクスを扱ったことがあれば、これらの構成オプションには馴染みがあるはずです。リスト 5 に、LDAP アイデンティティー・ストアを構成するためのオプションの一部を記載します。

リスト 5. LDAP アイデンティティー・ストアの構成

```
@LdapIdentityStoreDefinition(
   url = "ldap://localhost:33389/",
   callerBaseDn = "ou=caller,dc=jsr375,dc=net",
   groupSearchBase = "ou=group,dc=jsr375,dc=net"
)
@DeclareRoles({ "admin", "user", "demo" })
@WebServlet("/admin")
public class AdminServlet extends HttpServlet { ... }
```

LDAP アイデンティティー・ストアを構成するために使用できる 24 個のパラメーターについては、このリンク先の `LdapIdentityStoreDefinition` の Javadoc を参照してください。

カスタム・アイデンティティー・ストアを開発する

いずれの組み込みアイデンティティー・ストアも要件を満たさない場合は、`IdentityStore` インターフェースを使用してカスタム・ソリューションを開発することもできます。`IdentityStore`

インターフェースには 4 つのメソッドがあり、そのすべてのデフォルトの実装が用意されています。リスト 6 に、これらのメソッドそれぞれのシグニチャーを記載します。

リスト 6. IdentityStore の 4 つのメソッド

```
default CredentialValidationResult validate(Credential)
default Set<String> getCallerGroups(CredentialValidationResult)
default int priority()
default Set<ValidationType> validationTypes()
```

`IdentityStore` インターフェースに含まれるすべてのメソッドは `default` としてマークされているため、どの実装を呼び出すかを指定する必要はありません。デフォルトでは、2 つの主要なメソッドが呼び出されます。3 つ目のメソッドは、複数のアイデンティティー・ストアが構成されている場合に使用されます。

- **`validate()`** は、指定された `Credential` が有効であるかどうかを判断し、その結果を `CredentialValidationResult` として返します。
- **`getCallerGroups()`** は、呼び出し側が関連付けられているグループ名を含む `Set` を返し、それらのグループを、`CredentialValidationResult` インスタンス内にリストアップ済みのグループに集約します。
- **`getPriority()`** は、複数の `IdentityStore` が定義されている場合に使用されます。値が低いほど、優先順位は高くなります。優先順位が同じ場合の動作は定義されていません。
- **`validationTypes()`** は、実装されているメソッド (`validate()` や `getCallerGroups()`) を示す一連の ValidationType を返します。

`validate()` の呼び出しにより、指定された `Credential` が有効であるかどうかが判断されて、その結果が `CredentialValidationResult` で返されます。返された `CredentialValidationResult` インスタンスに対してさまざまなメソッドを呼び出すことで、呼び出し側の LDAP の識別名、一意のアイデンティティー・ストア ID、結果のステータス、アイデンティティー・ストア ID、`Principal`、グループ・メンバーシップの詳細がわかります。

> **注**: 複数の `IdentityStoreHandler` が構成されている場合、`IdentityStoreHandler` の動作を判断するには結果のステータスが重要になります。ステータスの値は、`NOT_VALIDATED`、`INVALID`、`VALID` のいずれかです。

validate() と getCallerGroups() を実装する

`validate()` メソッドと `getCallerGroups()` メソッドは、呼び出し側の `Credential` を検証する場合、呼び出し側のグループ情報を取得する場合にそれぞれ使用します。データ・ストア実装では、いずれか、または両方のメソッドを使用できます。実際に実装するメソッドは、`validationTypes()` メソッドによって宣言します。

この機能を使用すると、あるアイデンティティー・ストアは認証用、別のアイデンティティー・ストアは許可用といったように、アイデンティティー・ストアのタイプを柔軟に指定することができます。`validationTypes()` メソッドが返す一連の `ValidationType` には、`VALIDATE` または `PROVIDE_GROUPS`、あるいはこの両方が含まれる場合があります。`VALIDATE` 定数は `validate()` メソッドが実装されていることを意味し、`PROVIDE_GROUPS` は `getCallerGroups()` メソッドが実装されていることを意味します。両方とも返された場合は、両方のメソッドが実装されていることになります。

> **注**: IdentityStore で状態を維持したり、認証プロセスにおける呼び出し側の現在の進行状況を把握したりすることはすべきでありません。論理的には、アイデンティティー・ストアがユーザーの認証状態をトラッキングしても意味がありません。

複数のアイデンティティー・ストアを処理する

複数の `IdentityStore` 実装を扱う必要がある場合は、`IdentityStoreHandler` を使用します。このハンドラーには `validate()` というメソッドがあります。このメソッドのシグニチャーは、`IdentityStore` 実装での同じ名前のメソッドのシグニチャーと同じです。概念としては、ハンドラーによって、複数のアイデンティティー・ストアが実質的に単一の `IdentityStore` として動作するようにします。

