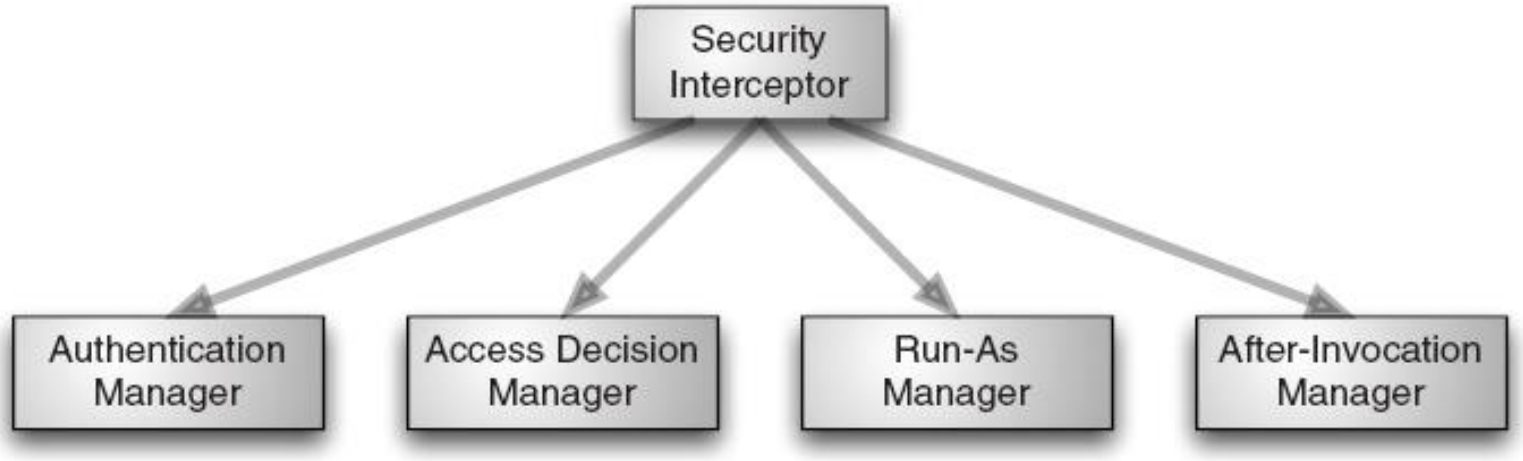
Spring Security

Method Seurity

Introduction

- Spring Security is a security framework that provides declarative security for your Spring-based applications.
- Spring Security provides a comprehensive security solution, handling authentication and authorization, at both the web request level and at the method invocation level.
- Based on the Spring Framework, Spring Security takes full advantage of dependency injection (DI) and aspect oriented techniques

- Spring Security can also enforce security at a lower level by securing method invocations.
- When securing methods, Spring Security uses Spring AOP to proxy objects, applying aspects that ensure that the user has proper authority to invoke the secured methods.
- Spring Security employs five core components to enforce security



Security interceptors

- The security interceptor can be thought of as a latch that prevents you from accessing a secured resource in your application
- The actual implementation of a security interceptor will depend on what resource is being secured.
- If you're securing a URL in a web application, the security interceptor will be implemented as a servlet filter.
- But if you're securing a method invocation, aspects will be used to enforce security
- It does not actually apply security rules. Instead, it delegates that responsibility to the various managers that are pictured at the bottom of the figure we have seen

Authentication managers

- The first gate of the Security Interceptors
- The authentication manager is responsible for determining who you are.
- It does this by considering your *principal* (typically a sername) and your *credentials* (typically a password).
- As with the rest of Spring Security (and Spring itself), the authentication manager is a pluggable interface-based component.
- This makes it possible to use Spring Security with virtually any authentication mechanism you can imagine

Access decisions managers

- The second gate of the security Interceptors
- The access decision manager performs authorization, deciding whether to let you in by considering your authentication information and the security attributes that have been associated with the secured resource.
- Just as with the authentication manager, the access decision manager is pluggable.

