6.3. Authentication

Table of contents

6.3.1. Overview

In this section, I will describe the authentication features that are offered in Spring Security.

In Spring Security, and only descriptions of the configuration file, it is possible to implement user authentication. As the authentication method that is provided by the Spring Security, DB authentication, LDAP authentication, CAS authentication, JAAS authentication, X509 authentication, but Basic authentication is supported, in the present guidelines will be described only for DB authentication.

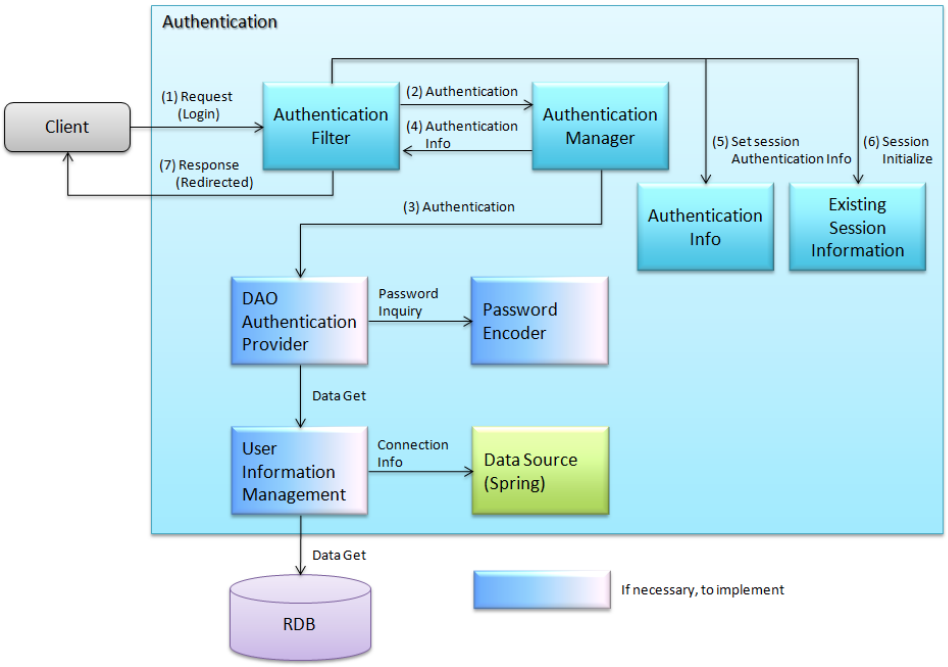
Tip

DB other than authentication details, see the official documentation of each authentication method.

- LDAP Authentication
- CAS Authentication
- Java Authentication and Authorization Service (JAAS) Provider
- X.509 Authentication
- Basic and Digest Authentication

6.3.1.1. Login

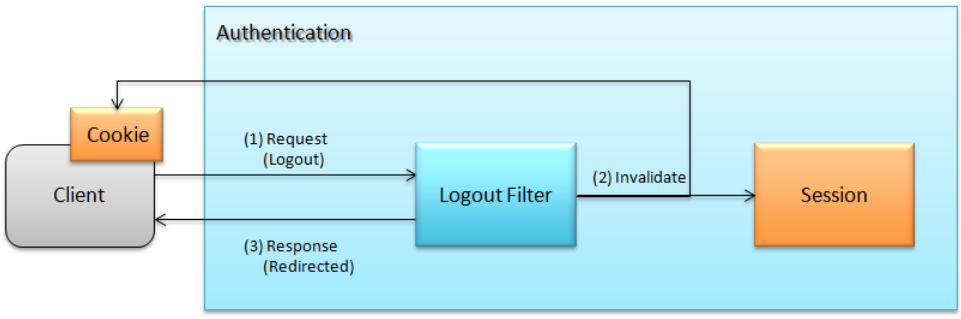
Spring Security I shows the flow of the login process to or below.



1. Upon receiving a request that specifies the authentication process, the authentication filter is activated.
2. Authentication filter, the user, the password is extracted from the request, to generate the authentication information. the generated authentication information is a parameter, and executes the authentication process of the authentication manager.
3. Authentication manager, to perform the authentication process of the specified authentication provider. Authentication provider obtains user information from the data source (DB and LDAP), performs user authentication, such as a password verification. At the time of authentication success, create an authentication information stored authenticated information, and returns the authentication manager. If authentication fails, and sends the authentication failure exception.
4. Authentication manager, returns the authentication information received in the authentication filter.
5. Authentication filter stores the received authentication information (authenticated) to the session.
6. During authentication success, the previous authentication session information to initialize, may want to create new session information.
7. I redirect to the specified authentication success / failure of the path. I plan to return the session ID to the client.

6.3.1.2. Logout

I shown in the following the flow of the logout process by Spring Security.



1. Upon receiving a request for a specified logout processing, logout filter is activated.
2. Log out filter discards the session information. Also, I want to set the response like to discard the client cookie (Cookie in the figure).
3. To the specified log out when the path, I want to redirect.

> **Note**
>
> After logout, session information remaining is to prevent spoofing by be utilized by a third party, the session information, the logout `org.springframework.security.web.session.ConcurrentSessionFilter` is discarded.

6.3.2. How to use

In order to use the authentication function, shown below describing contents in the configuration file of Spring Security.
For basic settings, *Spring Security Overview* see.

. 6.3.2.1 **<sec: http>** element set of

As shown in the following configuration example, the spring-security.xml <http> element auto-config attribute true it is possible to be, The basic configuration of the authentication function of Spring Security, it can be omitted.

```
<Beans  xmlns = "Http://Www.Springframework.Org/schema/beans"
    xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xmlns: sec = "http: // www .Springframework.Org / schema / Security "
    xmlns: context = "Http://Www.Springframework.Org/schema/context"
    xsi: schemaLocation = "Http://Www.Springframework.Org/schema/security
        http: // Www.Springframework.Org/schema/security/spring-security.Xsd
        Http://Www.Springframework.Org/schema/beans
        Http://Www.Springframework.Org/schema/beans/spring-beans.Xsd
        http: / /Www.Springframework.Org/schema/context
        Http://Www.Springframework.Org/schema/context/spring-context.Xsd " >
    <sec: http  auto-config = "true"  use-expressions = "true" >   <- - (1)!>
      <- OMITTED -!>
    </ sec: http>
</ beans>
```

No.	Description
(1)	auto-config = "true" and it is possible to set, Form-Login> , <http-Basic> , <logout> it is enabled even if you do not set the element.

Note

Form-Login> , <http-Basic> , <logout> I describes the elements.

Element name	Description
<Form-login>	org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter is enabled. UsernamePasswordAuthenticationFilter a user name, POST passwords sometimes taken out from the request, a Filter that performs authentication. For more information, *: set of <sec form-login> element* see.
<Http-basic>	org.springframework.security.web.authentication.www.BasicAuthenticationFilter is enabled. BasicAuthenticationFilter is a Filter for carrying out the process of the Basic authentication is implemented in accordance with RFC1945. Detailed use method, BasicAuthenticationFilter JavaDoc see.
<Logout>	Org.Springframework.Security.Web.Authentication.Logout.LogoutFilter , org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler is enabled. LogoutFilter is a Filter called on logout, Org.Springframework.Security.Web.Authentication.Rememberme.TiokeienBasedRememberMeServices (Delete Cookie) and, SecurityContextLogoutHandler I'm calling a (disabling session). For more information, *: set of <sec logout> element* see.

. 6.3.2.2 **<sec: form-Login>** element set of

In this section, **:** <sec form-Login> to explain the elements method of setting.

the attributes of the form-login element, I show below.

spring-security.xml

```
<Beans  xmlns = "Http://Www.Springframework.Org/schema/beans"
    xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xmlns: sec = "http: // www .Springframework.Org / schema / Security "
    xmlns: context = "Http://Www.Springframework.Org/schema/context"
    xsi: schemaLocation = "Http://Www.Springframework.Org/schema/security
        http: // Www.Springframework.Org/schema/security/spring-security.Xsd
        Http://Www.Springframework.Org/schema/beans
        Http://Www.Springframework.Org/schema/beans/spring-beans.Xsd
        http: / /Www.Springframework.Org/schema/context
        Http://Www.Springframework.Org/schema/context/spring-context.Xsd " >
  <sec: http  auto-config = "true"  use-expressions = "true" >
    <sec: form-Login  Login-page = "/ Login"
        default-target-url = "/"
        Login-processing-url = "/ authentication"
        Always-use-default-target = "false"
        authentication-failure-url = "? / Login error = true"
        authentication-failure-handler-ref = "EiyutihenticationFailureHandler"
        authentication-success-handler-ref = "EiyutihenticationSuccessHandler"  />  <-! in the attributes of
the specified order (1) to (7) - >
  </ sec: http>
```

```
</ beans>
```