そのために、`IdentityStoreHandler` の `validate()` メソッドは以下のロジックを使用してアイデンティティー・ストアのクエリーを実行します。

1. `validationTypes()` メソッドで宣言されている機能に従って、アイデンティティー・ストアの `validate()` メソッドを呼び出します。メソッドの呼び出し順は、`getPriority()` メソッドによって決定されます。
   - 結果として `VALID` ステータスが返されると、それ以上のアイデンティティー・ストアは調査されません。この場合、ロジックはステップ 2 にジャンプします。
   - ステータスが `INVALID` の場合、後で使用するためにそのステータスが記憶され、`IdentityStoreHandler` が引き続き残りのアイデンティティー・ストアを調査します。
2. `INVALID` ステータスしか返されなければ、`INVALID` を返します。そうでなければ、`NOT_VALIDATED` を返します。
3. 結果として `VALID` が返されて、そのアイデンティティー・ストアが検証タイプとして `PROVIDE_GROUPS` を宣言している場合、`IdentityStoreHandler` は呼び出し側のグループ・メンバーシップを収集し始めます。メンバーシップの収集は、`CredentialValidationResult` オブジェクト内で返された呼び出し側グループを集約することによって行われます。
   - 検証タイプとして `PROVIDE_GROUPS` のみを宣言しているすべての `IdentityStore` を調査するために、`getCallerGroups()` メソッドを呼び出します。返されたグループ名のリストは、累積されたグループのセットに集約されます。
4. すべての `IdentityStore` が調査されると、VALID ステータスと呼び出し側グループのリストを含めた `CredentialValidationResult` が作成されて返されます。

実際の調査

複数のアイデンティティー・ストアを調査しなければならないシナリオを見ていきましょう。このシナリオでは、IdentityStore 1 が RDBMS に接続し、IdentityStore 2 と IdentityStore 3 が LDAP コンテナーに接続します。

図 1 では、アイデンティティー・ストア・ハンドラーが優先順位に従って `IdentityStore` インスタンスを繰り返し処理し、各インスタンスに対して `validate()` メソッドを呼び出します。この処理は、VALID ステータスを返す `CredentialValidationResult` が見つかるまで続けられます。`VALID` ステータスが見つかるのは、IdentityStore 2 を調査した時点です。この時点で、ハンドラーは繰り返し処理を停止し、呼び出し側のグループを収集するために 2 回目の繰り返し処理を開始します。

