

# Spring Security

Составил: Черниговский А.С.  
Старший преподаватель кафедры “Информатика”  
ИКИТ СФУ

# Введение

- Spring Security – это фреймворк обеспечения безопасности, предоставляющий возможность декларативного управления безопасностью приложений на основе фреймворка Spring. Фреймворк Spring Security представляет собой всеобъемлющее решение по обеспечению безопасности, реализующее возможность аутентификации и авторизации как на уровне вебзапросов, так и на уровне вызовов методов.

# Обзор

- ACL
- CAS Client
- Configuration - Обеспечивает поддержку пространства имен XML
- Core - Основная библиотека Spring Security
- LDAP
- OpenID
- TagLibrary
- Web

# Использование конфигурационного пространства Spring Security

```
<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">
    <!-- Здесь располагаются элементы с префиксом security: -->
    </beans>
```

# Пространство имен security по умолчанию

```
<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">
  <!-- Здесь располагаются элементы без префикса security: -->

  </beans:beans>
```

# Подключение security.xml к web.xml

```
<context-param>  
  <param-name>contextConfigLocation</param-name>  
  <param-value>  
    /WEB-INF/security.xml  
  </param-value>  
</context-param>
```

# Безопасность веб-запросов

- Все взаимодействия с веб-приложениями на языке Java начинаются в компоненте `HttpServletRequest`. И коль скоро средством доступа к веб-приложению является запрос, то с него и следует начинать обеспечивать безопасность.
- Обычно настройка безопасности на уровне запросов начинается с объявления одного или более шаблонов URL-адресов, требующих некоторых привилегий, и ограничения доступа к содержимому по этим адресам для пользователей, не обладающих необходимыми привилегиями.
- Прежде чем ограничивать доступ пользователям, не обладающим необходимыми привилегиями, необходимо предусмотреть возможность определять, кто пользуется приложением. Поэтому приложение должно аутентифицировать пользователя, предложив ему идентифицировать себя.

# Сервлет-фильтры

`<filter>`

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

`<filter-class>`

`org.springframework.web.filter.DelegatingFilterProxy`

`</filter-class>`

`</filter>`

# DelegatingFilterProxy

- DelegatingFilterProxy – это специальный сервлет-фильтр, не имеющий самостоятельного значения, так как он просто делегирует фильтрацию реализации интерфейса `javax.servlet.Filter`, зарегистрированной в контексте приложения Spring в виде компонента



# Сервлет фильтры

- Чтобы задействовать фильтры из фреймворка Spring Security, в них необходимо внедрить какие-то другие компоненты. Однако в фильтры, зарегистрированные в файле web.xml, нельзя внедрить компоненты. Но с помощью DelegatingFilterProxy можно настроить фактические фильтры в Spring, воспользовавшись преимуществом поддержки внедрения зависимостей.
- Значение в элементе <filter-name> играет важную роль. Это имя будет использоваться для поиска компонента фильтра в контексте приложения Spring. Фреймворк Spring Security автоматически создаст компонент фильтра с идентификатором springSecurityFilterChain, потому что это имя указано в файле web.xml.

# FilterChainProxy

- Что касается самого компонента `springSecurityFilterChain`, это еще один специальный фильтр, известный как `FilterChainProxy`. Это единый фильтр, объединяющий в себе один или более других фильтров. Для обеспечения различных возможностей фреймворк Spring Security опирается на разные сервлет-фильтры. Но вам практически никогда не придется задумываться об этой особенности, потому что вам едва ли когда-нибудь потребуется явно объявлять компонент `springSecurityFilterChain` или любые другие фильтры, объединяемые в нем. Фреймворк Spring Security автоматически создает все необходимые компоненты при настройке элемента `<http>`, о чем рассказывается ниже.

# Минимальная настройка безопасности

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

# Аутентификация с помощью формы

- auto-config      приложение/login
- 
- <http auto-config="true" use-expressions="false">
- <form-login
- login-page="/mylogin"
- authentication-failure-url="/mylogin?error"/>
- </http>

# Аутентификация с помощью формы

- `<form-login`  
    `login-page="/mylogin"`  
    `authentication-failure-url="/login?error"`  
    `username-parameter="username"`  
    `password-parameter="password"/>`  
    `<csrf/>`
- `<input type="hidden" name="${_csrf.parameterName}"`
- `value="${_csrf.token}"/>`

# Завершение сеанса работы

- `<logout logout-url="/mylogout"/>`
- Необходимо отправить POST запрос по данному URL и будет выполнен logout

# Перехват запросов

- `<intercept-url pattern="/**" access="ROLE_USER" />`
- `<intercept-url pattern="/admin/**" access="ROLE_ADMIN" />`