Run-as managers

- You may be granted the rights to view a web page, but the objects that are used to create that page may have different security requirements than the web page.
- A run-as manager can be used to replace your authentication with an authentication that allows you access to the secured objects that are deeper in your application.
- Note that not all applications have a need for identity substitution.
- Therefore, run-as managers are an optional security component and are not necessary in many applications secured by Spring Security

After-invocation managers

Spring Security

Authenticating users

Authenticating Users

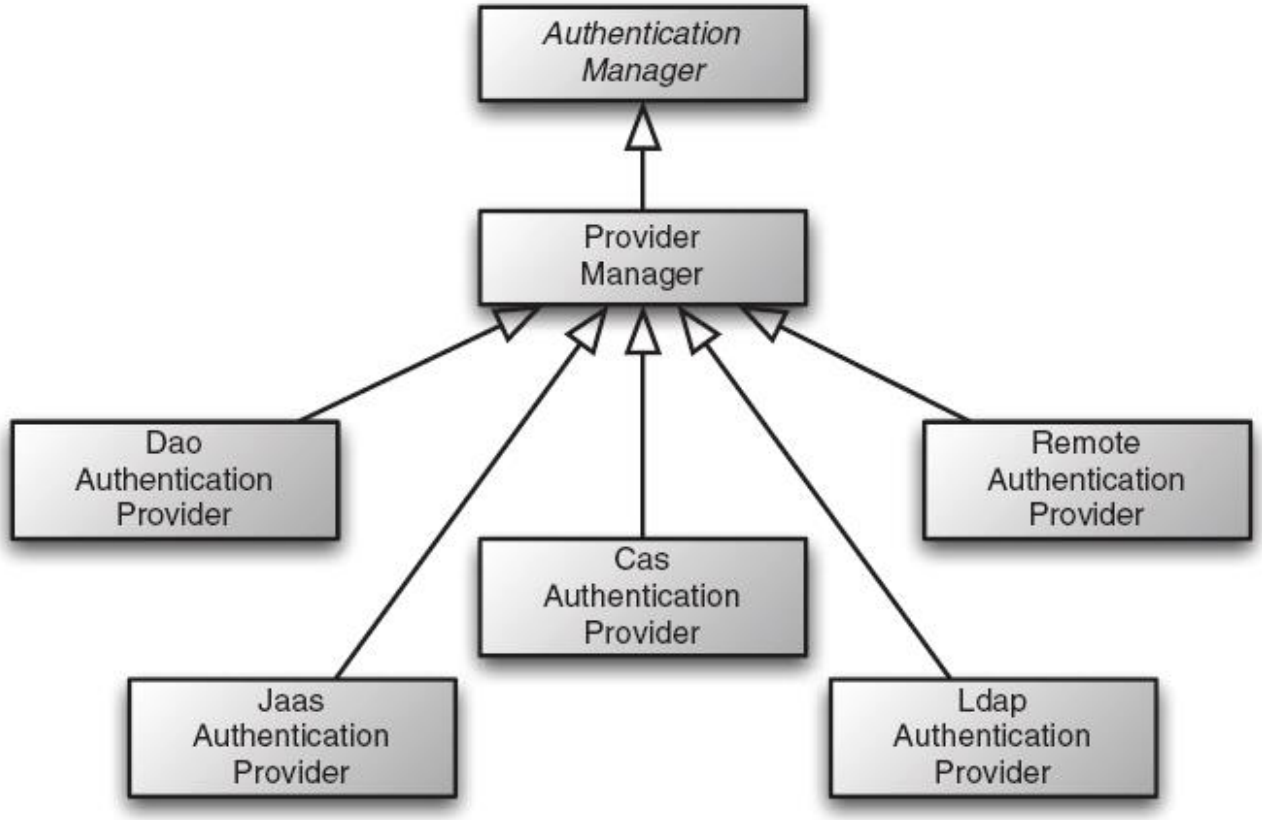
- In Spring Security, the authentication manager assumes the job of establishing a user's identity.
- An authentication manager is defined by the **org.acegisecurity.AuthenticationManager** interface

```
public interface AuthenticationManager {
public Authentication
authenticate(Authentication authentication)
throws AuthenticationException;
}
```

- Spring Security comes with ProviderManager, an implementation of AuthenticationManager that is suitable for most situations

Configuring a provider manager

- ProviderManager is an authentication manager implementation that delegates responsibility for authentication to one or more authentication providers



XML Configuration of Provider Manager

```
<bean id="authenticationManager"
    class="org.acegisecurity.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref bean="daoAuthenticationProvider"/>
      <ref bean="ldapAuthenticationProvider"/>
    </list>
  </property>
</bean>
```

- `ProviderManager` is given its list of authentication providers through its `providers` property.

- Typically, you'll only need one authentication provider, but in some cases, it may be useful to supply a list of several providers so that if authentication fails against one provider, another provider will be tried

ProviderManagers(contd.)

