javax.security.auth.login

Class Configuration

java.lang.Object
    javax.security.auth.login.Configuration

```
public abstract class Configuration
extends Object
```

A Configuration object is responsible for specifying which LoginModules should be used for a particular application, and in what order the LoginModules should be invoked.

A login configuration contains the following information. Note that this example only represents the default syntax for the `Configuration`. Subclass implementations of this class may implement alternative syntaxes and may retrieve the `Configuration` from any source such as files, databases, or servers.

```
Name {
      ModuleClass  Flag    ModuleOptions;
      ModuleClass  Flag    ModuleOptions;
      ModuleClass  Flag    ModuleOptions;
};
Name {
      ModuleClass  Flag    ModuleOptions;
      ModuleClass  Flag    ModuleOptions;
};
other {
      ModuleClass  Flag    ModuleOptions;
      ModuleClass  Flag    ModuleOptions;
};
```

Each entry in the `Configuration` is indexed via an application name, *Name*, and contains a list of LoginModules configured for that application. Each `LoginModule` is specified via its fully qualified class name. Authentication proceeds down the module list in the exact order specified. If an application does not have specific entry, it defaults to the specific entry for "*other*".

The *Flag* value controls the overall behavior as authentication proceeds down the stack. The following represents a description of the valid values for *Flag* and their respective semantics:

```
1) Required     - The LoginModule is required to succeed.
                If it succeeds or fails, authentication still continues
                to proceed down the LoginModule list.

2) Requisite    - The LoginModule is required to succeed.
                If it succeeds, authentication continues down the
                LoginModule list.  If it fails,
                control immediately returns to the application
                (authentication does not proceed down the
                LoginModule list).

3) Sufficient   - The LoginModule is not required to
                succeed.  If it does succeed, control immediately
                returns to the application (authentication does not
                proceed down the LoginModule list).
                If it fails, authentication continues down the
                LoginModule list.
```

```
      4) Optional     - The LoginModule is not required to
                      succeed.  If it succeeds or fails,
                      authentication still continues to proceed down the
                      LoginModule list.
```

The overall authentication succeeds only if all *Required* and *Requisite* LoginModules succeed. If a *Sufficient* `LoginModule` is configured and succeeds, then only the *Required* and *Requisite* LoginModules prior to that *Sufficient* `LoginModule` need to have succeeded for the overall authentication to succeed. If no *Required* or *Requisite* LoginModules are configured for an application, then at least one *Sufficient* or *Optional* `LoginModule` must succeed.

ModuleOptions is a space separated list of `LoginModule`-specific values which are passed directly to the underlying LoginModules. 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. The correct way to specify options in the `Configuration` is by using the following key-value pairing: *debug="true"*. The key and value should be separated by an 'equals' symbol, and the value should be surrounded by double quotes. If a String in the form, ${system.property}, occurs in the value, it will be expanded to the value of the system property. Note that there is no limit to the number of options a `LoginModule` may define.

The following represents an example `Configuration` entry based on the syntax above:

```
Login {
  com.sun.security.auth.module.UnixLoginModule required;
  com.sun.security.auth.module.Krb5LoginModule optional
                useTicketCache="true"
                ticketCache="${user.home}${/}tickets";
};
```

This `Configuration` specifies that an application named, "Login", requires users to first authenticate to the *com.sun.security.auth.module.UnixLoginModule*, which is required to succeed. Even if the *UnixLoginModule* authentication fails, the *com.sun.security.auth.module.Krb5LoginModule* still gets invoked. This helps hide the source of failure. Since the *Krb5LoginModule* is *Optional*, the overall authentication succeeds only if the *UnixLoginModule* (*Required*) succeeds.

Also note that the LoginModule-specific options, *useTicketCache="true"* and *ticketCache=${user.home}${/}tickets"*, are passed to the *Krb5LoginModule*. These options instruct the *Krb5LoginModule* to use the ticket cache at the specified location. The system properties, *user.home* and */* (file.separator), are expanded to their respective values.