# Настройка безопасности с применением выражений Spring

- `<http auto-config="true" use-expressions="true">`
- ...
- `</http>`

# Выражения, поддерживаемые Spring

- authentication - объект аутентификации пользователя
- denyAll - Всегда возвращает false
- hasAnyRole(list\_of\_roles) - true, если пользователь обладает какой-либо из привилегий, перечисленных в списке list\_of\_roles
- hasRole(role) - true, если пользователь обладает привилегией role
- hasIpAddress(IP Address) - IP-адрес пользователя (доступен только в веб-приложениях)
- isAnonymous() - true, если текущий пользователь не был аутентифицирован
- isAuthenticated() - true, если текущий пользователь был аутентифицирован
- isFullyAuthenticated() - true, если текущий пользователь был аутентифицирован и не использовал функцию «запомнить меня»
- isRememberMe() - true, если текущий пользователь был аутентифицирован автоматически
- permitAll - Всегда возвращает true
- principal - Основной объект, представляющий пользователя

# Принудительное использование протокола HTTPS

```
<intercept-url pattern="/form" requires-channel="https"/>
```

# Безопасность на уровне представлений

- `<security:accesscontrollist>` - Содержимое тега отображается, если текущий пользователь обладает одной из привилегий в указанном доменном объекте
- `<security:authentication>` - Обеспечивает доступ к свойствам объекта аутентификации текущего пользователя
- `<security:authorize>` - Содержимое тега отображается, если удовлетворяются указанные требования безопасности

# Подключение библиотеки тегов

- `<%@taglib prefix="security" uri="http://www.springframework.org/security/tags" %>`
- `<#assign security=JspTaglibs["http://www.springframework.org/security/tags"] />`

# Доступ к информации об аутентификации

Hello <security:authentication property="principal.username" />!

Для FreeMarker:

<security:authentication property="principal.username" />!

# Отображение с учетом привилегий

```
<body>  
  <h2>Welcome!</h2>  
  <@security.authorize access="hasRole('ROLE_ADMIN')">  
    <i>Это тайное админское сообщение</i>  
  </@security.authorize>  
</body>
```

# Пример

- 1) Отображаем ссылку только администраторам
- 2) Но если пользователь знает url он все еще может по ней перейти
- 3) Ограничиваем url на уровне обработки запросов
- 4) Но можно воспользоваться атрибутом authorize:

```
<security:authorize url="/admin/**">
```

```
    <spring:url value="/admin" var="admin_url" />
```

```
    <br/><a href="{admin_url}">Admin</a> </security:authorize>
```

# Аутентификация пользователей

- репозитория в памяти (настраиваемого в контексте приложения Spring);
- репозитория на основе JDBC;
- репозитория на основе LDAP;
- децентрализованных систем идентификации OpenID;
- централизованной системы аутентификации (Central Authentication System, CAS);
- сертификатов X.509; провайдеров на основе JAAS.

# Настройка репозитория в памяти

```
<user-service id="userService">
  <user name="admin" password=
"$2y$10$dZPGF39gkohmiVwAZQw.muH8W2anLtKwqMmGqWDbQlcdB30qF/NKe"
    authorities="ROLE_ADMIN"/>
  <user name="user" password=
"$2y$10$m83AE6lNYK8l2esyc9u5buZD2nY9lQ7/z7hOiy9nOzdOLnf1WHu1m"
    authorities="ROLE_USER"/>
</user-service>
<password-encoder hash="bcrypt"/>

<authentication-manager>
  <authentication-provider user-service-ref="userService" />
</authentication-manager>
```

# Аутентификация с использованием базы данных

- `<jdbc-user-service id="userService" data-source-ref="dataSource" />`
- `select username,password,enabled  
from users  
where username = ?`

# Аутентификация с использованием базы данных

- `<jdbc-user-service id="userService"  
data-source-ref="dataSource"  
users-by-username-query=  
"select login, password, true from  
public.&#34;User&#34;where login=?"  
authorities-by-username-query=  
"select login, 'ROLE_ADMIN' from  
public.&#34;User&#34; where login=? />`

# LDAP

- Не рассматривается, но достаточно важная тема. Например именно при помощи LDAP производится аутентификация в сервисах СФУ.

# Защита методов

- @PreAuthorize
- @PostAuthorize
- @PostFilter
- @PreFilter

# Проверка условия перед вызовом метода

- ```
@PreAuthorize("hasRole('ROLE_USER')")  
public void addComment(Comment comment) {  
    // ...  
}
```

# Проверка условия после вызова метода

- `@PostAuthorize("returnObject.user.username == principal.username")`  
`public Comment getCommentById(long id) {`  
`// ...`  
`}`

# Фильтрация после вызова метода

- `@PreAuthorize("hasRole('ROLE_USER')")`  
`@PostFilter("filterObject.user.username == principal.name")`  
`public List<Comment> getABunchOfComments() {`  
`...`  
`}`

Спасибо за внимание!