- Spring Security comes with authentication providers for every occasion

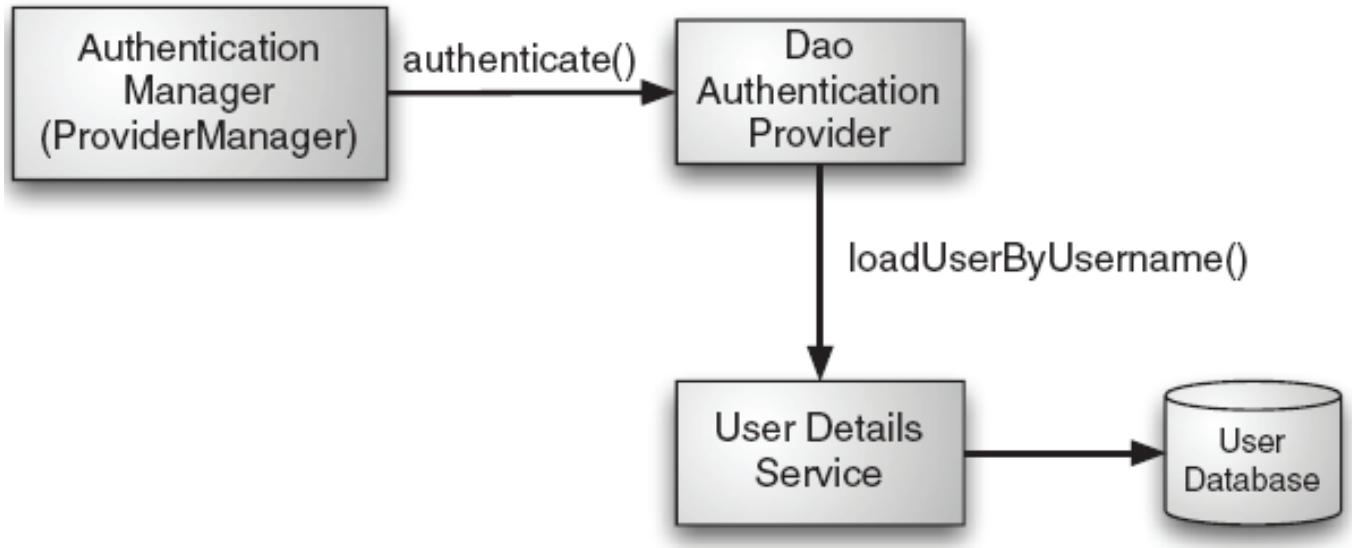
Authentication provider (`org.acegisecurity.*`)	Purpose
`adapters.AuthByAdapterProvider`	Authentication using container adapters. This makes it possible to authenticate against users created within the web container (e.g., Tomcat, JBoss, Jetty, Resin, etc.).
`providers.anonymous.AnonymousAuthenticationProvider`	Authenticates a user as an anonymous user. Useful when a user token is needed, even when the user hasn't logged in yet.
`providers.cas.CasAuthentication-Provider`	Authentication against the JA-SIG Central Authentication Service (CAS). Useful when you need single sign-on capabilities.
`providers.dao.DaoAuthentication-Provider`	Retrieving user information, including username and password from a database.
`providers.dao.LdapAuthenticationProvider`	Authentication against a Lightweight Directory Access Protocol (LDAP) server.

ProviderManagers

Authentication provider (`org.acegisecurity.*`)	Purpose
`providers.jaas.JaasAuthenticationProvider`	Retrieving user information from a JAAS login configuration.
`providers.rememberme.RememberMeAuthenticationProvider`	Authenticates a user that was previously authenticated and remembered. Makes it possible to automatically log in a user without prompting for username and password.
`providers.rcp.RemoteAuthenticationProvider`	Authentication against a remote service.
`providers.TestingAuthenticationProvider`	Unit testing. Automatically considers a `TestingAuthenticationToken` as valid. Not for production use.
`providers.x509.X509AuthenticationProvider`	Authentication using an X.509 certificate. Useful for authenticating users that are, in fact, other applications (such as a web-service client).
`runas.RunAsImplAuthenticationProvider`	Authenticating a user who has had their identity substituted by a run-as manager.

Authenticating against a database

- A DaoAuthenticationProvider is a simple authentication provider that uses a Data Access Object (DAO) to retrieve user information (including the user's password) from a relational database.