No.	Description
(1)	`login-page` I specify the path to the login form screen to attribute. If not specified, "/ spring_security_login" becomes the default path, a login screen that Spring Security is available used. When "unauthenticated user" has access to only inaccessible page "authenticated user", it is redirected to this path. **In this guideline, do not use the above default value "/ spring_security_login", it is recommended that you change the system own values.** In this example I have to specify the "/ login".
(2)	`default-target-url` I specify the transition destination path at the time of authentication success to attribute. If not specified, "/" becomes the default path. `authentication-success-handler-ref` if you have an attribute specified, this setting is not used.
(3)	`login-processing-url` I specify the path to perform the authentication process to attribute. If not specified, "/ j_spring_security_check" becomes the default path. **In this guideline, do not use the above default value "/ j_spring_security_check", it is recommended that you change the system own values.** In this example I have to specify the "/ authentication".
(4)	`always-use-default-target` to attribute, after a successful login `default-target-url` to set whether always a transition to the path that you specified. `true` if is specified, `default-target-url` always transition to the path that you specified. `false` if (default) is specified, "the path for displaying a protected page you are trying to access a pre-log" or " `default-target-url` transition to any of the path specified in ". `authentication-success-handler-ref` if you have an attribute specified, this setting is not used.
(5)	`authentication-failure-url` I want to set the authentication failure of transition destination. If not specified, `Login-page` path specified in the attribute applies. `authentication-failure-handler-ref` if you have an attribute specified, this setting is not used.
(6)	`authentication-failure-handler-ref` to attribute called at the time of authentication failure, I specify the handler class. For more information, *setting the handler class at the time of authentication error* see.
(7)	`authentication-success-handler-ref` called at the time of authentication success to attribute, I specify the handler class.

The attributes other than the above, Spring Security documentation see.

> Warning
>
> **The default value for the Spring Security "/ spring_security_login, / j_spring_security_check" reason not to recommend the use of**
>
> If you are using the default value, the application is, about what you are using Spring Security, and lead to discovery. Therefore, if the Spring Security vulnerabilities have been discovered, the risk of attack that with the vulnerability increases. To avoid the risk described above also, it is recommended that you do not use the default value.

6.3.2.2.1. Creating a Login form

A login form to be used at the time of authentication you create in JSP.

- src / main / webapp / WEB-INF / views / login.jsp

```
<Form: form  Action = "$ {PageContext.Request.ContextPath} / authentication"  method = "POST" > <-! (1)
->
    <-! OMITTED ->
    <input  type = "text"  id = "username"  name = "j_username" > <-! (2) ->
    <input  type = "password"  id = "password"  name = "j_password" > <-! (5) ->
    <input  type = "Submit"  value = "Login" >
</ form: form>
```

No.	Description
(1)	Specifies the transition destination for performing the authentication process on the action attribute of the form. Transition destination path is specified in the login-processing-url attribute, specifying / authentication. $ {PageContext.request.contextPath} / authentication process by accessing the authentication is performed. HTTP method is to specify the "POST".
(4)	In the authentication process, the elements to be treated as the "user ID". The name attribute, it can specify a default value of Spring Security "j_username".

(5)	In the authentication process, the elements are treated as "password". The name attribute, it can specify a default value of Spring Security "j_password".

The following are additional to the case of displaying an authentication error message

```
<C: if  test = "$ {Param.Error}" > <- (1) -!>
    <t: MessagesPanel
        MessagesAttributeName = "SPRING_SECURITY_LAST_EXCEPTION" /> <-! (2) ->
</ c: if>
```

No.	Description
(1)	I make a determination of the error message that is set in the request parameters. value and that is set to authentication-failure-url attribute of the form-login element, The values set in the "defaultFailureUrl" authentication error handler, note it is necessary to change the determination process that. In this example, when there is authentication-failure-url = settings such as "/ login? Error = true", and an example is shown.
(2)	I output the exception message to be output at the time of authentication error. Provides a common library `org.terasoluna.gfw.web.message.MessagesPanelTag` it is recommended that it be output by specifying a. " `: <t messagesPanel>` how to use the "tag, *message management* see.

Note

For settings required when accessing from JSP to exception object of authentication error

Exception object of authentication error, in session scope `"SPRING_SECURITY_LAST_EXCEPTION"` is stored in the attribute name. In order to access an object that is stored from the JSP in the session scope, JSP of `page` directive `session` attribute `true` there is a need to.

- src / main / webapp / WEB-INF / views / common / include.jsp

```
<% @  page  session = "true" %>
```

The default setting of blank project, I'm so as not to be able to access from JSP in session scope. This is because the easy to ensure the session is not used.

- spring-mvc.xml

I define the Controller to display the login form.

```
<Mvc: view-controller  path = "/ Login"  view-name = "Login"  /> <-! (1) ->
```

No.	Description
(1)	Once you "/ login" to be accessed, to define the Controller only want to return the "login" as view name. `InternalResourceViewResolver` src / main / webapp / WEB-INF / views / login.jsp is output by. This simple controller is not required to implement by Java.

Tip

The above configuration is equivalent to the following Controller.

```
AttoController
RequestMapping ( "/ Login" )
public  class  LoginController  {

    RequestMapping
    public  String  index ()  {
        Return  "Login" ;
    }
}
```

If is required simply there is only method of returning a view name only one Controller, `: <mvc view-controller>` may be used.

6.3.2.2.2. Login form of attribute name change

"J_username", "j_password" is the default value of Spring Security. `<Form-login>` by setting the element can be changed to any value.

- spring-security.xml

`username` , `password` attributes

```
<Sec: http  auto-config = "true"  use-expressions = "true" >
  <sec: form-Login
      username-parameter = "username"
      password-parameter = "password"  />  ! <- an attribute of the specified order (1) ~ (2) ->
  <- OMITTED ->!
</ sec: http>
```

No.	Description
(1)	`username-parameter` attribute `username` of the input field of `name` attributes, I have changed to "username".
(2)	`password-parameter` attribute `password` input field of `name` attributes, I have changed to "password".

6.3.2.3. Setting the authentication processing

To set the authentication process in the spring Security, `AuthenticationProvider` and `UserDetailsService` to define.

`AuthenticationProvider` is responsible for the following role.

- If you have successfully authenticated, to return the authentication user information
- If it fails to authenticate, to throw an exception

`UserDetailsService` is responsible for acquiring authenticated user information from the persistent layer.

Each may be used what is provided by default, and may be used by its own extension. Combination is also free.

6.3.2.3.1. **`AuthenticationProvider`** of class setting

`AuthenticationProvider` as the implementation of the provider to perform the authentication DB `org.springframework.security.authentication.dao.DaoAuthenticationProvider` explaining how to use.

- spring-security.xml

```
<Sec: authentication-Manager> <- (1) -!>
    <sec: authentication-provider  user-service-ref = "UserDetailsService" > <- (2) -!>
        <sec: password-encoder  ref = "PasswordEncoder"  /> <- (3) ->!
    </ sec: authentication-provider>
</ sec: authentication-Manager>
```

No.	Description
(1)	`:` `<Sec authentication-manager>` within the element `:` `<sec authentication-provider>` I define the elements. And a plurality specified, although it is possible to combine the authentication method is not described here.
(2)	`<Sec: authentication-provider>` element in `AuthenticationProvider` I to define. By default, `DaoAuthenticationProvider` is enabled. Other `AuthenticationProvider` If you want to specify in the ref attribute, specify the Bean ID of AuthenticationProvider target . `user-service-ref` attribute to, to get the authentication user information `UserDetailsService` I specify the Bean Id of. `DaoAuthenticationProvider` If you want to use, this setting is essential. For more information, *setting of UserDetailsService class* see.
(3)	During password verification, I specify the Bean ID of the class to perform the encoding of the password entered from the form. If they are not specified, the password is handled by the "plain text". For more information, *password hashed* see.

Get the data from the persistent layer only "User ID" and "Password", the if requirement that authentication `DaoAuthenticationProvider` may be use.
Data acquisition method from the persistence layer will be described `UserDetailsService` decide on.

6.3.2.3.2. **`UserDetailsService`** of class setting

`AuthenticationProvider` of `userDetailsService` setting the Bean that you specified in the property.

`UserDetailsService` is an interface with the following method.

```
UserDetails  LoadUserByUsername ( String  username )  throws  UsernameNotFoundException
```

By implementing this interface, it is possible to acquire the authenticated user information from an arbitrary location.

Here, you can use the JDBC, to get the user information from DB
`Org.Springframework.Security.Core.Userdetails.Jdbc.JdbcDaoImpl` to explain the.