There is only one Configuration object installed in the runtime at any given time. A Configuration object can be installed by calling the `setConfiguration` method. The installed Configuration object can be obtained by calling the `getConfiguration` method.

If no Configuration object has been installed in the runtime, a call to `getConfiguration` installs an instance of the default Configuration implementation (a default subclass implementation of this abstract class). The default Configuration implementation can be changed by setting the value of the "login.configuration.provider" security property (in the Java security properties file) to the fully qualified name of the desired Configuration subclass implementation. The Java security properties file is located in the file named <JAVA_HOME>/lib/security/java.security. <JAVA_HOME> refers to the value of the java.home system property, and specifies the directory where the JRE is installed.

Application code can directly subclass Configuration to provide a custom implementation. In addition, an instance of a Configuration object can be constructed by invoking one of the `getInstance` factory methods with a standard type. The default policy type is "JavaLoginConfig". See the Configuration section in the Java Cryptography Architecture Standard Algorithm Name Documentation for a list of standard Configuration types.

See Also:

`LoginContext`

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
`static interface`	**`Configuration.Parameters`** This represents a marker interface for Configuration parameters.

Constructor Summary

Constructors

Modifier	Constructor and Description
protected	**Configuration**() Sole constructor.

Method Summary

Methods

Modifier and Type	Method and Description
abstract **AppConfigurationEntry**[]	**getAppConfigurationEntry**(**String** name) Retrieve the AppConfigurationEntries for the specified *name* from this Configuration.
static **Configuration**	**getConfiguration**() Get the installed login Configuration.
static **Configuration**	**getInstance**(**String** type, **Configuration.Parameters** params) Returns a Configuration object of the specified type.
static **Configuration**	**getInstance**(**String** type, **Configuration.Parameters** params, **Provider** provider) Returns a Configuration object of the specified type.
static **Configuration**	**getInstance**(**String** type, **Configuration.Parameters** params, **String** provider) Returns a Configuration object of the specified type.
Configuration.Parameters	**getParameters**() Return Configuration parameters.
Provider	**getProvider**() Return the Provider of this Configuration.
String	**getType**() Return the type of this Configuration.
void	**refresh**() Refresh and reload the Configuration.
static void	**setConfiguration**(**Configuration** configuration) Set the login Configuration.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Configuration

```
protected Configuration()
```

Sole constructor. (For invocation by subclass constructors, typically implicit.)

Method Detail

getConfiguration

```
public static Configuration getConfiguration()
```

Get the installed login Configuration.

Returns:

the login Configuration. If a Configuration object was set via the `Configuration.setConfiguration` method, then that object is returned. Otherwise, a default Configuration object is returned.

Throws:

`SecurityException` - if the caller does not have permission to retrieve the Configuration.

See Also:

`setConfiguration(javax.security.auth.login.Configuration)`

setConfiguration

```
public static void setConfiguration(Configuration configuration)
```

Set the login `Configuration`.

Parameters:

`configuration` - the new `Configuration`

Throws:

`SecurityException` - if the current thread does not have Permission to set the `Configuration`.

See Also:

`getConfiguration()`

getInstance

```
public static Configuration getInstance(String type,
                        Configuration.Parameters params)
                                 throws NoSuchAlgorithmException
```

Returns a Configuration object of the specified type.

This method traverses the list of registered security providers, starting with the most preferred Provider. A new Configuration object encapsulating the ConfigurationSpi implementation from the first Provider that supports the specified type is returned.

Note that the list of registered providers may be retrieved via the `Security.getProviders()` method.

Parameters:

`type` - the specified Configuration type. See the Configuration section in the Java Cryptography Architecture Standard Algorithm Name Documentation for a list of standard Configuration types.

`params` - parameters for the Configuration, which may be null.

Returns:

the new Configuration object.

Throws:

`SecurityException` - if the caller does not have permission to get a Configuration instance for the specified type.

`NullPointerException` - if the specified type is null.

`IllegalArgumentException` - if the specified parameters are not understood by the ConfigurationSpi implementation from the selected Provider.

