Краткий обзор Spring Security

Ключевые объекты контекста Spring Security.

Процесс аутентификации в Spring Security.

Подключение собственно самого Spring Security к проекту.

Spring Security это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Проект был начат Беном Алексом (Ben Alex) в конце 2003 года под именем «Acegi Security», первый релиз вышел в 2004 году. Впоследствии проект был поглощён Spring'ом и стал его официальным дочерним проектом. Впервые публично представлен под новым именем Spring Security 2.0.0 в апреле 2008 года.

**Ключевые объекты контекста Spring Security:**

SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением. По умолчанию SecurityContextHolder используетThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов исполняющихся в том же самом потоке. Для того что бы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy). Более подробно SecurityContextHolder

SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.

Authentication представляет пользователя (Principal) с точки зрения Spring Security.

GrantedAuthority отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN.

UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetails содержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.

UserDetailsService, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса: *UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;*

Позволяет получить из источника данных объект пользователя и сформировать из него объект UserDetails который будет использоваться контекстом Spring Security.

**Аутентификация**

(1) Пользователю будет предложено войти в систему предоставив имя (логин или email) и пароль. Имя пользователя и пароль объединяются в экземпляр класса UsernamePasswordAuthenticationToken(экземпляр интерфейса Authentication) после чего он передается экземпляру AuthenticationManager для проверки.

(2) В случае если пароль не соответствует имени пользователя будет выброшено исключение BadCredentialsException с сообщением “Bad Credentials”.

(3) Если аутентификация прошла успешно возвращает полностью заполненный экземпляр Authentication.

(4) Для пользователя устанавливается контекст безопасности путем вызова метода SecurityContextHolder.getContext().setAuthentication(…), куда передается объект который вернул AuthenticationManager.

**Подключение поддержки Security для SpringFramework приложения:**

1. **pom.xml**

<properties>

<spring.version>3.1.4.RELEASE</spring.version>

</properties>

<!-- Spring Security -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>${spring.version}</version>

</dependency>

2. **web.xml**

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

Ну и конечно же сам файл с настройкой **security.xml**

<beans:beans xmlns="http://www.springframework.org/schema/security"

xmlns:beans="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security-3.0.xsd">

<http access-denied-page="/error403.jsp">

<intercept-url pattern="/index\*" access="ROLE\_USER,ROLE\_ANONYMOUS"/>

<intercept-url pattern="/add\*" access="ROLE\_USER"/>

<intercept-url pattern="/delete/\*" access="ROLE\_ADMIN"/>

<form-login login-page="/login.jsp" default-target-url="/index" authentication-failure-url="/login.jsp?error=true"/>

<logout logout-url="/logout" logout-success-url="/index"/>

<anonymous username="guest" granted-authority="ROLE\_ANONYMOUS"/> //настройка роли ANONYMOUS

<remember-me/> //т.к. тут параметр “key” не установлен, то для возможности использования подхода на основе хеш-токенов будет сгенерировано безопасное случайное значение ключа для хеширования (*но на это необходимо дополнительное время*)

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name="admin" password="pass" authorities="ROLE\_ADMIN,ROLE\_USER"/>

<user name="user1" password="1111" authorities="ROLE\_USER"/>

<user name="user2" password="2222" disabled="true" authorities="ROLE\_USER"/>

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

Пояснение к коду security.xml:

Подразумевается что на странице login.jsp есть форма с action="/j\_spring\_security\_check" в которой содержаться input'ы для имени и пароля с name=«j\_username» и name=«j\_password», а также checkbox c name="\_spring\_security\_remember\_me". Это всё специальные значения которые требует Spring Security иначе параметры просто не будут передаваться в контекст безопасности. После успешного прохождения аутентификации пользователь будет перенаправлен на страницу /index где уже применяются правила авторизации. Если не указывать форму и url в http spring-security тогда по умолчанию будет работать basic authentication или можно подключить basic authentication насильно указав в http spring-security <http-basic />

На url /index\* доступ могут иметь пользователи с правами ROLE\_USER, а также гости (все подключения которые не прошли аутентификацию получают имя guest и права ROLE\_ANONYMOUS).

Доступ к url /add\* имеют только пользователи с правами ROLE\_USER Доступ к url /delete\* только пользователи с правами ROLE\_ADMIN

Также, авторизованный доступ можно прикручивать на методы, для этого в security.xml нужно добавить, следующий элемент:

<global-method-security secured-annotations="enabled" />

и в самом коде:

public interface AdminService {

@Secured("ROLE\_ADMIN")

public Account editAccount(Account account);

}

В примере пользователи хранятся списком в .xml файле т.е. сравнение UsernamePasswordAuthenticationToken будет производиться с этими данными. Для того чтобы сравнивать с пользователями в базе данных (используя ORM), нужно реализовать метод loadUserByUsername интерфейса UserDetailsService и указать в <authentication-provider user-service-ref=«userDetailsService» > ссылку на bean вашей реализации UserDetailsService. Если же вы не используете ORM и вам нужно просто вытянуть из базы пользователей и их права с помощью JDBC, нужно определить DataSource бин с помощью которого Spring будет знать как достучаться до БД и в <authentication-provider> указать ссылку на этот бин и определить два запроса по которым будут вытягиваться нужные данные users-by-username-query и authorities-by-username-query.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/<yourDataBaseName>" />

<property name="username" value="root" />

<property name="password" value="password" />

</bean>

<authentication-provider>

<jdbc-user-service data-source-ref="dataSource"

users-by-username-query="select username, password, enabled

from users where username = ?"

authorities-by-username-query="select u.username, au.authority from users u,

authorities au where u.id = au.user\_id and u.username = ?" />

</authentication-provider>

</beans>

Также может осуществляться проверка хешированных паролей:

<authentication-manager>

<authentication-provider>

<password-encoder hash="sha"/> //алгоритм по умолчанию, но лучше использовать bcrypt или sha512

<user-service>

<user name="jimi" password="d7e6351eaa13189a5a3" authorities="ROLE\_USER, ROLE\_ADMIN" />

<user name="bob" password="4e7421b1b8765d8f9406d" authorities="ROLE\_USER" />

</user-service>

</authentication-provider>

</authentication-manager>

//блок дополнительной настройки шифровальщика

<password-encoder hash="sha">

<salt-source user-property="username"/> //источник криптографической соли

</password-encoder>