`JdbcDaoImpl` may be performed the following Bean defined in the spring-security.xml To use the.

```
<-! Omitted ->

<Bean  id = "UserDetailsService"
  class = "Org.Springframework.Security.Core.Userdetails.Jdbc.JdbcDaoImpl" >
  <property  name = "dataSource"  ref = "dataSource" />
</ bean>
```

`JdbcDaoImpl` , it has defined the default SQL to retrieve the authentication user information and authorization information, it has become the premise that table corresponding to these are provided. Premise to that table definition Spring Security documentation see.
User information from an existing table, if you want to get the authorization information, it may be modified to suit the SQL that is issued to an existing table.
SQL to be used are the following three.

- User information acquisition query

  By creating a table that matches the user information acquisition query, unnecessary query specified in the configuration file to be described later.
  "Username", "password", "enabled" is field is mandatory,
  In query specification of the configuration file will be described later, by applying the alias table name, no problem without column names match.
  For example, by setting the following, such as SQL can be used to "email" column as "username", "enabled" is always `true` becomes.

  ```
  SELECT  email  AS  username ,  pwd  AS  password ,  true  AS  enabled  FROM  Customer  WHERE  email
  =  ?
  ```

  Creating a login form described above, the "user ID" is specified in the parameters of the query.

- User privileges acquisition query

  Is a query that retrieves the authorization information for the user.

- Group rights acquisition query

  is a query that retrieves the authorization information of the group to which the user belongs. Group permissions by default is disabled and is not addressed in this guideline.

Below, I show examples of defining a DB, the configuration file example of Spring Security.

For the definition of the table
As we implement DB authentication process, to define a table required.
Described above, it is the default user information obtaining query matching table.
Therefore, the definition of the table below is the minimum required (physical name is tentative name).

Table name: account

Logical name	Physical name	Type	Description
User ID	username	String	User ID for uniquely identifying the user.
Password	password	String	User password. I be stored in hashed state.
Enable flag	enabled	Truth-value	Invalid user, flags that represent the user. User that has been set to "false" as disabled user, the authentication error.
Authority name	authority	String	Not required if you do not require authorization features.

`JdbcDaoImpl` an example of setting a customized are shown below.

```
<-! Omitted ->

<Bean  id = "UserDetailsService"
  class = "Org.Springframework.Security.Core.Userdetails.Jdbc.JdbcDaoImpl" >
  <property  name = "RolePrefix"  value = "ROLE_"  /> <-! (1) ->
  < property  name = "dataSource"  ref = "dataSource"  />
  <property  name = "EnableGroups"  value = "false"  /> <-! (2) ->
  <property  name = "UsersByUsernameQuery"
    value = "SELECT username, ? password, enabled FROM account WHERE username = "  /> <-! (3) ->
  <property  name = "EiuthoritiesByUsernameQuery"
    value = ? "SELECT username, Authority FROM account WHERE username ="  /> <-! ( 4) ->
</ bean>
```

No.	Description
(1)	I specify a prefix of authority name. In the case of the authority name is "USER", which is stored on the DB, authority name this authentication user object has it become "ROLE_USER".

It is necessary to set to match a naming convention and authorization functions. For more information on the approval function, *authorized* see.

(2)	In authorization features, and can be used to specify the use of the concept of "group rights". It is not addressed in this guideline.
(3)	I set the query to retrieve the user information. Data to be acquired, "User ID", "Password", and then "valid flag". If it is not performed authentication determination by "valid flag", and SELECT result "true" fixed "valid flag". Incidentally, to write queries that can be uniquely retrieve user. In the case where it is more than the number acquired, 1 of the record is used as a user.
(4)	I set the query to get the permission of the user. Data to be acquired, "User ID", and then "authorization ID". If you do not want to use the functionality of the authorization, "authorization ID" may be any fixed value.

Note

If you do the authentication that can not be achieved by simply changing the query, `UserDetailsService` it is necessary to realize by extending. How to extend the *extension of UserDetailsService* see.

6.3.2.4. **`UserDetails`** of class usage

After a successful authentication `UserDetailsService` created `UserDetails` how to use the, will be described.

6.3.2.4.1. In Java class **`UserDetails`** use object

After successful authentication, `UserDetails` class `org.springframework.security.core.context.SecurityContextHolder` is stored in.

`SecurityContextHolder` from `UserDetails` I shows an example to get.

```
public  static  String  getUsername ()  {
    Authentication  authentication  =  SecurityContextHolder . getContext ()
            . getAuthentication ();  // (1)
    if  ( authentication  ! =  null )  {
        Object  principal  =  authentication . getPrincipal ();  // (2)
        if  ( principal  instanceof  UserDetails )  {
            Return  (( UserDetails )  principal .) getUsername ();  // (3)
        }
        Return  ( String )  principal . toString ();
    }
    Return  null ;
}
```

No.	Description
(1)	`SecurityContextHolder` from `org.springframework.security.core.Authentication` I get the object.
(2)	`Authentication` from object `UserDetails` I get the object.
(3)	`UserDetails` from the object, I get a user name.

`SecurityContextHolder` from `UserDetails` how to get the object is available in static method from anywhere, convenient the other hand, would increase the degree of coupling module. Test it is difficult also performed.

`UserDetails` object `AuthenticationPrincipal` be acquired by utilizing.
`AuthenticationPrincipal` in order to use the `org.springframework.security.web.bind.support.AuthenticationPrincipalArgumentResolver` the `: <mvc argument-resolvers>` it is necessary to set in.

- spring-mvc.xml

```
<Mvc: annotation-driven>
    <mvc: argument-Resolvers>
        <bean
            class = "Org.Springframework.Data.Web.PieijiieibieruieichieienudilerMethodArgumentResolver"  />
        <bean
            class =
"Org.Springframework.Security.Web.Bind.Support.EiyutietchiienutiaishieitiaioenuPrincipalArgumentResolver"  />
     </ mvc: argument-Resolvers>
</ mvc: annotation-driven>
```

Spring as follows in the MVC in the Controller `SecurityContextHolder` without using the `UserDetails` I can retrieve the object.

```
RequestMapping ( method  =  RequestMethod . GET )
```

```
public  String  view ( AttoAuthenticationPrincipal  SampleUserDetails  UserDetails ,  // (1)
        Model  model )  {
    // Get account object
    Account  account  =  UserDetails . getAccount ();  // (2)
    model . addAttribute ( account );
    Return  "account / view" ;
}
```

No.	Description
(1)	`AuthenticationPrincipal` to get the user information that you are logged in using.
(2)	`SampleUserDetails` I get the account information from.

Note

`AuthenticationPrincipal` type of argument that annotate the `UserDetails` there needs to be a class that inherits from type. Usually *extension of UserDetailsService* you create in `UserDetails` may be using inheritance class.

`SampleUserDetails` class *Spring Security tutorial* is a class that you create in. For more information about *the creation of authentication services* see.

When accessing UserDetails object in the Controller will be recommended way .

Note

Controller obtains the Service class `UserDetails` uses the information of the object, `SecurityContextHolder` is recommended not to use.

`SecurityContextHolder` is `UserDetails` it is desirable to use only within the method that does not pass the object in the argument.

6.3.2.4.2. In JSP **`UserDetails`** access to

In Spring Security, as a mechanism for using the authentication information in the JSP, it provides a JSP taglib. Following declaration in order to use this taglib is required.

```
<% @  taglib  uri = "Http://Www.Springframework.Org/security/tags"  prefix = "sec" %>
```

Note

Template of TERASOLUNA Server Framework for Java (5.x) is already set to WEB-INF / views / common / include.jsp If you are using.

As an example if you want to display the authentication user name in a JSP, shows how to use.

```
<Sec: authentication  property = "Principal.Username"  /> <-! (1) ->
```

No.	Description
(1)	`<Sec: authentication>` tag in the `Authentication` and access to the object, `property` can be property access specified for the attribute. In this example `getPrincipal (). getUsername ()` I output the results of.

```
<Sec: authentication  property = "principal"  var = "UserDetails"  />  <-! (1) ->
$ {F: h (UserDetails.Username)} <-! (2) ->
```

No.	Description
(1)	`property` a property that is specified in the attribute `var` can be stored in a variable with the name you to attribute.
(2)	(1) After you have stored in the variable: In a JSP `UserDetails` have access to.

Note

In the controller `UserDetails` is to get the `Model` but can also be added to, it is sufficient to use a JSP tag when displaying the JSP.

Note

Setting of UserDetailsService class is described in `JdbcDaoImpl` generates `UserDetails` does not only hold the minimum amount of information such as "user ID" and "authority". If the "user first and last name," such as requiring other user information as a display item of screen `UserDetails` and `UserDetailsService` there is a need to extend the. How to extend the *extension of UserDetailsService* see.

6.3.2.5. The session management in Spring Security

Session information generation scheme and at the time of login, I will describe how to configure the when an exception occurs.
<Session-management> By specifying the tag, it is possible to customize the management of the session.