`NoSuchAlgorithmException` - if no Provider supports a ConfigurationSpi implementation for the specified type.

Since:

1.6

See Also:

`Provider`

getInstance

```
public static Configuration getInstance(String type,
                        Configuration.Parameters params,
                        String provider)
                                 throws NoSuchProviderException,
                                        NoSuchAlgorithmException
```

Returns a Configuration object of the specified type.

A new Configuration object encapsulating the ConfigurationSpi implementation from the specified provider is returned. The specified provider must be registered in the provider list.

Note that the list of registered providers may be retrieved via the `Security.getProviders()` method.

Parameters:

`type` - the specified Configuration type. See the Configuration section in the Java Cryptography Architecture Standard Algorithm Name Documentation for a list of standard Configuration types.

`params` - parameters for the Configuration, which may be null.

`provider` - the provider.

Returns:

the new Configuration object.

Throws:

`SecurityException` - if the caller does not have permission to get a Configuration instance for the specified type.

`NullPointerException` - if the specified type is null.

`IllegalArgumentException` - if the specified provider is null or empty, or if the specified parameters are not understood by the ConfigurationSpi implementation from the specified provider.

`NoSuchProviderException` - if the specified provider is not registered in the security provider list.

`NoSuchAlgorithmException` - if the specified provider does not support a ConfigurationSpi implementation for the specified type.

Since:

1.6

See Also:

`Provider`

getInstance

```
public static Configuration getInstance(String type,
                        Configuration.Parameters params,
                        Provider provider)
                                 throws NoSuchAlgorithmException
```

Returns a Configuration object of the specified type.

A new Configuration object encapsulating the ConfigurationSpi implementation from the specified Provider object is returned. Note that the specified Provider object does not have to be registered in the provider list.

Parameters:

`type` - the specified Configuration type. See the Configuration section in the Java Cryptography Architecture Standard Algorithm Name Documentation for a list of standard Configuration types.

`params` - parameters for the Configuration, which may be null.

`provider` - the Provider.

Returns:

the new Configuration object.

Throws:

`SecurityException` - if the caller does not have permission to get a Configuration instance for the specified type.

`NullPointerException` - if the specified type is null.

`IllegalArgumentException` - if the specified Provider is null, or if the specified parameters are not understood by the ConfigurationSpi implementation from the specified Provider.

`NoSuchAlgorithmException` - if the specified Provider does not support a ConfigurationSpi implementation for the specified type.

Since:

1.6

See Also:

`Provider`

getProvider

```
public Provider getProvider()
```

Return the Provider of this Configuration.

This Configuration instance will only have a Provider if it was obtained via a call to `Configuration.getInstance`. Otherwise this method returns null.

Returns:

the Provider of this Configuration, or null.

Since:

1.6

getType

```
public String getType()
```

Return the type of this Configuration.

This Configuration instance will only have a type if it was obtained via a call to `Configuration.getInstance`. Otherwise this method returns null.

Returns:

the type of this Configuration, or null.

Since:

1.6

getParameters

```
public Configuration.Parameters getParameters()
```

Return Configuration parameters.

This Configuration instance will only have parameters if it was obtained via a call to `Configuration.getInstance`. Otherwise this method returns null.

Returns:

Configuration parameters, or null.

Since:

1.6

getAppConfigurationEntry

```
public abstract AppConfigurationEntry[] getAppConfigurationEntry(String name)
```

Retrieve the AppConfigurationEntries for the specified *name* from this Configuration.

Parameters:

`name` - the name used to index the Configuration.

Returns:

an array of AppConfigurationEntries for the specified *name* from this Configuration, or null if there are no entries for the specified *name*

refresh

```
public void refresh()
```

Refresh and reload the Configuration.

This method causes this Configuration object to refresh/reload its contents in an implementation-dependent manner. For example, if this Configuration object stores its entries in a file, calling `refresh` may cause the file to be re-read.

The default implementation of this method does nothing. This method should be overridden if a refresh operation is supported by the implementation.

Throws:

`SecurityException` - if the caller does not have permission to refresh its Configuration.

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.