図 1. IdentityStoreHandler によりアイデンティティー・ストアの最初の調査

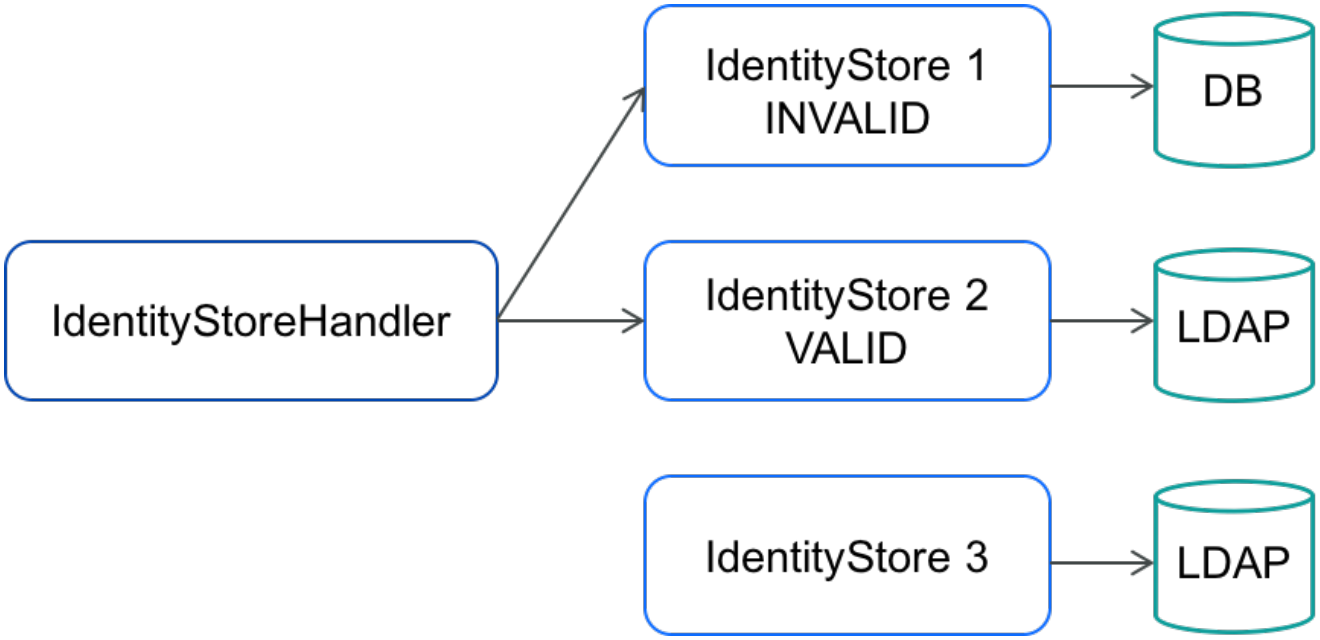
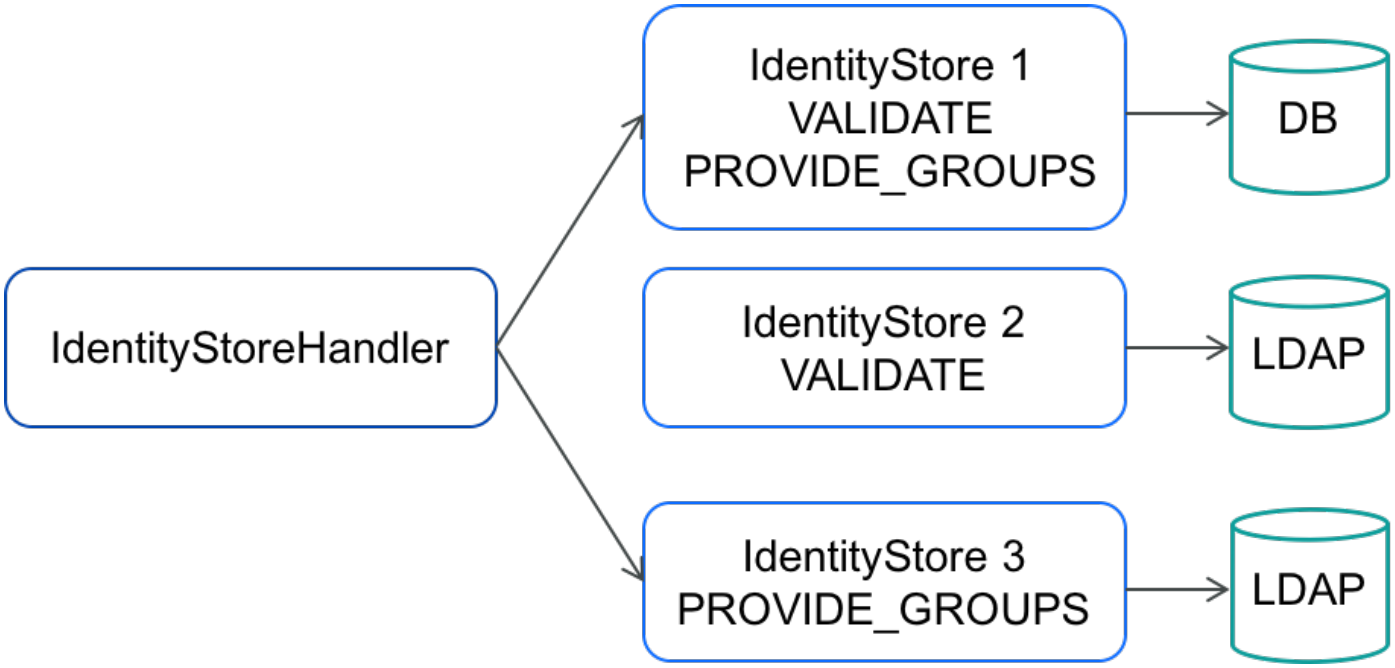


図 2 に、2 回目の調査を示します。ハンドラーは各 `IdentityStore` インスタンスに対して `getCallerGroups()` メソッドを呼び出します。その際、検証タイプとして `PROVIDE_GROUPS` のみを宣言します。

このシナリオで、この指定に一致するアイデンティティー・ストアは IdentityStore 3 だけです。メソッドから返される呼び出し側グループは、IdentityStore 2 から返された `CredentialValidationResult` インスタンスに対する `getCallerGroups()` の呼び出しによって返されたグループ名のセットに結合されます。

図 2. IdentityStoreHandler によりアイデンティティー・ストアの 2 回目の調査



すべての `IdentityStore` の調査が完了すると、`VALID` ステータスと呼び出し側グループのリストを格納した `CredentialValidationResult` が作成されて返されます。

この単純な例は、ある `IdentityStore` によって呼び出し側を調査し、別の IdentityStore によってグループ・メンバーシップリストを作成できる仕組みを明らかにしています。

Cookie を使用した資格情報

第 2 回で説明した `HttpAuthenticationMechanism` でおわかりのように、Cookie を使用すればカスタム `IdentityStore` ソリューションをごく簡単に開発できます。`RememberMeIdentityStore` は `IdentityStore` と同様のインターフェースですが、認証メカニズムではなく、`@RememberMe` アノテーションをサポートするインターセプター・バインディングによって使用されるように意図されています。