I show an example of setting the spring-security.xml below.

```
<Sec: http  auto-config = "true"  create-session = "IfRequired"  > <-! (1) ->
  <-! OMITTED ->
  <sec: session-Management
    invalid-session-url = " / "
    session-authentication-error-url = "/"
    session-fixation-Protection = "MigrateSession"  /> <-! in the attributes of the specified order (2) ~ (4)
->
  <-! OMITTED - >
</ sec: http>
```

No.	Description
(1)	<Http> tag create-session in the attribute, I specify the session of creating policy. Can be the following values. • Always : Spring Security, create a new session if there is no existing session, if the session exists, reuse. • IfRequired : (default) Spring Security creates If the session is necessary. If the session is already reuse without creating. • Never : Spring Security is not to create a session, if present session, to be reused. • stateless : Spring Security, you do not want to create a session, you do not want to use be present session. Therefore, it is necessary to perform authentication each time.
(2)	invalid-session-url Attributes, invalid session ID is I specify the path to transition if it is requested (session timeout occurs). If you do not specify, existence check session subsequent processing is invoked without being run. For more information, see " *Detection of session timeout* see ".
(3)	session-authentication-error-url Attributes, Org.Springframework.Security.Web.Authentication.Session.EsuiesuesuionAuthenticationStrategy I specify the path to transition when an exception occurs in. If not specified, is set to "401 Unauthorized" to the response code, an error response is performed. This setting, <form-Login> not used when performing authentication using the tag. SessionAuthenticationStrategy exception that occurred in the, <form-Login> tag authentication-failure-url attribute or authentication-failure-handler-ref is handled in accordance with the definition of the attribute.
(4)	session-fixation-protection to the attributes, I specify a session management system at the time of authentication success. Can be the following values. • none : As it is I want to use the previous login session. • MigrateSession : (default on Servlet 3.0 and earlier container) Discards the previous login session is newly create a new session, to take over the information that has been stored in the previous login session. • ChangeSessionId : (default on Servlet 3.1 and later container) Has been added from the servlet 3.1 javax.servlet.http.HttpServletRequest # ChangeSessionId () to change the session ID using the method. • newsession : Discards the previous login session is newly create a new session, information that has been stored in the previous login session is not over. The purpose of this function, it is possible to allocate a new session ID for each login, session Fixation attacks is to prevent. Therefore, unless there is a clear intention, it is recommended that you use the default value.

6.3.2.5.1. Of session timeout detection

If you want to detect the session timeout, `invalid-session-url` may be specify the path to transition when the session timeout has occurred in the attribute.

`invalid-session-url` If you specify the attributes, `http` element `pattern` for all requests that match the path pattern that you specified in the attribute, the presence of the session check (existence check of the requested session ID) is performed.

Note

If the path and it does not detect the path to detect the session timeout are mixed, `http` there is a need to define multiple elements. `http` If the element to define multiple, it is necessary to note that there are things that set is reduced maintainability would be redundant.

If the setting in order to detect the session timeout is redundant, I want you to consider that you create a custom filter that can be specified in application path or exclusion path. When you create a custom filter, it is preferable to the use or reference to the following classes that are provided by Spring Security.

- `org.springframework.security.web.session.SessionManagementFilter`
  Processing for session existence check (checking the presence of the requested session ID) is implemented.
- `org.springframework.security.web.session.SimpleRedirectInvalidSessionStrategy`
  Processing after detecting the session timeout (invalid session ID) is implemented.
  By default, it is redirected to the specified path after generating the session.
- `org.springframework.security.web.util.matcher.RequestMatcher`
  Is an interface for matching determination of the request, can be used in the determination process of the application path or paths excluded.
  Some useful implementation classes in the same package have been provided.

Note

`<Csrf>` by specifying the elements *CSRF measures* if you are doing, there is a case that session time-out is detected by the CSRF protection function.

Below, I show the conditions under which the session timeout is detected by the CSRF protection function.

- The destination of the CSRF token I have to HTTP session (default).
- CSRF token can not be retrieved from the HTTP session.
- *The request that is to be checked of CSRF token* is.

If the session timeout is detected by the CSRF protection function, it becomes one of the following operations.

- `invalid-session-url` when there is a specified attributes, after generating the session `invalid-session-url` is redirected to the path that you specified.
- `invalid-session-url` if you do not specify the attributes, `<access-denied-handler>` specified in the element `org.springframework.security.web.access.AccessDeniedHandler` handling in accordance with the definition of is performed.

`AccessDeniedHandler` For information on how to define, " *set of spring-security.xml* see ".

6.3.2.5.2. Concurrent Session Control of use setting

In Spring Security, the ability to control the number of sessions that one user can log in at the same time (Concurrent Control Session you are providing).
The user here, `Authentication.GetPrincipal ()` is obtained in refers to the authenticated user object.

Note

This feature application server is implemented session replication by one configuration or session server or cluster, (that is, all the applications are are using the same session area) is effective in the case. If you have configured with multiple units or multiple instances, if the session area exists separately, and be careful because you can not control the concurrent logins in this function.

Control method in the case where it exceeds the maximum number of sessions, there are the following patterns. Be used properly by the business requirements.

1. If you exceed the maximum number of sessions of one user, to disable the user that is not the most used (post-win)
2. If you exceed the maximum number of sessions of one user, it does not accept the new login (earlier wins)

In either case, in order to enable this function, it is necessary to add the following configuration in web.xml.

```
<Listener>
  <listener-class> Org.Springframework.Security.Web.Session.HttpSessionEventPublisher </ listener-class> <-!
(1) ->
</ listener>
```

No.	Description

(1) As we use a concurrent Session Control, `Org.Springframework.Security.Web.Session.HttpSessionEventPublisher` a, it is necessary to define the listener.

. 6.3.2.5.3 **`<sec: concurrency-Control>`** set of

If you want to use the Concurrent Session Control is, **`:`** `<sec session-Management>` as child elements of the element : <sec concurrency-control> to specify the elements.

```
<Sec: http  auto-config = "true"  >
  <sec: session-Management>
      <sec: concurrency-Control
          error-if-maximum-exceeded = "true"
          max-sessions = "2"
          expired-url = "/ ExpiredSessionError .jsp "  /> ! <- an attribute of the specified order (1) ~ (3) -
>
      </ sec: session-Management>
  </ sec: session-Management>
</ sec: http>
```

No.	Attribute name	Description	Default value	Default value Description
(1)	`error-if-maximum-exceeded`	I specify the behavior when a login attempt with that exceed the maximum possible number of login sessions. `true` if you have set, and by generating an authentication error, it does not accept the new login. (Earlier wins)	false	Login becomes possible, not the least used (the oldest last access time) session is invalidated. If a request from a client utilizing the invalidated session occurs, `expired-url` to transition to the URL that is specified by the attribute. (Post wins)
(2)	`max-sessions`	1 I want to specify the maximum number of sessions that can log in as a user. If you set the 2, it is possible to log in two sessions with the same user.	One	The default is only one session
(3)	`expired-url`	URL that transition if the request is generated from clients that are using the invalidated session.	Without	fixed message notifying that a session is invalidated is answered.

Note

If you want to customize the filter (FORM_LOGIN_FILTER) for the authentication process, `<sec: concurrency-Control>` In addition to the specified element, two of the following `SessionAuthenticationStrategy` it is necessary to activate the class.

- `org.springframework.security.web.authentication.session.ConcurrentSessionControlAuthenticationStrategy`
  Class to check the number of sessions per user login after successful authentication.
- `org.springframework.security.web.authentication.session.RegisterSessionAuthenticationStrategy`
  Class to register a session of successful authentication to session management area.

In Spring Security 3.1 is dependent on version 1.0.X.RELEASE, `Org.Springframework.Security.Web.Authentication.Session.ShioenushiyuaruaruentSessionControlStrategy` but class that has been provided, it is deprecated in API from Spring Security 3.2 . If you are upgrading from Spring Security 3.1 after Spring Security 3.2, it is necessary to change to use a combination of the following classes.

- `ShioenushiyuaruaruienutiesuiesuesuaioenushioenutirolAuthenticationStrategy` (added in Spring Security 3.2)
- `Aruijiaiesutiiaruesuiesuesuion AuthenticationStrategy` (added in Spring Security 3.2)
- `org.springframework.security.web.authentication.session.SessionFixationProtectionStrategy`

For specific definition method, Spring Security Reference-Web Application Security (Control Concurrency) - should be a sample code of the reference.

