Spring Security/Конфигурирование с помощью пространства имён

Материал из Викиучебника — открытые книги для открытого мира
< Spring Security

Эта статья представляет собой перевод Spring Security Reference Documentation, Ben Alex, Luke Taylor 3.0.2.RELEASE, Глава 2, Конфигурирование пространства имен Security.

Содержание

Введение

Конфигурирование с помощью пространства имен было доступно, начиная с версии 2.0 Spring Framework. Это позволило дополнить традиционный синтаксис бинов контекста приложения Spring'a элементами из дополнительных схем XML. Более подробную информацию можно найти в справочном руководстве Spring. Элементы пространства имен могут использоваться просто для более краткой настройки отдельных бинов или можно воспользоваться мощью альтернативного синтаксиса конфигурации, которая более точно соответствует предметной области и скрывает сложности, лежащие в основе фреймворка, от пользователя. Простой элемент может скрывать тот факт, что несколько бинов и этапов обработки будут добавлены к контексту приложения. Например, если добавить следующие элемент из пространства имен security в контекст приложения, то будет запускается встроенный LDAP сервер для использования в приложении в целях:

```
<security:ldap-server />
```

Это гораздо проще, чем настраивать зависимости эквивалентных бинов Apache Directory Server. Атрибуты элемента ldap-server большинство типовых конфигурационных требований и пользователь освобожден от необходимости беспокоиться о том, какие бины необходимо создавать и какие соответственно какие имена бинов.[1]. При использовании хорошего XML-редактора при редактировании файла контекста приложения будет предоставлена информация о имеющихся атрибутах и элементах. Мы рекомендуем вам попробовать SpringSource Tool Suite, который имеет специальные возможности для работы со стандартными пространствами имен Spring'a.

Чтобы начать использовать пространство имен Security в контексте вашего приложения, все что нужно вам сделать это добавить объявление схемы в файл контекста приложения:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:security="http://www.springframework.org/schema/security"
  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-3.1.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security-3.1.xsd">
    ...
</beans>
```

Во многих примерах, которые вы увидите, мы будем часто использовать по умолчанию пространство имен "security", а не "bean", это означает, что мы можем опустить префикс для всех элементов пространства имен "security", в результате чего содержание конфигурационного файла будет проще для чтения и понимания. Вы можете поступать подобным образом, если ваш контекст приложения разделен

на несколько отдельных файлов и большая часть конфигурации, которая относится к безопасности, собрана в одном из них. В этом случае файл контекста приложения для конфигурирования безопасности будет начинаться следующим образом:

```
<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-3.1.xsd
           http://www.springframework.org/schema/security
           http://www.springframework.org/schema/security/spring-security-3.1.xsd">
    ...
</beans:beans>
```

Мы предполагаем что этот синтаксис будет использоваться в этой главе.

Устройство пространства имен

Пространство имен спроектировано таким образом, чтобы охватить наиболее общие варианты использования фреймворка, и предоставить простой и краткий синтаксис по включению возможностей фреймворка в приложение. Дизайн базируется на крупномасштабных зависимостях внутри фреймворка и может быть разделен на следующие области:

- *Web/HTTP Security* - наиболее сложная часть. Устанавливает фильтры и связанные с ними сервисные бины, используемые аутентификационными механизмами фреймворка, защитой URL, показом страниц ошибок и аутентификации и многое другое
- *Business Object (Method) Security* — опции для защиты уровня сервисов.
- *AuthenticationManager* — обрабатывает аутентификационные запросы от других частей фреймворка.
- *AccessDecisionManager* — предоставляет решение о разрешении доступа к веб-страницам и методам. Всегда будет зарегистрированный по умолчанию менеджер, но вы также можете свой собственный менеджер, объявленный с помощью обычного <bean/> синтаксиса Spring.
- *AuthenticationProviders* — механизмы, опираясь на которые, менеджер аутентификации выполняет фактическую аутентификацию пользователей. Пространство имен предоставляет поддержку нескольких стандартных вариантов, а так же средства добавления пользовательских бинов, объявленных с помощью традиционного синтаксиса.
- *UserDetailsService* — тесно связан с провайдером аутентификации, но так же может запрашиваться другими бинами.

В последующих разделах мы увидим, как выполнять их конфигурирование.

Начинаем конфигурирование с помощью пространства имен Security

В этом разделе мы рассмотрим, как можно создать конфигурацию с помощью пространства имен, чтобы использовать основные возможности фреймворка. Предположим, что вы изначально хотите как можно быстрее приступить к работе и добавить поддержку аутентификации и контроля доступа в уже существующее веб-приложение, с несколькими тестовыми логинами. Мы рассмотрим как потом перейти к аутентификации, которая будет получать необходимые сведения из базы данных или другого хранилища с информацией о безопасности. В последующих разделах будут представлены более продвинутые варианты конфигурации пространства имен.

Конфигурация web.xml

Первое что необходимо сделать, это добавить следующее объявление фильтра в ваш web.xml файл:

```
<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>
```

Это дает возможность встроиться внутрь веб-инфраструктуры Spring Security. Класс `DelegatingFilterProxy`, из состава Spring Framework , передает запросы реализации фильтра, которая определена как Spring бин в контексте вашего приложения. В данном случае бин называется "springSecurityFilterChain", который является внутренним инфраструктурным бином, создаваемым при обработке пространства имен для решения вопросов связанных с веб-безопасностью. Заметим, что вы не должны использовать бин с этим именем самостоятельно. После того как вы добавили эти строки в ваш `web.xml`, вы можете приступить к редактированию файла контекста приложения. Сервисы веб-безопасности настраиваются с помощью элемента `<http>`.

Минимальная <http> конфигурация

Все что нужно чтобы начала работать веб-безопасность, это написать следующие строки:

```
<http auto-config='true'>
   <intercept-url pattern="/**" access="ROLE_USER" />
</http>
```

Которые говорят о том, что мы хотим, чтобы в нашем приложение все URL-адреса были защищенными, для доступа к ним требуется роль `ROLE_USER`. Элемент `<http>` является родительским для всей функциональности связанной с веб доступом в пространстве имен security. Элемент `<intercept-url>` задает шаблон, с которым сравниваются URL-адресов входящих запросов, для шаблона используется синтаксис в стиле Ant path. Атрибут `access` определяет требования к правам доступа для запросов, совпадающих с данным шаблоном для доступа просит соответствующие заданной модели. В конфигурации по умолчанию, это как правило список ролей, разделенных запятыми, пользователь должен обладать одной из них, чтобы иметь возможность выполнить запрос. Префикс "ROLE_" является маркером, который показывает что нужно выполнить простое сравнение с полномочиями пользователя. Иными словами, будет использоваться выполняться обычная проверка, основанная на ролях пользователей. Контроль доступа в Spring Security не ограничивается использованием простых ролей (отсюда использования префикса, чтобы различать различные типы атрибутов безопасности). Позже мы увидим, как может меняться их интерпретация [2].

> ***Примечание:*** Вы можете использовать несколько элементов `<intercept-url>` для задания различных требований контроля доступа для различных наборов URL-адресов, но они будут обрабатываться в том порядке как они заданы в файле и будет использовано первое совпадение. Так что вы должны размещать наиболее важные шаблоны в самом начале списка. Вы также можете добавить атрибут `method` , чтобы ограничить совпадение конкретным видом HTTP запроса (GET, POST, PUT и т. д.). Если запрос соответствует нескольким шаблонам, то совпадение с конкретным типом запроса будет иметь приоритет над порядком расположения.

Чтобы добавить нескольких пользователей, можно задать тестовые данные прямо в пространстве имен:

```
<authentication-manager>
   <authentication-provider>
     <user-service>
```

```
      <user name="jimi" password="jimispassword" authorities="ROLE_USER, ROLE_ADMIN" />
      <user name="bob" password="bobspassword" authorities="ROLE_USER" />
    </user-service>
  </authentication-provider>
</authentication-manager>
```

В приведенной выше конфигурации, определены два пользователя, их пароли и роли в приложении (которые будут использоваться для контроля доступа). Кроме того, можно загрузить информацию о пользователе из стандартного файла свойств с использованием атрибута `properties` для тега `user-service`. См. раздел «Аутентификация in-memory» для более подробной информации о формате файла. Использование элемента `<authentication-provider>` означает, что это информация о пользователях будет использоваться менеджером аутентификации для обработки запросов аутентификации. Может иметься несколько элементов `<authentication-provider>` для задания нескольких источник аутентификационной информации, и каждый из них будет опрашиваться по очереди.

С этого момента вы можете запустить ваше приложение и получите запрос для выполнения аутентификации (логина). Попробуйте выполнить это или поэкспериментируйте с "учебным" приложением, которое поставляется вместе с проектом. Приведенная выше конфигурация, по факту добавляет не сильно много сервисов, потому что мы использовали атрибут `auto-config`. Например, автоматически включается обработка логина на основе веб-форм.

Что делает включение auto-config?

Атрибут `auto-config`, который мы уже использовали, это просто сокращенный синтаксис:

```
<http>
  <form-login />
  <http-basic />
  <logout />
</http>
```

Эти элементы отвечают соответственно за установку логина на основе веб-формы, базовый логин и выход из приложения [3]. Каждый из них имеет атрибуты, которые могут быть использованы для изменения их поведения.

Базовый вход в систему и вход на основе веб-форм

Вы можете быть удивлены когда при попытке войти в систему появится веб-форма для логина, так как мы ничего не говорили ни о каких HTML или JSP файлах. В действительности, поскольку мы не явно установили URL для страницы входа в систему, Spring Security будет генерировать ее автоматически, основываясь на доступных возможностях и используя стандартные значения для URL, который будет обрабатывать в ход в систему по умолчанию, после того как пользователь войдет в систему он будет перенаправлен на URL для которого был послан запрос. Тем не менее, пространство имен предлагает поддержку, которая позволяет настраивать эти параметры. Например, если вы хотите использовать свою собственную страницу для входа в систему, то вы можете использовать:

```
<http auto-config='true'>
  <intercept-url pattern="/login.jsp*" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
  <intercept-url pattern="/**" access="ROLE_USER" />
  <form-login login-page='/login.jsp'/>
</http>
```

Обратите внимание, что вы все еще можете использовать `auto-config`. Элемент `form-login` просто перекрывает настройки по умолчанию. Также отметим, что мы добавили дополнительный элемент `intercept-url`, что запросы к станице входа в систему были доступны для анонимных пользователей [4]. В противном случае запрос совпадет с шаблоном `/**` и доступ к странице входа в систему будет запрещен! Это распространенная ошибка конфигурации, которая ведет к бесконечному циклу в приложении. Spring

Security запишет предупреждение в лог если доступ к странице входа в систему будет закрыт системой безопасности. Возможно также, чтобы все запросы, соответствующие некоторому шаблону, шли в обход цепочки фильтров безопасности:

```
<http auto-config='true'>
  <intercept-url pattern="/css/**" filters="none"/>
  <intercept-url pattern="/login.jsp*" filters="none"/>
  <intercept-url pattern="/**" access="ROLE_USER" />
  <form-login login-page='/login.jsp'/>
</http>
```

Важно понимать, что эти запросы не будут ничего «знать» о конфигурациях Spring Security, связанных с веб-безопасностью или дополнительных атрибутах, таких как `requires-channel`, так что во время обработки такого запроса вы не сможете получить доступ к информации о текущем пользователе или вызвать защищенные методы. Используйте `access='IS_AUTHENTICATED_ANONYMOUSLY'`, как альтернативу, если вы хотите чтобы к запросу применялась цепочка фильтров безопасности.

Если вместо веб-формы вы хотите использовать базовую аутентификацию, то измените конфигурацию следующим образом:

```
<http auto-config='true'>
    <intercept-url pattern="/**" access="ROLE_USER" />
    <http-basic />
</http>
```

Базовая аутентификация будет иметь приоритет и будет выдавать запрос для логина, когда пользователь попытается получить доступ к защищенному ресурсу. В этой конфигурации по прежнему будет доступна веб-форма для выполнения логина, например если вы захотите чтобы пользователь выполнит аутентификацию через форму встроенную в другую веб-страницу.

Установка «места назначения» по умолчанию, после выполнения логина

Если при попытке получить доступ к защищенному ресурсу, пользователю не будет показана форма аутентификации, то тогда в игру вступает опция `default-target-url`. Пользователь после входа в систему будет отправлен по этому URL, по умолчанию это "/". Вы также можете настроить поведение таким образом, чтобы пользователь всегда перемещался на эту страницу (независимо от того выполнял он вход в систему "по требованию", либо явно вошел в систему), путем установки значения атрибута `always-use-default-target` в `"true"`. Это полезно, если ваше приложение требует чтобы пользователь всегда начинал работу с "домашней" страницы, например:

```
<http>
   <intercept-url pattern='/login.htm*' filters='none'/>
   <intercept-url pattern='/**' access='ROLE_USER' />
   <form-login login-page='/login.htm' default-target-url='/home.htm'
           always-use-default-target='true' />
</http>
```

Использование других провайдеров аутентификации

На практике могут потребоваться более масштабируемые источник информации о пользователях, чем несколько имен, добавленных в файл контекста приложения. Скорее всего, вы будете хранить информацию о пользователях в чем-то вроде базы данных или LDAP сервера. Конфигурирование с помощью пространства имен LDAP рассматривается в главе LDAP, поэтому мы не будем рассматривать его здесь. Если у вас есть собственная реализация интерфейса `UserDetailsService`, которая в контексте вашего приложения называется "myUserDetailsService", то вы можете выполнять аутентификацию с его помощью, используя

```
<authentication-manager>
  <authentication-provider user-service-ref='myUserDetailsService'/>
</authentication-manager>
```

Если вы хотите использовать базу данных, то тогда задайте:

```
<authentication-manager>
  <authentication-provider>
    <jdbc-user-service data-source-ref="securityDataSource"/>
  </authentication-provider>
</authentication-manager>
```

Где "securityDataSource" это имя бина `DataSource`, заданного в контексте приложения, указывающего на базу данных, содержащую стандартные таблицы данных Spring Security с информацией о пользователе. Кроме того, можно настроить бин Spring Security `JdbcDaoImpl` и указать его, используя атрибут `user-service-ref`:

```
<authentication-manager>
  <authentication-provider user-service-ref='myUserDetailsService'/>
</authentication-manager>

<beans:bean id="myUserDetailsService"
    class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">
  <beans:property name="dataSource" ref="dataSource"/>
</beans:bean>
```

Можно также использовать стандартный `AuthenticationProvider`:

```
<authentication-manager>
  <authentication-provider ref='myAuthenticationProvider'/>
</authentication-manager>
```

где `myAuthenticationProvider` имя бина в контексте вашего приложения, который реализует `AuthenticationProvider`. См. раздел 2.6, “Менеджер аутентификации и пространство имен” для получения дополнительной информации как конфигурировать `AuthenticationManager` в Spring Security с использованием пространства имен.

Добавление кодирования паролей

Часто данные пароля шифруются с помощью алгоритма хеширования. Эта возможность поддерживается элементом `<password-encoder>`. Конфигурация провайдера аутентификации с поддержкой алгоритма шифрования SHA будет выглядеть следующим образом:

```
<authentication-manager>
  <authentication-provider>
    <password-encoder hash="sha"/>
    <user-service>
      <user name="jimi" password="d7e6351eaa13189a5a3641bab846c8e8c69ba39f"
            authorities="ROLE_USER, ROLE_ADMIN" />
      <user name="bob" password="4e7421b1b8765d8f9406d87e7cc6aa784c4ab97f"
            authorities="ROLE_USER" />
    </user-service>
  </authentication-provider>
</authentication-manager>
```

Когда используются шифрованные пароли, то хорошей идеей будет «добавить соли» к паролю, что позволит защититься от атак с использованием словарей паролей, эта возможность так же поддерживается Spring Security. В идеале вы должны использовать случайным образом сгенерированные

значения «соли» для каждого пользователя, но можно использовать и любое из свойств объекта `UserDetails`, который загружается `UserDetailsService`.

```
<password-encoder hash="sha">
  <salt-source user-property="username"/>
</password-encoder>
```

Вы можете установить свой собственный шифратор паролей с помощью атрибута `ref` элемента `password-encoder`. Он должен содержать имя бина из контекста приложения, который является экземпляром Spring Security интерфейса `PasswordEncoder`.

Дополнительные возможности для веб

Аутентификация «Запомни меня»

Смотри отдельную главу «Аутентификация «Запомни меня» для получения информации о пространстве имен `remember-me`.

Добавление безопасности для HTTP/HTTPS канала

Если ваше приложение поддерживает доступ как по HTTP, так и по HTTPS, и вам требуется чтобы к отдельным URL можно было получить доступ только через HTTPS, то тогда можно воспользоваться атрибутом `requires-channel` в элементе `<intercept-url>`:

```
<http>
  <intercept-url pattern="/secure/**" access="ROLE_USER" requires-channel="https"/>
  <intercept-url pattern="/**" access="ROLE_USER" requires-channel="any"/>
  ...
</http>
```

Если будет использована данная конфигурация, то если пользователь попробует обратиться по протоколу HTTP по адресу, совпадающему с шаблоном, то он сперва будет перенаправлена на HTTPS URL. Возможные значения атрибута "http", "https" или "any". Использование значения "any"означает что можно использовать как HTTP так и HTTPS.

Управление сессиями

Обнаружение таймаута

Вы можете настроить Spring Security так, чтобы автоматически обнаруживать что присланный ID сессии является не корректным и перенаправлять пользователя на соответствующий URL. Это достигается с помощью элемента `session-management`:

```
<http>
  ...
  <session-management invalid-session-url="/sessionTimeout.htm" />
</http>
```

Управление параллельными сессиями

Если вы хотите ввести ограничения на способность одного пользователя войти в приложение, то Spring Security поддерживает эту возможность "из коробки" с помощью следующего простого дополнения. Прежде всего, необходимо добавить следующий слушатель в файл `web.xml`, чтобы Spring Security получал

информацию о событиях в течение жизненного цикла сессии:

```
<listener>
  <listener-class>
    org.springframework.security.web.session.HttpSessionEventPublisher
  </listener-class>
</listener>
```

Затем добавить следующие строки в контекст приложения:

```
<http>
  ...
  <session-management>
      <concurrency-control max-sessions="1" />
  </session-management>
</http>
```

Это предотвратит возможность многократного входа в систему одного пользователя — второй вход в систему приведет к тому, что предыдущий вход будет считаться недействительным. Вам может потребоваться предотвратить возможность выполнить второй вход в систему, в этом случае вы можете использовать:

```
<http>
  ...
  <session-management>
      <concurrency-control max-sessions="1" error-if-maximum-exceeded="true" />
  </session-management>
</http>
```

Попытка выполнить второй вход в систему будет отклонена. Под «отклонена» мы понимает, что пользователь будет отправлен на страницу `authentication-failure-url`, если используется аутентификация с помощью веб-форм. Если попытка выполнить повторную аутентификацию будет предпринята с помощью не интерактивного механизма, такого например, как «запомни меня», то клиенту будет отправлена ошибка 402 «не авторизован». Если вместо сообщения об ошибке вы хотите использовать собственную страницу для показа ошибок, то используйте атрибут `session-authentication-error-url` элемента `session-management`.

Если вы используете собственный фильтр аутентификации для входя в систему с помощью веб-форм, то вам придется явно настроить поддержку управления параллельными сессиями. Дополнительная информация содержится в главе «Управление сессиями».

Защита от атак «Исправление сессии»

Существует потенциальная опасность атаки «исправление сессии», когда злоумышленник создает сессию, а потом побуждает другого (легального) пользователя войти в систему с той же сессией (например, отправив ему ссылку содержащую идентификатор сессии в качестве параметра). Spring Security автоматически защищает от этих атак, путем создания новой сессии при входе пользователя. Если вам не нужна эта защита или она конфликтует с другим требованиями, вы можете управлять поведением защиты, используя атрибут `session-fixation-protection` элемента `<session-management>`, который имеет три опции

- migrateSession — создает новую сессию и копирует в нее атрибуты из существующей сессии. Это поведение по умолчанию.
- none — ничего не делает. Останется оригинальная сессия
- newSession — создает новую «чистую» сессию без копирования в нее данных из существующей сессии.

Поддержка OpenID

Пространство имен поддерживает использование входа в систему с помощью OpenID, вместо или совместно со входом в систему с помощью веб-формы, с помощью небольшого изменения конфигурации:

```
<http>
  <intercept-url pattern="/**" access="ROLE_USER" />
  <openid-login />
</http>
```

Затем вы должны зарегистрировать у себя OpenID провайдера (такого например, как myopenid.com), и добавить информацию о пользователе в in-memory `<user-service>`:

```
<user name="http://jimi.hendrix.myopenid.com/" authorities="ROLE_USER" />
```

Теперь вы можете войти в систему, используя сайт myopenid.com сайт для аутентификации. Кроме того, для работы с OpenID можно выбрать конкретный бин `UserDetailsService`, установив атрибут `user-service-ref` элемента `openid-login`. Для дополнительной информации см. предыдущий раздел об аутентификационных провайдерах. Обратите внимание, что в приведенной выше конфигурации, мы опустили атрибут `password`, так как этот набор данных пользователя используется только для загрузки полномочий пользователя. Случайный пароль генерируется внутри системы, предохраняя от случайного использования пользовательских данных как источника аутентификации в другом месте конфигурации.

Обмен атрибутами

Для OpenID поддерживается обмен атрибутами. Как пример, следующая конфигурация будет пытаться получить значения электронной почты и полного имени пользователя из провайдера OpenID и использовать их в своем приложении:

```
<openid-login>
  <attribute-exchange>
    <openid-attribute name="email" type="http://axschema.org/contact/email" required="true" />
    <openid-attribute name="name" type="http://axschema.org/namePerson" />
  </attribute-exchange>
</openid-login>
```

«Тип» каждого OpenID атрибута это URI, заданный определенной схемой, в нашем случае это http://axschema.org/. Если для успешной аутентификации какой-то атрибут должен быть получен в обязательном порядке, то тогда можно установить атрибут `required`. Поддерживаемая схема и набор возможных атрибутов зависят от вашего OpenID провайдера. Значения атрибутов возвращаются как часть процесса аутентификации и впоследствии могут быть доступными с помощью следующего кода:

```
OpenIDAuthenticationToken token = (OpenIDAuthenticationToken)SecurityContextHolder.getContext().getAuthentication();
List<OpenIDAttribute> attributes = token.getAttributes();
```

`OpenIDAttribute` содержит: тип атрибута и запрашиваемое значение (или значения, в случае если атрибут допускает множество значений). Мы сможет увидеть больше примеров использования класса `SecurityContextHolder` когда будет рассматривать ключевые компоненты Spring Security в главе «Технический обзор».

Добавление собственных фильтров

Если вы уже использовали Spring Security ранее, то вы знаете что фреймворк поддерживает цепочки фильтров, чтобы применять свои сервисы. Вы можете захотеть добавить в соответствующее место в стеке свой собственный фильтр или использовать Spring Security фильтр для которого пока еще нет конфигурационных опций в пространстве имен (например CAS). Или вы может быть захотите

использовать собственную реализацию стандартного фильтра, конфигурируемого пространством имен, такого как `UsernamePasswordAuthenticationFilter`, который создается элементом `<form-login>`, но при этом воспользоваться преимуществами, которые даются при явном использовании бинов. Как это можно сделать в конфигурации пространства имен, где цепочки фильтров не представлены явно?

Порядок фильтров всегда жестко задан при использовании конфигурирования с помощью пространства имен. Когда создается контекст приложения, код отвечающий за обработку пространства имен выполняет сортировку бинов фильтров и для каждого стандартного фильтра Spring Security известен его псевдоним и место в цепочке.

> ***Примечание:*** В предыдущих версиях сортировка выполнялась после того как экземпляры фильтров уже были созданы, в фазе пост-процессинга контекста приложения. Теперь, в версии 3.0+ сортировка выполняется на уровне метаданных бинов, перед тем как будут созданы экземпляры классов. Это имеет отношение к тому, как добавлять свои собственные фильтры. Т. к. список фильтров должен быть полностью известен во время разбора элемента `<http>`, то в версии 3.0 синтаксис немного изменился.

Фильтры, псевдонимы, и элементы/атрибуты пространства имен, которые создают фильтры, показаны в таблице 2.1 «Псевдонимы стандартных фильтров и их порядок». Фильтры перечислены в том порядке, в котором они располагаются в цепочке фильтров.

Псевдоним	Класс фильтра	Элемент или атрибут пространства имен
CHANNEL_FILTER	ChannelProcessingFilter	http/intercept-url@requires-channel
CONCURRENT_SESSION_FILTER	ConcurrentSessionFilter	session-management/concurrency-control
SECURITY_CONTEXT_FILTER	SecurityContextPersistenceFilter	http
LOGOUT_FILTER	LogoutFilter	http/logout
X509_FILTER	X509AuthenticationFilter	http/x509
PRE_AUTH_FILTER	AstractPreAuthenticatedProcessingFilter дочерние классы	N/A
CAS_FILTER	CasAuthenticationFilter	N/A
FORM_LOGIN_FILTER	UsernamePasswordAuthenticationFilter	http/form-login
BASIC_AUTH_FILTER	BasicAuthenticationFilter	http/http-basic
SERVLET_API_SUPPORT_FILTER	SecurityContextHolderAwareFilter	http/@servlet-api-provision
REMEMBER_ME_FILTER	RememberMeAuthenticationFilter	http/remember-me
ANONYMOUS_FILTER	AnonymousAuthenticationFilter	http/anonymous
SESSION_MANAGEMENT_FILTER	SessionManagementFilter	session-management
EXCEPTION_TRANSLATION_FILTER	ExceptionTranslationFilter	http
FILTER_SECURITY_INTERCEPTOR	FilterSecurityInterceptor	http
SWITCH_USER_FILTER	SwitchUserFilter	N/A

Вы можете добавить собственный фильтр в стек, используя элемент `custom-filter` и одно из этих имен, чтобы указать в какой позиции должен появиться ваш фильтр

