javax.security.auth.spi

Interface LoginModule

`public interface` **`LoginModule`**

`LoginModule` describes the interface implemented by authentication technology providers. LoginModules are plugged in under applications to provide a particular type of authentication.

While applications write to the `LoginContext` API, authentication technology providers implement the `LoginModule` interface. A `Configuration` specifies the LoginModule(s) to be used with a particular login application. Therefore different LoginModules can be plugged in under the application without requiring any modifications to the application itself.

The `LoginContext` is responsible for reading the `Configuration` and instantiating the appropriate LoginModules. Each `LoginModule` is initialized with a `Subject`, a `CallbackHandler`, shared `LoginModule` state, and LoginModule-specific options. The `Subject` represents the `Subject` currently being authenticated and is updated with relevant Credentials if authentication succeeds. LoginModules use the `CallbackHandler` to communicate with users. The `CallbackHandler` may be used to prompt for usernames and passwords, for example. Note that the `CallbackHandler` may be null. LoginModules which absolutely require a `CallbackHandler` to authenticate the `Subject` may throw a `LoginException`. LoginModules optionally use the shared state to share information or data among themselves.

The LoginModule-specific options represent the options configured for this `LoginModule` by an administrator or user in the login `Configuration`. The options are defined by the `LoginModule` itself and control the behavior within it. For example, a `LoginModule` may define options to support debugging/testing capabilities. Options are defined using a key-value syntax, such as *debug=true*. The `LoginModule` stores the options as a `Map` so that the values may be retrieved using the key. Note that there is no limit to the number of options a `LoginModule` chooses to define.

The calling application sees the authentication process as a single operation. However, the authentication process within the `LoginModule` proceeds in two distinct phases. In the first phase, the LoginModule's `login` method gets invoked by the LoginContext's `login` method. The `login` method for the `LoginModule` then performs the actual authentication (prompt for and verify a password for example) and saves its authentication status as private state information. Once finished, the LoginModule's `login` method either returns `true` (if it succeeded) or `false` (if it should be ignored), or throws a `LoginException` to specify a failure. In the failure case, the `LoginModule` must not retry the authentication or introduce delays. The responsibility of such tasks belongs to the application. If the application attempts to retry the authentication, the LoginModule's `login` method will be called again.

In the second phase, if the LoginContext's overall authentication succeeded (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules succeeded), then the `commit` method for the `LoginModule` gets invoked. The `commit` method for a `LoginModule` checks its privately saved state to see if its own authentication succeeded. If the overall `LoginContext` authentication succeeded and the LoginModule's own authentication succeeded, then the `commit` method associates the relevant Principals (authenticated identities) and Credentials (authentication data such as cryptographic keys) with the `Subject` located within the `LoginModule`.

If the LoginContext's overall authentication failed (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules did not succeed), then the `abort` method for each `LoginModule` gets invoked. In this case, the `LoginModule` removes/destroys any authentication state originally saved.

Logging out a `Subject` involves only one phase. The `LoginContext` invokes the LoginModule's `logout` method. The `logout` method for the `LoginModule` then performs the logout procedures, such as removing Principals or Credentials from the `Subject` or logging session information.

A `LoginModule` implementation must have a constructor with no arguments. This allows classes which load the `LoginModule` to instantiate it.

See Also:

`LoginContext`, `Configuration`

Method Summary

Methods

Modifier and Type	Method and Description
boolean	**abort**() Method to abort the authentication process (phase 2).
boolean	**commit**() Method to commit the authentication process (phase 2).
void	**initialize**(**Subject** subject, **CallbackHandler** callbackHandler, **Map**<**String**,?> sharedState, **Map**<**String**,?> options) Initialize this LoginModule.
boolean	**login**() Method to authenticate a `Subject` (phase 1).
boolean	**logout**() Method which logs out a `Subject`.

Method Detail

initialize

```
void initialize(Subject subject,
                CallbackHandler callbackHandler,
                Map<String,?> sharedState,
                Map<String,?> options)
```

Initialize this LoginModule.

This method is called by the `LoginContext` after this `LoginModule` has been instantiated. The purpose of this method is to initialize this `LoginModule` with the relevant information. If this `LoginModule` does not understand any of the data stored in `sharedState` or `options` parameters, they can be ignored.

Parameters:

`subject` - the `Subject` to be authenticated.

`callbackHandler` - a `CallbackHandler` for communicating with the end user (prompting for usernames and passwords, for example).

`sharedState` - state shared with other configured LoginModules.

`options` - options specified in the login `Configuration` for this particular `LoginModule`.

login

```
boolean login()
                throws LoginException
```

Method to authenticate a `Subject` (phase 1).

The implementation of this method authenticates a `Subject`. For example, it may prompt for `Subject` information such as a username and password and then attempt to verify the password. This method saves the result of the authentication attempt as private state within the LoginModule.

Returns:

true if the authentication succeeded, or false if this `LoginModule` should be ignored.

Throws:

`LoginException` - if the authentication fails

commit

```
boolean commit()
               throws LoginException
```

Method to commit the authentication process (phase 2).

This method is called if the LoginContext's overall authentication succeeded (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules succeeded).

If this LoginModule's own authentication attempt succeeded (checked by retrieving the private state saved by the `login` method), then this method associates relevant Principals and Credentials with the `Subject` located in the `LoginModule`. If this LoginModule's own authentication attempted failed, then this method removes/destroys any state that was originally saved.

Returns:

true if this method succeeded, or false if this `LoginModule` should be ignored.

Throws:

`LoginException` - if the commit fails

abort

```
boolean abort()
              throws LoginException
```

Method to abort the authentication process (phase 2).

This method is called if the LoginContext's overall authentication failed. (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules did not succeed).

If this LoginModule's own authentication attempt succeeded (checked by retrieving the private state saved by the `login` method), then this method cleans up any state that was originally saved.

Returns:

true if this method succeeded, or false if this `LoginModule` should be ignored.

Throws:

`LoginException` - if the abort fails

logout

```
boolean logout()
               throws LoginException
```

Method which logs out a `Subject`.

An implementation of this method might remove/destroy a Subject's Principals and Credentials.

Returns:

true if this method succeeded, or false if this `LoginModule` should be ignored.

Throws:

`LoginException` - if the logout fails

Submit a bug or feature

For further API reference and developer documentation, see Java SE Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.