6.3.2.6. Setting the handler class at the time of authentication error

`<Sec: form-login>` element `authentication-failure-handler-ref` to attribute `org.springframework.security.web.authentication.ExceptionMappingAuthenticationFailureHandler` was setting in class, And exceptions to be sent at the time of authentication error, I can specify the transition destination corresponding to it. Transition destination that you specify, it unauthenticated user is accessible.

spring-security.xml

```
<Sec: http  auto-config = "true"  use-expressions = "true" >
```

```
  <sec: form-Login  Login-page = "/ Login"
      authentication-failure-handler-ref = "EiyutihenticationFailureHandler"
      authentication-success-Handler- ref = "EiyutihenticationSuccessHandler"  />
</ sec: http>

<Bean  id = "EiyutihenticationFailureHandler"
class =
"Org.Springframework.Security.Web.Authentication.IekkusushiipitiaioenuemueipipiaienujieiyutihenticationFailure
Handler" >
<property  name = "DefaultFailureUrl"  value = "/ Login / DefaultError"  /> <-! (1) ->
  <property  name = "ExceptionMappings" > <-! (2) ->
    <props>
      <prop  key =
        "Org.Springframework.Security.Authentication.BadCredentialsException" > <-! (3) ->
          / Login / badCredentials
      </ Prop>
      <prop  key =
        "Org.Springframework.Security.Core.Userdetails.UsernameNotFoundException" > <-! (4) ->
          / Login / usernameNotFound
      </ Prop>
      <prop  key =
        "Org.Springframework.Security.Authentication.DisabledException" > <-! (5) ->
          / Login / disabled
      </ Prop>
      <prop  key =
        "Org.Springframework.Security.Authentication.ProviderNotFoundException" > <-! (6) ->
          / Login / providerNotFound
      </ Prop>
      <prop  key =
        "Org.Springframework.Security.Authentication.EiyutietchiinticationServiceException" > <-! (7) ->
          / Login / authenticationService
      </ Prop>
      <-! OMITTED ->
    </ props>
  </ property>
</ bean>
```

No.	Description
(1)	I specify the default transition destination path at the time of error. to be described later `exceptionMappings` If an exception is not defined in properties occurs, a transition to the transition destination specified in this property.
(2)	and exceptions to catch, the transition destination when an exception occurs, I will be specified in a list format. Specify the exception class in key, to set the transition destination in value.

a typical exception that Spring Security throws, are described below.

No.	Types of errors	Description
(3)	`BadCredentialsException`	Thrown by the password verification failure during authentication error.
(4)	`UsernameNotFoundException`	Thrown at the time by the authentication error unauthorized user ID (nonexistent user ID). `org.springframework.security.authentication.dao.AbstractUserDetailsAuthenticationProvider` If a class that inherits from it is specified in the authentication provider, `hideUserNotFoundExceptions` the `false` above exception If you do not change the, `BadCredentialsException` be changed.
(5)	`DisabledException`	At the time of authentication error due to an invalid user ID, and is thrown.
(6)	`ProviderNotFoundException`	Thrown at the time of authentication provider class undetected error. Reasons such as setting error, I will occur if the authentication provider class is illegal.
(7)	`AuthenticationServiceException`	Thrown at the time of authentication service error. DB connection error, etc., I will occur when any error occurs in the authentication service.

Warning

In this example, `UsernameNotFoundException` Although by transitioning by handling, and inform the user that the user ID does not exist, since the presence or absence of a particular ID known, undesirable security perspective. Therefore, the message to notify the user, the screen transition that does not distinguish the type of exception, better to the message.

. 6.3.2.7 **<sec: logout>** element set of

In this section, `<sec: logout>` to explain the method of setting the element.

spring-security.xml

```
<Sec: http  auto-config = "true"  use-expressions = "true" >
  <-! OMITTED ->
  <sec: logout
```

```
        logout-url = "/ logout"
        logout-success-url = "/"
        Invalidate- session = "true"
        delete-cookies = "JSESSIONID"
        success-handler-ref = "LogoutSuccessHandler"
      />  <-! in the attributes of the specified order (1) ~ (5) ->
    <-! OMITTED ->
  </ sec: http>
```

No.	Description
(1)	`logout-url` to attribute, specify the path to run the logout process. If not specified, "/ j_spring_security_logout" becomes the default path. **In this guideline, do not use the above default value "/ j_spring_security_logout", it is recommended that you change the system own values.** In this example I have to specify the "/ logout".
(2)	`logout-success-url` attribute to, I want to specify the transition destination path in after logout. If not specified, "/" is the default path. If you specify this attribute, `success-handler-ref` It is an error at startup when you specify the attributes.
(3)	`invalidate-session` to attribute to set whether to discard the session on logout. The default is `true` is. `true` if the session is destroyed on logout.
(4)	`delete-cookies` to attribute, are listed cookie name to be deleted on logout. If multiple statements separated by ",".
(5)	`success-handler-ref` to attribute, I specify the handler class to call after logout success. If you specify this attribute, `logout-success-url` It is an error at startup when you specify the attributes.

Warning

The reason you do not recommend the use of default values for the Spring Security "/ j_spring_security_logout"

If you are using the default value, the application is, about what you are using Spring Security, and lead to discovery. Therefore, if the Spring Security vulnerabilities have been discovered, the risk of attack that with the vulnerability increases. To avoid the risk described above also, it is recommended that you do not use the default value.

Note

CSRF measures are described in `: <sec csrf>` If you are using, for CSRF token check is made, **and sends the POST logout request, CSRF tokens need to be sent** . Describes how to embed a CSRF token below.

- *How to embed a CSRF token in the automatic*

```
<Form: form  method = "POST"
   Action = "$ {PageContext.Request.ContextPath} / logout" >
  <input  type = "Submit"  value = "Logout"  />
</ form: form>
```

This case is output HTML as follows. CSRF token is set in the hidden.

```
<Form  id = "command"  Action = "/ your-context-path / logout"  method = "POST" >
  <input  type = "Submit"  value = "Logout"  />
  <input  type = "hidden"  name = "_Csrf "  value = "5826038F-0A84-495b-A851-C363e501b73b"  />
</ form>
```

- *How to explicitly embed the CSRF token*

```
<Form   method = "POST"
  Action = "$ {PageContext.Request.ContextPath} / logout" >
  <sec: CsrfInput />
   <input  type = "Submit"  value = "Logout"  />
</ form>
```

Similarly the following such as HTML is output also. CSRF token is set in the hidden.

```
<Form   method = "POST"
  Action = "/ your-context-path / logout" >
  <input  type = "hidden"  name = "_Csrf"  value = "5826038F-0A84-495b-A851-C363e501b73b"  />
  <input  type = "Submit"  value = "Logout"  />
</ form>
```

. 6.3.2.8 **<sec: Remember-me>** element set of

" Remeber Me A ", the convenience for the user to frequently access the website, and as one of the functions to increase,

Is a function of holding the login state.
This function, if the user has been allowed to hold the login state, even after closing the browser
holds the login information to the cookie, user name, and is a function that allows you to log in without having to re-enter the password.

: <Sec remember-me> the attributes of the elements, I show below.

spring-security.xml

```
<Sec: http  auto-config = "true"  use-expressions = "true" >
  <-! OMITTED ->
  <sec: Remember-me  key = "terasoluna-Tourreservation-miles / YlnHv"
          token-validity-seconds = "# {30 * 24 * 60 * 60}"  />   <! - an attribute of the specified order (1)
~ (2) ->
  <- OMITTED ->!
</ sec: http>
```

No.	Description
(1)	key to attribute to specify a unique key for holding a cookie for Remeber Me. If not specified, in order to generate a unique key at startup, it is recommended that it should be specified when considering the activation time increase.
(2)	" token-validity-seconds to attribute specifies the effective time of the cookie for Remeber Me in seconds. In this example you have set for 30 days. If not specified, the default 14 days is the expiration date.

The attributes other than the above, Spring Security documentation see.

The login form it is necessary to prepare the flag to enable the following as "Remeber Me" feature.

```
<Form  method = "POST"
  Action = "$ {PageContext.Request.ContextPath} / authentication" >
    <- OMITTED -!>
    <label  for = "_Spring_security_remember_me" > Remember Me: </ label>
    <input  name = " _Spring_security_remember_me "
      id = "_Spring_security_remember_me"  type = "checkbox"
      Checked = "Checked" >  <-! (1) ->
    <input  type = "Submit"  value = "LOGIN" >
    <-! OMITTED ->
 < / form>
```

No.	Description
(1)	The HTTP parameters, _Spring_security_remember_me it is possible to set the, true when it is requested by, it is possible to avoid the next authentication.

6.3.3. How to extend

6.3.3.1. **UserDetailsService** extension of

If the user ID at the time of authentication, also information other than the password you want to get,

- org.springframework.security.core.userdetails.UserDetails
- org.springframework.security.core.userdetails.userDetailsService

There is a need to implement.

If you need to always appear in the header of the screen attached information such as login user's name and department, to get from DB in every request it is inefficient. UserDetails it is allowed to hold the object, SecurityContext or <sec: authentication> To make can be accessed from the tag is required this extension.