`RememberMeIdentityStore` を使用する目的は以下のとおりです。

- 呼び出し側の「ログイン維持」ログイン・トークンを生成する
- 「ログイン維持」ログイン・トークンに関連付けられている呼び出し側を記憶する
- 呼び出し側が戻った場合にログイン・トークンを検証し、追加の資格情報を要求することなく呼び出し側を再認証する

`validate()` メソッドが渡された `RememberMeCredential` を検証する一方、`generateLoginToken()` メソッドがトークンを特定のグループおよびプリンシパルに関連付けます。呼び出し側のログイン・トークンが見つからない場合、あるいはログイン・トークンの有効期間が切れている場合は、通常の認証が行われます。

第 3 回のまとめ

Java エンタープライズ・アプリケーションに外部呼び出し側の認証・許可メカニズムを統合しやすくするための手段は、これまで長く待ち望まれていました。こうした単純化を実現したのが、`IdentityStore` インターフェースです。`IdentityStore` は、コンテナーとサーバー全体にわたる移植性を確保し、複数のアイデンティティー・ストアを簡単かつシームレスに通信できるようにします。

カスタム・アイデンティティー・ストアを実装する必要がないのであれば、1 つのアノテーションと接続に関するいくつかの詳細だけで、LDAP コンテナーまたは RDBMS を構成できます。Java EE 8 アイデンティティー・サーバーはいずれも組み込み `HttpAuthenticationMechanism` をサポートしているので、LDAP ログインを Web ユーザーに結びつけるのは極めて簡単で、いくつかのアノテーションが必要になるだけです。

このチュートリアル・シリーズの最終回では新しい `SecurityContext` を紹介します。お見逃しなく。

学んだ知識をテストしてください

1. 次のアノテーションのうち、組み込みアイデンティティー・ストアを構成するために使用するのはどれですか？(該当するすべての項目を選択)
   a. `@LdapIdentityStoreDefinition`
   b. `@DatabaseIdentityStoreDefinition`
   c. `@RdbmsIdentityStoreDefinition`
   d. `@DataBaseIdentityStoreDefinition`
   e. `@RememberMeIdentityStoreDefinition`
2. `IdentityStore` インターフェースの次のメソッドのうち、デフォルトの実装があるものはどれですか？
   a. `priority()` と `validationTypes()` のみ。
   b. `priority()` のみ。優先順位が設定されていない場合は、デフォルトで 100 に設定される。
   c. `CredentialValidationResult()`、`priority()`、`validationTypes()` のみ。
   d. 4 つすべてのインターフェース・メソッド。
   e. デフォルトの実装があるインターフェース・メソッドはない。
3. 複数の `IdentityStore` 実装がある場合、`validate()` メソッドの呼び出しで `VALID` が返された場合の `IdentityStoreHandler` のデフォルトの動作はどれですか？
   a. そのアイデンティティー・ストアの 2 回目の繰り返し処理を開始する前に、残りのアイデンティティー・ストアの調査を続ける。
   b. 繰り返し処理を停止し、`CredentialValidationResult` オブジェクトを返して呼び出し側の許可を確認する。
   c. アイデンティティー・ストアの繰り返し処理を再開し、`getCallerGroups()` メソッドを呼び出す。
   d. そのアイデンティティー・ストアに対して `getCallerGroups()` メソッドを呼び出し、`CredentialValidationResult` オブジェクトを作成して返す。
   e. どれも当てはまらない。

4. 次の型のうち、`IdentityStore` インスタンスに対して `getCallerGroups()` メソッドを呼び出すと返されるものはどれですか？
    a. `List<###>`
    b. `Set<###>`
    c. `Map<#####, ###>`
    d. `Set<####>`
    e. `List<####>`
5. 次の記述のうち、`RememberMeIdentityStore` に当てはまるものはどれですか？
    a. `RememberMeIdentityStore` は `IdentityStore` を拡張する。
    b. `@RememberMe` アノテーションをサポートするインターセプター・バインディングによって使用されるよう意図されている。
    c. 追加の資格情報を要求せずに呼び出し側を再認証するために使用できる。
    d. 3 つの組み込み IdentityStore 型のうちの 1 つである。
    e. 「ログイン維持」ログイン・トークンの有効期間が切れた場合、通常の認証が行われる。

このリンク先のページで正解を確認してください。

関連トピック

- Java EE Security API 仕様
- Java EE Security API 仕様の GitHub ページ
- Soteria 参照実装
- Java EE Security API 実装
- Devoxx 2017 での、新しい Java Security API の暫定版のプレゼンテーション
- Alex の著書: Java EE 8: Only What's New