```
<http>
    <custom-filter position="FORM_LOGIN_FILTER" ref="myFilter" />
 </http>

 <beans:bean id="myFilter" class="com.mycompany.MySpecialAuthenticationFilter"/>
```

Вы можете так же использовать атрибуты `after` или `before` если вы хотите чтобы ваш фильтр был вставлен после или перед другим фильтром в стеке. Имена "FIRST" и "LAST" могут быть использованы с атрибутом `position`, чтобы указать что вы хотите чтобы ваш фильтр появился перед или после всего стека соответственно.

Как избежать конфликта позиций фильтров

Если вы вставили пользовательский фильтр, который может занять позицию стандартного фильтра, созданного пространством имен, то тогда важно чтобы вы по ошибке не включили версию фильтра из пространства имен. Избегайте использования атрибута `auto-config` и удалите все элементы, которые могут создать фильтры, чью функциональность вы хотите заменить.

Обратите внимание, что вы не можете заменить фильтры которые создаются элементом `<http>` - `SecurityContextPersistenceFilter`, `ExceptionTranslationFilter` или `FilterSecurityInterceptor`.

Если вы замените фильтр пространства имен, который запрашивает точку входа аутентификации (т. е. там, где запускается процесс аутентификацsource lang="xml">

```
ии при попытке неаутентифицированного получить  доступ к защищенному ресурсу), вам также необходимо добавить пользовательск
```

Установка пользовательской AuthenticationEntryPoint

Если вы не используете для входа в систему веб-форму, или OpenID, или базовую аутентификации с помощью пространства имен, то вы можете определить фильтр аутентификации и точку входа с использованием обычного синтаксиса бинов и связать их с пространством имен, как мы только что видели. Соответствующий `AuthenticationEntryPoint` может быть установлен с помощью атрибута `entry-point-ref` элемента `<http>`.

Учебное приложение CAS является хорошим примером использования пользовательских бинов с пространством имен. Если вы не знакомы с точками входа аутентификации, то они будут обсуждаться в главе «Технический обзор».

Защита методов

Начиная с версии 2.0 в Spring Security улучшена поддержка безопасности методов, расположенных на сервисном уровне. Имеется поддержка как аннотаций безопасности JSR-250, так и оригинальных @Secured аннотаций фреймворка. Начиная с версии 3.0 вы можете также использовать новые, основанные на выражениях, аннотации. Вы можете применить систему безопасности к одному бину, используя элементы перехвата-методов при объявлении бина или можете защитить несколько бинов по всему уровню сервисов, используя точки срезов в стиле AspectJ.

Элемент <global-method-security>

Этот элемент используется чтобы включить возможность использования системы безопасности, основанной на аннотациях (путем установки соответствующих атрибутов у элементов), а также для того чтобы собрать в одном месте все объявления системы безопасности, которые будут действовать в контексте всего приложения. Вы должны только объявить один элемент `<global-method-security>`. Для включение поддержки Spring Security потребуется аннотация @Secured:

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

Добавление аннотации к методу (класса или интерфейса) будет соответственно ограничивать доступ к этому методу. Родная поддержка аннотаций Spring Security определяет несколько атрибутов для метода. Эти атрибуты затем передаются в `AccessDecisionManager`, чтобы принять фактическое решение:

```
public interface BankService {

  @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
  public Account readAccount(Long id);

  @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
  public Account[] findAccounts();

  @Secured("ROLE_TELLER")
  public Account post(Account account, double amount);
}
```

Поддержка аннотаций JSR-250 может быть включена с помощью:

```
<global-method-security jsr250-annotations="enabled" />
```

Они основаны на стандартах и позволяет легко создавать ограничения на основе ролей, которые будут применяться к методам, но они не имеют таких возможностей, как родные аннотации Spring Security. Чтобы воспользоваться новым синтаксисом, основанным на выражениях, вы должны использовать

```
<global-method-security pre-post-annotations="enabled" />
```

и Java код будет выглядеть

```
public interface BankService {

  @PreAuthorize("isAnonymous()")
  public Account readAccount(Long id);

  @PreAuthorize("isAnonymous()")
  public Account[] findAccounts();