6.3.3.1.1. **UserDetails** extension of

Customer information to other authentication information also hold ReservationUserDetails I create a class.

```
public  class  ReservationUserDetails  extends  User  {  // (1)
    // OMITTED

    Private  Final  Customer  Customer ;  // (2)

    Private  static  Final  List <?  extends  GrantedAuthority >  DEFAULT_AUTHORITIES  =  Collections
            . singletonList ( new  SimpleGrantedAuthority ( "ROLE_USER" ));          // (3)

    public  ReservationUserDetails ( Customer  Customer )  {
        super ( Customer . GetCustomerCode (),
                Customer . GetCustomerPassword (),  true ,  true ,  true ,  true ,  DEFAULT_AUTHORITIES );  //
(4)
```

```
        this . Customer  =  Customer ;
    }

    public  Customer  getCustomer ()  {  // (5)
        Return  Customer ;
    }
}
```

No.	Description
(1)	`UserDetails` is the default class of, `Org.Springframework.Security.Core.Userdetails.User` I inherit the class.
(2)	I hold the DomainObject class with the authentication information and customer information.
(3)	Authorization information, `Org.Springframework.Security.Core.Authority.SimpleGrantedAuthority` I created in the constructor of. Here I will give authority called "ROLE_USER". The implementation is simple implementation, originally authorization information should be obtained from another table in the DB.
(4)	To the constructor of the superclass, user ID with the DomainObject, to set the password.
(5)	`UserDetails` methods to access customer information via.

Note

`User` can only inherit a class, if you can not get the business requirements, `UserDetails` may be implement the interface.

6.3.3.1.2. Own **`UserDetailsService`** implementation of

`UserDetailsService` I create a ReservationUserDetailsService class that implements.
In this example, `Customer` implementing the process of acquiring the object `CustomerSharedService` a class by injection, and acquires customer information from the DB.

```
public  class  AruiesuirvationUserDetailsService  implements  UserDetailsService  {
    Inject
    CustomerSharedService  CustomerSharedService ;

    Override
    public  UserDetails  LoadUserByUsername ( String  username )  throws  UsernameNotFoundException  {
        Customer  Customer  =  CustomerSharedService . findOne ( username );
        // OMITTED
        Return  new  ReservationUserDetails ( Customer );
    }

}
```

6.3.3.1.3. How to use

You created `AruiesuirvationUserDetailsService` , `ReservationUserDetails` to describe how to use the.

- spring-security.xml

```
<Sec: authentication-Manager>
    <sec: authentication-provider  user-service-ref = "UserDetailsService" > <-! (1) ->
        <sec: password-encoder  ref = "PasswordEncoder"  />
    </ sec: authentication-provider>
</ sec: authentication-Manager>

<Bean  id = "UserDetailsService"
    class = "Com.Example.Domain.Service.Userdetails.AruiesuirvationUserDetailsService" > <-! (2) ->
</ bean>
<-! OMITTED ->
```

No.	Description
(1)	`ReservationUserDetailsService` the Bean ID of I defined in ref attribute.
(2)	`ReservationUserDetailsService` I Bean define.

- JSP

`<Sec: authentication>` You can use the tag `Customer` to access the object.

```
<Sec: authentication  property = "Principal.Customer"  var = "Customer" /> <- (1) -!>
$ {f: h (Customer.CustomerName)} <-! (1) ->
```

No.	Description
(1)	`ReservationUserDetails` with a `Customer` I store object in a variable.
(2)	Was stored in the variable `Customer` I display any of the properties of the object. `f: h ()` For, *XSS countermeasures* see.

- Controller

```
RequestMapping ( method  =  RequestMethod . GET )
public  String  view ( AttoAuthenticationPrincipal  ReservationUserDetails  UserDetails ,  Model  model
)  {
    // Get Customer
    Customer  Customer  =  UserDetails . getCustomer ();  // (1)
    // OMITTED ...
}
```

No.	Description
(1)	`ReservationUserDetails` from, in login `Customer` I get the object. To perform a business process by passing the object to the Service class.

Note

If the customer information is changed, it is not once logout `ReservationUserDetails` has `Customer` object is not changed.

Frequently or modified can information, information that is changed by a user other than the logged-in user (such as an administrator) is better that you do not want to keep.

6.3.3.2. **`AuthenticationProvider`** extension of

Are provided by Spring Security authentication provider if there is a business requirement that can not be supported by, `Org.Springframework.Security.Authentication.AuthenticationProvider` there is a need to create a class that implements the interface.

Here, the user name, password, **company ID (unique authentication parameters)** shows an extended example for performing DB authentication using the three parameters.

In order to achieve the above requirements, it is necessary to create a class that will be described below.

No.	Description
(1)	To hold user name, password, company identifier (own authentication parameters) `Org.Springframework.Security.Core.Authentication` implementation class of interface. Here, `Org.Springframework.Security.Authentication.YuesuiaruenueiemuiPieisswordAuthenticationToken` you create by inheriting the class.
(2)	Do DB authentication using user name, password, company identifier (own authentication parameters) `Org.Springframework.Security.Authentication.AuthenticationProvider` implementation class. Here, `Org.Springframework.Security.Authentication.Dao.DaoAuthenticationProvider` you create by inheriting the class.

(3)	Username, password, company ID (unique authentication parameters) is acquired from the request parameter, `AuthenticationManager` (`AuthenticationProvider` pass) `Authentication` servlet filter class for generating. Here, `Org.Springframework.Security.Web.Authentication.YuesuiaruenueiemuiPieiesuswordAuthenticationFilter` you create by inheriting the class.

Tip

Here, because it is an example of adding own parameters as parameters for authentication, `Authentication` implementation class and interface `Authentication` become extensions required servlet filter class for generating.

When authenticating only the user name and password, `AuthenticationProvider` it only creates the implementation class of the interface, it is possible to extend the authentication process.

6.3.3.2.1. `YuesuiaruenueiemuiPieisswordAuthenticationToken` extension of

Here, `YuesuiaruenueiemuiPieisswordAuthenticationToken` inherits the class, in addition to the user name and password, to create a class that holds a company identifier (unique authentication parameters).

```
// Import OMITTED
public class ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken extends
    YuesuiaruenueiemuiPieisswordAuthenticationToken {

    Private static Final long serialVersionUID = SpringSecurityCoreVersion . SERIAL_VERSION_UID ;

    // (1)
    Private Final String companyId ;

    // (2)
    public ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken (
            Object principal , Object Credentials , String companyId ) {
        super ( principal , Credentials );

        this . companyId = companyId ;
    }

    // (3)
    public ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken (
            Object principal , Object Credentials , String companyId ,
            Collection <? extends GrantedAuthority > Authorities ) {
        super ( principal , Credentials , Authorities );
        this . companyId = companyId ;
    }

    public String GetCompanyId () {
        Return companyId ;
    }

}
```

No.	Description
(1)	I want to create a field that holds the company identifier.
(2)	Create a constructor to use when creating an instance to hold the pre-authentication information (information that is specified in the request parameter).
(3)	Create a constructor to use when creating an instance to hold the authenticated information. By passing the authorization information to the argument of the constructor of the parent class, it becomes authenticated state.

6.3.3.2.2. `DaoAuthenticationProvide` extension of

Here, `DaoAuthenticationProvider` inherit the class, user name and password to create a class that DB authentication using the company identifier.

```
// Import OMITTED
public class ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuesudaburyuordAuthenticationProvider extends
    DaoAuthenticationProvider {

    // Omitted

    Override
    protected void EididiaitiionalAuthenticationChecks ( UserDetails UserDetails ,
            YuesuiaruenueiemuiPieisswordAuthenticationToken authentication )
            throws AuthenticationException {
```

```
        // (1)
        super . EididiaitiionalAuthenticationChecks ( UserDetails ,  authentication );

        // (2)
        ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken
ShioemupieienuwaiaidiyuesuiaruenuamePasswordAuthentication  =
            ( ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken )  authentication ;
        String  RequestedCompanyId  =  ShioemupieienuwaiaidiyuesuiaruenuamePasswordAuthentication .
GetCompanyId ();
        String  companyId  =  (( SampleUserDetails )  UserDetails )
                . getAccount . () GetCompanyId ();
        if  (! companyId . equals ( RequestedCompanyId ))  {
            throw  new  BadCredentialsException ( messages . getMessage (
                    "EibiesutiarueishitiyuesuiarudiitieiilsAuthenticationProvider.BadCredentials" ,
                    "Bad Credentials" ));
        }
    }

    Override
    protected  Authentication  ShiarueateSuccessAuthentication ( Object  principal ,
            Authentication  authentication ,  UserDetails  user )  {
        String  companyId  =  (( SampleUserDetails )  user .) getAccount ()
                . GetCompanyId ();
        // (3)
        Return  new  ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken ( user ,
                authentication . getCredentials () ,  companyId ,
                user . GetAuthorities ());
    }

    Override
    public  boolean  Supports ( Class <?>  authentication )  {
        // (4)
        Return  ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken . class
                . IsAssignableFrom ( authentication );
    }

}
```

No.	Description
(1)	Call the method of the parent class, to perform the check processing Spring Security has to offer. Password authentication is performed at this timing.
(2)	If the password authentication is successful, checking the validity of the company identifier (unique authentication parameters). In the above example, it is checked whether the company identifier held in the company identifier and tables that are request matches.
(3)	If the password authentication and its own authentication process is successful, authenticated status `CompanyIdUsernamePasswordAuthenticationToken` and returns Create a.
(4)	`CompanyIdUsernamePasswordAuthenticationToken` castable the `Authentication` when is specified, using this class to perform the authentication process.