Authenticating against a database

- Configuring a DaoAuthenticationProvider couldn't be simpler.
- The following XML excerpt shows how to declare a DaoAuthenticationProvider bean and wire it with a reference to its DAO

```
<bean id="authenticationProvider"
class="org.acegisecurity.providers.dao.
                                    DaoAuthenticationProvider">
<property name="userDetailsService"
ref="userDetailsService"/>
</bean>
```

- The userDetailsService property is used to identify the bean that will be used to retrieve user information from the database.
- This property expects an instance of org.acegisecurity.userdetails.UserDetailsService

Authenticating against a database

- You may not need to implement a UserDetailsService as Spring Security comes with two ready-made implementations of Authentication-Dao to choose from:
  - InMemoryDaoImpl
  - JdbcDaoImpl

Using an in-memory DAO (InMemoryDaoImpl)

- Spring Security comes with InMemoryDaoImpl, an implementation of UserDetailsService that draws its user information from its Spring configuration.
- Here's an example of how you may configure an InMemoryDaoImpl in the Spring configuration file

```
<bean id="authenticationDao"
    class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
  <property name="userMap">
    <value>
      palmerd=4moreyears,disabled,ROLE_PRESIDENT
      bauerj=ineedsleep,ROLE_FIELD_OPS
      obrianc=nosmile,ROLE_SR_ANALYST,ROLE_OPS
      myersn=traitor,disabled,ROLE_CENTRAL_OPS
    </value>
  </property>
</bean>
```

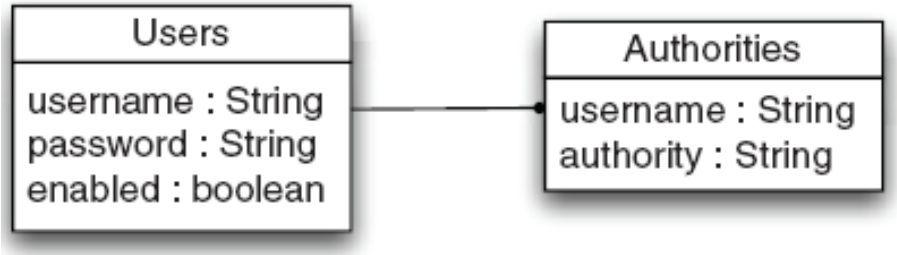
Declaring a JDBC DAO(JdbcDaoImpl)

- JdbcDaoImpl is a simple, yet flexible, authentication DAO that retrieves user information from a relational database.
- In its simplest form, all it needs is a reference to a javax.sql.DataSource, and it can be declared in the Spring configuration file

```
<bean id="authenticationDao"
    class="org.acegisecurity.userdetails.jdbc.JdbcDaoImpl">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

JdbcDaoImple and User Table schema

- JdbcDaoImpl makes some basic assumptions about how user information is stored in the database.
- Specifically, it assumes a Users table and an Authorities table



JdbcDaoImple and Query for Users and Authorities

- When JdbcDaoImpl looks up user information, it will query with the following SQL:

```
SELECT username, password, enabled FROM
users
WHERE username = ?
```

- Likewise, when looking up a user's granted authorities, JdbcDaoImpl will use the following SQL:

```
SELECT username, authority
FROM authorities
WHERE username = ?
```

JdbcDaoImple and custom Query for Users

```
<bean id="authenticationDao"
   class="org.acegisecurity.userdetails.jdbc.JdbcDaoImpl">

<property name="dataSource" ref bean="dataSource" />

<property name="usersByUsernameQuery">

<value>

   SELECT email as username, password, enabled

   FROM Motorist

   WHERE email=?

</value>

</property>

</bean>
```

JdbcDaoImple and custom Query for Authorities

```
<bean id="authenticationDao"
   class="org.acegisecurity.userdetails.jdbc.JdbcDaoImpl">

<property name="dataSource" ref="dataSource" />

...

<property name="authoritiesByUsernameQuery">

<value>

SELECT email as username, privilege as authority

FROM Motorist_Privileges mp, Motorist m

WHERE mp.motorist_id = m.id

AND m.email=?

</value>

</property>

</bean>
```

Spring Security

Controlling access

(Authorization)

Access Decision Managers

- An *access decision manager* is responsible for deciding whether the user has the proper privileges to access secured resources.
- Access decision managers are defined by the **org.acegisecurity.AccessDecisionManager**

Voting access decisions

- Spring Security's access decision managers are ultimately responsible for determining the access rights for an authenticated user.
- However, they do not arrive at their decision on their own. Instead, they poll one or more objects that vote on whether or not a user is granted access to a secured resource.
- Once all votes are in, the decision manager tallies the votes and arrives at its final decision

Access Decision Manager Implementations

- Spring Security comes with three implementations of AccessDecisionManager

Access decision manager	How it decides to grant/deny access
org.acegisecurity.vote.AffirmativeBased	Allows access if at least one voter votes to grant access
org.acegisecurity.vote.ConsensusBased	Allows access if a consensus of voters vote to grant access
org.acegisecurity.vote.UnanimousBased	Allows access if all voters vote to grant access

```
<bean id="accessDecisionManager"
    class="org.acegisecurity.vote.UnanimousBased">
  <property name="decisionVoters">
    <list>
      <ref bean="roleVoter"/>
    </list>
  </property>
</bean>
```