  @PreAuthorize("hasAuthority('ROLE_TELLER')")
  public Account post(Account account, double amount);
}
```

Аннотации на основе выражений являются хорошим выбором, если вам нужно определить простые правила, но которые выходят за рамки проверки имен ролей на основе списка пользовательских полномочий. Вы можете использовать в одном приложении более одного типа аннотаций, но должны избегать смешивания аннотации разных типов в одном интерфейсе или классе, чтобы избежать путаницы.

Добавление срезов с помощью protect-pointcut

Особенно мощный инструмент, это использование `protect-pointcut`, т. к. это позволяет применить систему безопасности ко множеству бинов с помощью одного простого объявления. Рассмотрим следующий пример:

```
<global-method-security>
  <protect-pointcut expression="execution(* com.mycompany.*Service.*(..))"
       access="ROLE_USER"/>
</global-method-security>
```

Это защитит все методы в бинах, объявленных в контексте приложения, чьи классы расположены в пакете com.mycompany и чьи имена заканчиваются на "Service". Только пользователи с ролью ROLE_USER смогу вызывать эти методы. Как и в случае соответствия URL шаблону, самые важные шаблоны для поиска совпадений должны идти первыми в списке срезов, будет использовано первое выражение, которое совпадет с шаблоном.

AccessDecisionManager по умолчанию

В этом разделе предполагается, что вы уже имеете некоторые знания о базовой архитектуре управления доступом в Spring Security. Если у вас их нет, то вы можете пропустить его и вернуться к нему позже, содержание этого раздела имеет значение для тех, кому необходимо выполнять дополнительные настройки, чтобы получить более сложную систему безопасности, чем просто основанную на ролях пользователей.

Когда вы используете конфигурирование с помощью пространства имен, автоматически регистрируется экземпляр класса по умолчанию `AccessDecisionManager` создается и может использоваться для принятия решения при попытках вызова методов и доступа к URL, основываясь на ваших объявлениях в `intercept-url` и `protect-pointcut` (или в аннотациях, если вы используете защиту методов с помощью аннотаций).

Стратегия по умолчанию, это использовать `AffirmativeBased` `AccessDecisionManager` с `RoleVoter` и `AuthenticatedVoter`. Более подробно об этом можно прочитать в главе «Авторизация».

Настройка AccessDecisionManager

Основный интерфейс, который предоставляет в Spring Security услуги аутентификации, это `AuthenticationManager`. Обычно это экземпляр класса `ProviderManager`, с которым вы уже можете быть знакомы, если использовали фреймворк ранее. Если нет, то мы рассмотрим его позднее в главе «Технический обзор». Экземпляр бина регистрируется с помощью элемента пространства имен `authentication-manager`. Вы не можете использовать собственный экземпляр `AuthenticationManager` если вы пользуетесь системой безопасности методом или HTTP с помощью пространства имен, но это не проблема если у вас полный контроль над используемыми `AuthenticationProvider`'ами.

Вы можете захотеть зарегистрировать дополнительные бины `AuthenticationProvider` для `ProviderManager` и вы можете сделать это используя элемент `<authentication-provider>` с атрибутом `ref`, где значение атрибута это имя бина провайдера, который вы хотите добавить. Например:

```
<authentication-manager>
  <authentication-provider ref="casAuthenticationProvider"/>
</authentication-manager>

<bean id="casAuthenticationProvider"
    class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
  ...
</bean>
```

Другим общим требованием является то, что другой компонент в контексте может потребовать ссылку на `AuthenticationManager`. Вы можете легко зарегистрировать псевдоним для `AuthenticationManager` и использовать это имя в других местах контекста приложения.

```
<security:authentication-manager alias="authenticationManager">
   ...
</security:authentication-manager>

<bean id="customizedFormLoginFilter"
      class="com.somecompany.security.web.CustomFormLoginFilter">
   <property name="authenticationManager" ref="authenticationManager"/>
   ...
</bean>
```

Примечания

1. ↑ You can find out more about the use of the `ldap-server` element in the chapter on LDAP.
2. ↑ The interpretation of the comma-separated values in the access attribute depends on the implementation of the AccessDecisionManager which is used. In Spring Security 3.0, the attribute can also be populated with an EL expression.
3. ↑ In versions prior to 3.0, this list also included remember-me functionality. This could cause some confusing errors with some configurations and was removed in 3.0. In 3.0, the addition of an AnonymousAuthenticationFilter is part of the default <http> configuration, so the <anonymous /> element is added regardless of whether auto-config is enabled.
4. ↑ See the chapter on anonymous authentication and also the AuthenticatedVoter class for more details on how the value IS_AUTHENTICATED_ANONYMOUSLY is processed.