Tip

User existence check, user status check (disabled user, during the lock user, checks such as use expired user), `EididiaitiionalAuthenticationChecks` before the method is called, is performed as a treatment of the parent class.

6.3.3.2.3. **`YuesuiaruenueiemuiPieiesuswordAuthenticationFilter`** extension of

Here, `YuesuiaruenueiemuiPieiesuswordAuthenticationFilter` inherit the class, authentication information (user name, password, company identifier) the `AuthenticationProvider` to create a servlet filter class for deliver to.

`attemptAuthentication` implementation methods, `YuesuiaruenueiemuiPieiesuswordAuthenticationFilter` you can copy the methods of the class, is one that you customized.

```
// Import OMITTED
public  class  ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuswordAuthenticationFilter  extends
    YuesuiaruenueiemuiPieiesuswordAuthenticationFilter  {

    Override
    public  Authentication  AttemptAuthentication ( HttpServletRequest  request ,
            HttpServletResponse  response )  throws  AuthenticationException  {

        if  (! request . getMethod . () equals ( "POST" ))  {
            throw  new  EiyutietchiinticationServiceException ( "Authentication method not supported:"
                    +  request . getMethod ());
        }

        // (1)
        // Obtain UserName, Password, companyId
```

```
        String  username  =  super . ObtainUsername ( request );
        String  password  =  super . ObtainPassword ( request );
        String  companyId  =  ObtainCompanyId ( request );
        if  ( username  ==  null )  {
            username  =  "" ;
        }  else  {
            username  =  username . trim ();
        }
        if  ( password  ==  null )  {
            password  =  "" ;
        }
        ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken  authrequest  =
            new  ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken ( username ,  password ,
companyId );

        // Allow subclasses to set the "details" property
        SetDetails ( request ,  authrequest );

        Return  this . GetAuthenticationManager . () authenticate ( authrequest );  // (2)
    }

    // (3)
    protected  String  ObtainCompanyId ( HttpServletRequest  request )  {
        Return  request . getParameter ( "companyId" );
    }
}
```

No.	Description
(1)	Authentication information obtained from the request parameter (user name, password, company identifier) than, `ShioemupieienuwaiaidiyuesuiaruenueiemuiPieisswordAuthenticationToken` to generate an instance of.
(2)	Specified authentication information (in the request parameters `CompanyIdUsernamePasswordAuthenticationToken` by specifying the instance of), `Org.Springframework.Security.Authentication.AuthenticationManager` of `authenticate` to call the method. `AuthenticationManager` When you call a method of, `AuthenticationProvider` it is a mechanism that authentication process is called.
(3)	Company identifier, `"companyId"` I get from the request parameter named.

Note

For input check of authentication information

In load reduction, etc. to the DB server, for obvious input error, there is a case in which you want to do a check in advance. In that case, *extension of UsernamePasswordAuthenticationFilter* As, `YuesuiaruenueiemuiPieiesuswordAuthenticationFilter` it is possible to extend, and can perform input validation process.

The input check in the above example is not performed.

Todo

Input checking credentials, it handles the request Controller class, it is also possible to perform by using a Bean Validation.

For information on how to input checks using the Bean Validation, it is planned to be added in the future.

6.3.3.2.4. Application of extended authentication process

I apply a user name, password, the DB authentication function using the company identifier (own authentication parameters) in Spring Security.

`spring-security.xml`

```
 <-! Omitted ->

 <- (1) -!>
 <sec: http
     auto-config = "false"
     use-expressions = "true"
     entry-point-ref = "EruojiaienuyuarulAuthenticationEntryPoint" >

     <-! Omitted ->

     <- (2) -!>
     <sec: custom-filter
         position = "FORM_LOGIN_FILTER"
         ref = "ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuswordAuthenticationFilter"  />

     <-! Omitted ->

     <Sec: csrf  token-repository-ref = "CsrfTokenRepository"  />
```

```
    <Sec: logout
        logout-url = "/ logout"
        logout-success-url = "/ Login"
        delete-cookies = "JSESSIONID"  />

    <-! Omitted ->

    <Sec: Intercept-url  pattern = "/ Login"  access = "permitAll"  />
    <sec: Intercept-url  pattern = "/ **"  access = "isAuthenticated ()"  />

    <-! Omitted ->

</ Sec: http>

<-! (3) ->
<bean  id = "EruojiaienuyuarulAuthenticationEntryPoint"
    class = "Org.Springframework.Security.Web.Authentication.EruojiaienuyuarulAuthenticationEntryPoint" >
    <constructor-Arg  value = "/ Login"  />
</ bean>

<-! (4) ->
<bean  id = "ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuswordAuthenticationFilter"
    class =
"Com.Example.App.Common.Security.ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuswordAuthenticationFilter" >
    <-! (5) ->
    <property  name = "AruikyuyuaiaruiesueiyutihenticationRequestMatcher" >
        <bean  class = "Org.Springframework.Security.Web.Authentication.Logout.LogoutFilter $
EfuaierutiirProcessUrlRequestMatcher" >
            <constructor-Arg  value = "/ authentication"  />
        </ bean>
    </ property>
    <-! (6) ->
    <property  name = "AuthenticationManager"  ref = "AuthenticationManager"  />
    <-! (7) ->
    <property  name = "EsuiesuesuionAuthenticationStrategy"  ref = "EsuiesuesuionAuthenticationStrategy"  />
    <-! (8) - >
    <property  name = "EiyutihenticationFailureHandler" >
        <bean  class =
"Org.Springframework.Security.Web.Authentication.EsuaiemupieruiyuaruerueiyutihenticationFailureHandler" >
            <constructor-Arg  value = "/ Login? error = true"  />
        </ bean>
    </ property >
    <-! (9) ->
    <property  name = "EiyutihenticationSuccessHandler" >
        <bean  class =
"Org.Springframework.Security.Web.Authentication.EsuaiemupieruiyuaruerueiyutihenticationSuccessHandler"  />
    </ property>
</ bean>

<- (6 ') -!>
<sec: authentication-Manager  alias = "AuthenticationManager" >
    <sec: authentication-provider  ref =
"ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuesudaburyuordAuthenticationProvider"  />
</ sec: authentication-Manager>
<bean  id = "ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuesudaburyuordAuthenticationProvider "
    class =
"Com.Example.App.Common.Security.ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuesudaburyuordAuthenticationProv
ider" >
    <property  name = "UserDetailsService"  ref = "SampleUserDetailsService"  />
    <property  name = "PasswordEncoder"  ref = "PasswordEncoder"  />
</ bean>

<- - (7 ')!>
<bean  id = "EsuiesuesuionAuthenticationStrategy"
    class =
"Org.Springframework.Security.Web.Authentication.Session.ShioemupioesuaitiiesuiesuesuionAuthenticationStrategy
" >
    <constructor-Arg>
        <util: list>
            <bean  class = "Org.Springframework.Security.Web.Csrf.ShisrfAuthenticationStrategy" >
                <constructor-Arg  ref = "CsrfTokenRepository"  />
            </ bean>
            <bean  class =
"Org.Springframework.Security.Web.Authentication.Session.EsuiesuesuaioenuefuixationProtectionStrategy"  />
        </ util: list>
    </ constructor-Arg>
</ bean>

<Bean  id = "CsrfTokenRepository"
    class = "Org.Springframework.Security.Web.Csrf.EichititipiesuessionCsrfTokenRepository"  />


<-! Omitted ->
```

No.	Description
(1)	`custom-filter` when to replace to the "FORM_LOGIN_FILTER" using elements, `http` element attributes, it is necessary to make the following settings. • Since it is not possible to use the automatic configuration, `auto-config = "false"` Specify the, `auto-config` to remove attributes.

- `form-login` because you can not use the element, `entry-point-ref` using the attribute, use `AuthenticationEntryPoint` explicitly specify a.

(2)	`custom-filter` using the element to replace the "FORM_LOGIN_FILTER". `custom-filter` element `position` to attribute `"FORM_LOGIN_FILTER"` Specify the, `ref` to specify the bean ID of servlet filter that extends to the attribute.
(3)	`http` element `entry-point-ref` specified in the attribute `AuthenticationEntryPoint` I define a bean of. Here, `form-Login` is used when you specify the elements `org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint` you are defining a bean of class.
(4)	I define a bean of servlet filter to be used as "FORM_LOGIN_FILTER". Here, we extended servlet filter class (`ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuswordAuthenticationFilter` you are defining a bean of).
(5)	`requiresAuthenticationRequestMatcher` the property, for detecting a request to perform an authentication process `RequestMatcher` specifying the instance. Here, `/ authentication` are set to perform an authentication process when there is a request in the path. This is, `form-Login` elements `login-processing-url` attribute to `"/ authentication"` is synonymous with the specified a.
(6)	`authenticationManager` to property, `authentication-Manager` of the element `alias` I specify the value set for the attribute. `authentication-manager` of the element `alias` If you specify the attribute, Spring Security has generated `AuthenticationManager` the bean of, it becomes as can be DI to other bean.
(6 ')	Spring Security generates `AuthenticationManager` against, the extended `AuthenticationProvider` (`ShioemupieienuwaiaidiyuesuiaruenueiemuiPieiesuesudaburyuordAuthenticationProvider` to set).
(7)	`sessionAuthenticationStrategy` to property, component that controls the handling of the time of authentication success session (`EsuiesuesuionAuthenticationStrategy` I specify the bean of).
(7 ')	To control the handling at the time of authentication success session component (`EsuiesuesuionAuthenticationStrategy` I define a bean of). Here, it is provided by the Spring Security, - Component to remake the CSRF token (`ShisrfAuthenticationStrategy`) - Component that generates a new session in order to prevent session Fixation attack (`EsuiesuesuaioenuefuixationProtectionStrategy`) I have enabled.
(8)	`authenticationFailureHandler` the property, I specify the handler class that is called authentication failure.
(9)	`authenticationSuccessHandler` the property, I specify the handler class called at the time of authentication success.

Note

`auto-config = "false"` if you have specified, `<sec: http-Basic>` element and `<sec: logout>` element, not enabled If you do not explicitly defined.

6.3.3.2.5. Creating a Login form

Here, *create a login form* with respect to the screen that was introduced in (JSP), to add a company identifier.

```
<Form: form  Action = "$ {PageContext.Request.ContextPath} / authentication"  method = "POST" >
    <-! OMITTED ->
    <span> User Id </ span> <br>
    <input  type = "text "  id = "username"  name = "j_username" > <br>
    <span> Company Id </ span> <br>
    <input  type = "text"  id = "companyId"  name = "companyId" > <br>   <! - (1) ->
     <span> Password </ span> <br>
    <input  type = "password"  id = "password"  name = "j_password" > <br>
    <-! OMITTED ->
</ form: form>
```

No.	Description
(1)	In the input field name of the company identifier, `"companyId"` I specify a.

6.3.4. Appendix

6.3.4.1. Transition destination specified possible authentication success handler

In the authentication using Spring Security, if successful authentication,

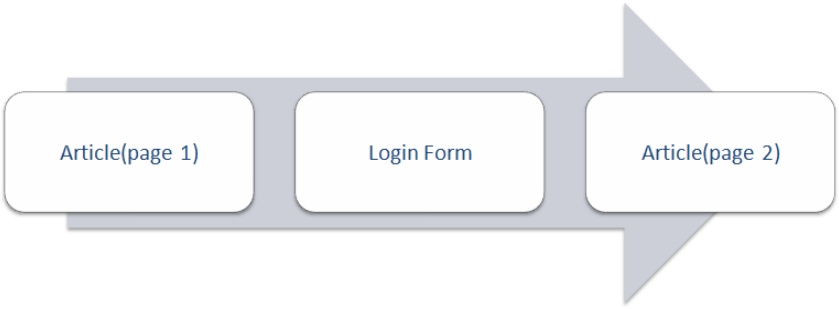
- bean definition file (`spring-security.xml`) path described in (`<form-Login>` elements `default-target-url` path specified for the attribute)
- Path for displaying was accessed before login "authentication is necessary protection Page"

I will transition to.

The common library, in addition to the functions Spring Security is provided, the class can be specified (the transition destination of the path request parameters `org.terasoluna.gfw.web.security.RedirectAuthenticationHandler` offering).

`RedirectAuthenticationHandler` is a class that was created in order to realize a mechanism such as the following.

- It is necessary to login in order to view the page
- I want to specify the transition destination page after login in JSP side (transition source JSP)



Picture - Screen_Flow

`RedirectAuthenticationHandler` an example of the use of, I shown below.

Description example of JSP of transition original screen

```
<Form: form  Action = "$ {PageContext.Request.ContextPath} / Login"  method = "Get" >
    <-! OMITTED ->
  <input  type = "hidden"  name = "redirectTo"
    value = "$ {pageContext .Request.ContextPath} / Reservetour / read?
    $ {f: query (ReserveTourForm)} & Page.Page = $ {f: h (param ['Page.Page'])}
    & Page.Size = $ {f: h (param ['Page.Size'])} "  />   <- (1) ->!
</ form: form>
```

No.	Description
(1)	as hidden item, I set the "URL of the page you want to transition to after successful login". hidden field name of the item (request parameter name) is " `redirectTo` I specify the ". Field name (request parameter names), `AruidiairectAuthenticationHandler` of `targetUrlParameter` it is necessary to match the value of the property.

Description of JSP login screen example

```
<Form: form  Action = "$ {PageContext.Request.ContextPath} / authentication"  method = "POST" >
     <-! OMITTED ->
     <input  type = "Submit"
       value = "Login" >
     <input  type = " hidden "  name = "redirectTo"  value = "$ {f: h (Param.RedirectTo)}"  />   ! <- (1) ->
     <- OMITTED ->!
</ form: form>
```

No.	Description
(1)	as hidden item, I set the "URL of the page you want to transition to after a successful login" which was passed in the request parameters from the original transition screen. hidden field name of the item (request parameter name) is " `redirectTo` I specify the ". Field name (request parameter names), `AruidiairectAuthenticationHandler` of `targetUrlParameter` it is necessary to match the value of the property.

Spring Security configuration file

```
<Sec: http  auto-config = "true" >
    <-! OMITTED ->
    <-! (1) ->
    <sec: form-Login
        Login-page = "/ Login"
        Login-processing-url = "/ authentication"
        authentication-failure-handler-ref = "EiyutihenticationFailureHandler"
        authentication-success-handler-ref = "EiyutihenticationSuccessHandler"  />
    ! <- OMITTED ->
</ sec: http>

<-! (2) ->
<bean  id = "EiyutihenticationSuccessHandler"
    class = "Org.Terasoluna.Gfw.Web.Security.AruidiairectAuthenticationHandler" >
</ bean>

<-! (3) ->
<bean  id = "EiyutihenticationFailureHandler"
    class =
"Org.Springframework.Security.Web.Authentication.IekkusushiipitiaioenuemueipipiaienujieiyutihenticationFailure
Handler" >
    <property  name = "DefaultFailureUrl"  value = "? / Login error = true" />  <-! (4) ->
    <property  name = "UseForward"  value = "true" />  <-! (5) ->
</ bean>
```

No.	Description
(1)	`authentication-failure-handler-ref` (authentication handler settings at the time of error) and `authentication-success-handler-ref` I specify the BeanId of (handler settings at the time of authentication success).
(2)	`authentication-success-handler-ref` as a bean that is referenced from `org.terasoluna.gfw.web.security.RedirectAuthenticationHandler` I to define.
(3)	`authentication-failure-handler-ref` as a bean that is referenced from `org.springframework.security.web.authentication.ExceptionMappingAuthenticationFailureHandler` I to define.
(4)	I specify the authentication failure of the transition destination path. In the above example, the query (which indicates that it is a transition after path and authentication error of login screen `error = true` you have set a).
(5)	**If you want to use this function, it is necessary to specify the useForward to true.** `true` and by specifying the, when a transition to the screen to be displayed in the authentication failure (login screen), Forward instead Redirect is used. This is because it is necessary to include "URL of the page to transition after a successful login to the" in the request parameters in the request for authentication process. When using the Redirect would display an authentication error screen, since the "URL of the page to transition after successful login" is not possible take over from the request parameter, it transitions to the screen the login is specified even if successful can not. In order to avoid this event, by using the Forward it is necessary to take over as the new "URL of the page to transition after successful login" from the request parameter.

Tip

`RedirectAuthenticationHandler` because the open redirector vulnerability is applied, " http://google.com unable to transition to an external site such as ". If you want to move to the external site, `Org.Springframework.Security.Web.RedirectStrategy` create a class that implements the, `AruidiairectAuthenticationHandler` of `targetUrlParameterRedirectStrategy` it is possible to be done by injection to the property.

As a side note when expanding, `redirectTo` be tampered values it is necessary to avoid problems. For example, I can be considered measures such as the following.

- Instead of specifying the transition destination URL directly to redirect to a URL corresponding to the ID by the ID, such as the page number.
- Check the transition destination URL, and redirect only URL that matches the white list.