OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

18 – Spring Security

Дворжецкий Юрий

Как меня слышно && видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели:

- Разобраться в архитектуре самого замечательного и популярного фреймворка по безопасности
- После него, мы научимся создавать полностью приложения с логином и входом

Планы:

- Начался тот самый Enterprose
- И большие фреймворки требующии более тщательного ищучения.
- ДЗ нет!
- Задание будет на следующем.

Совсем чуть-чуть орг.вопросов:

- Уже продвинулся в проверке.
- Скоро и до Вас очередь дойдёт.
- Опять в планах за ночь-завтра раскидаться
- Со всех отзыв!

OTUS

Поехали?

OTUS

Безопасность приложений

Упражнение

- Какие задачи безопасности возникают в Enterprise приложении?

Задачи

- Пользователь – тот, за кого он себя выдаёт.
- Пользователю предоставляется доступ, только к тому функционалу, к которому он имеет доступ.

Упражнение

- Какие механизмы необходимы для решения этsих задач?

Механизмы

- Механизм подтверждения подлинности пользователя.
- Механизм предоставления/запрещения доступа пользователям.
- Механизм хранения прав доступа
- Механизм проверки прав доступа

Сами или Spring Security?

- Реализация механизмов самостоятельно – сложная задача.
- Механизмы от приложения очень похожи.
- Если микросервисы – нужно в каждый пихать это решение.
- А есть Spring Security

Spring Security

- Вот Spring Security и содержит реализацию механизмов безопасности

Spring Security

- Аутентификация – проверка что пользователь, тот за кого себя выдаёт
- Авторизация – проверка/предоставление пользователю прав доступа к объекту (объекты разные)

Механизмы аутентификации

- HTTP Basic authentication
- HTTP Digest authentication
- HTTP X.509
- Form-based authentication

Механизмы аутентификации

- OpenID
- LDAP
- JDBC/In-Memory хранение пользователей
- И есть интерфейсы для свой реализации

Механизмы авторизации

- По URL-ам
- Методы в сервисах
- По объектам

Чем не является

- Не является Firewall.
- Не является антивирусом.
- Не спасает приложение от различных атак – SQL Injection, XSS и т.д.
- Хотя есть защита от некоторых
- Не шифрует/обфусцирует.

Версии/Совместимость

- Раньше был совсем другим фреймворком
- Он, кстати, совсем другую архитектуру имеет
- Версионировался отдельно от Spring Framework
- Сейчас - 5.0.7.RELEASE
- Java 6+ для 4-ой, Java 8 для 5ой

Сайт проекта

Spring Security

https://projects.spring.io/spring-security/

Документация

Spring Security Reference

https://docs.spring.io/spring-security/site/docs/5.0.7.RELEASE/reference/htmlsingle/

Документация

http://www.mkyong.com/tutorials/spring-security-tutorials/

Вопросы?

Архитектура, основные компоненты

Spring Security

- **spring-security-core** – содержит основные абстракции
- **spring-security-web** – содержит дополнительные классы, для работы в Web-окружении (Servlets)
- **spring-security-config** – для описания конфигурации Spring Security с помощью Spring Security XML
- spring-security-web + spring-security-config = минимальный набор зависимостей

Spring Security

- spring-security-acl – содержит описание абстракций ACL (авторизация)
- spring-security-ldap – LDAP
- spring-security-openid – интеграция по протоколу OAuth 2.0 + OpenID Connect
- spring-security-test – классы для unit-тестирования

Упражнение

- Обзор приложения

- https://github.com/ydvorzhetskiy/spring-framework-18

Вопросы?

Немного про упражнения

- Я этот тренинг читал очно
- Предлагаю сделать также как читал
- Когда называю класс – смотрим его
- Shift два раза или Ctrl + N

Основные компоненты

- SecurityContext
- Authentication
- UserDetails
- UserDetailsService
- GrantedAuthority

SecurityContext

- Интерфейс SecurityContext
  - Отражает текущий контекст безопасности
  - Является контейнером для объекта типа Authentication

SecurityContextHolder

- MODE_THREADLOCAL
- MODE_INHERITABLETHREADLOCAL
- MODE_GLOBAL

Authentication

- Authentication – отражает информацию о текущем пользователе и его привилегиях

Authentication

- Principal:
  - В терминал логин-пароль – это логин
  - Отражает учетную запись пользователя
  - В Authentication представлено объектом типа Object
  - Зачастую реализуется объектом типа UserDetails

Authentication

- Свойство – Credentials:
- Подтверждает аутентичность пользователя
- В терминах логин-пароль – это пароль

Authentication

- Authorities – это права которые зарегистрированы:

Как происходит работа

Это один из flow (обычно многое другое)

- Пользователь отправляет логин-пароль (или много другое)
- Spring Security сзаполняет SecurityContextHolder

UserDetails

- UserDetails – абстрактное представление учетной записи пользователя

UserDetailsService

- UserDetailsService – интерфейс объекта, реализующего загрузку пользовательских данных из хранилища
- Реализации:
  - InMemoryDaoImpl
  - JdbcDaoImpl

Пример

```
@Bean
public UserDetailsService daoUserDetailsService(
    DataSource dataSource
) {
    JdbcDaoImpl dao = new JdbcDaoImpl();
    dao.setDataSource(dao);
    return dao;
}
```

AuthenticationManager

- AuthenticationManager – интерфейс объекта выполняющего аутентификацию

AuthenticationProvider

- AuthenticationProvider – интерфейс объекта, выполняющего аутентификацию
- Масса реализаций уже готовых
- Скорее всего будет залазить в UserDetails

PasswordEncoder

- PasswordEncoder :
  - ShaPasswordEncoder
  - Md5PasswordEncoder
  - Md4PasswordEncoder
  - PlaintextPasswordEncoder
- Какой из не Deprecated

AccessDecisionManager

- AccessDecisionManager – интерфейс объекта, который принимает решение о доступе к запрашиваемому ресурсу

AbstractSecurityInterceptor

- AbstractSecurityInterceptor:
  - Authentication из SecurityContextHolder
  - AuthenticationManager
  - AccessDecisionManager

SecurityInterceptors

- FilterSecurityInterceptor
  - Управляет доступом на уровне URL
  - Использует цепочку HTTP фильтров для управления доступом
- MethodSecurityInterceptor
  - Управляет доступом на уровне методов класса
  - Использует Spring AOP

Конфигурация Java-based

```
@EnableWebSecurity
@Configuration
public class ConfigClass extends
WebSecurityConfigurerAdapter {}
```

или

```
@EnableWebSecurity
@SpringBootApplication
public class App {}
```

Архитектура Spring Security

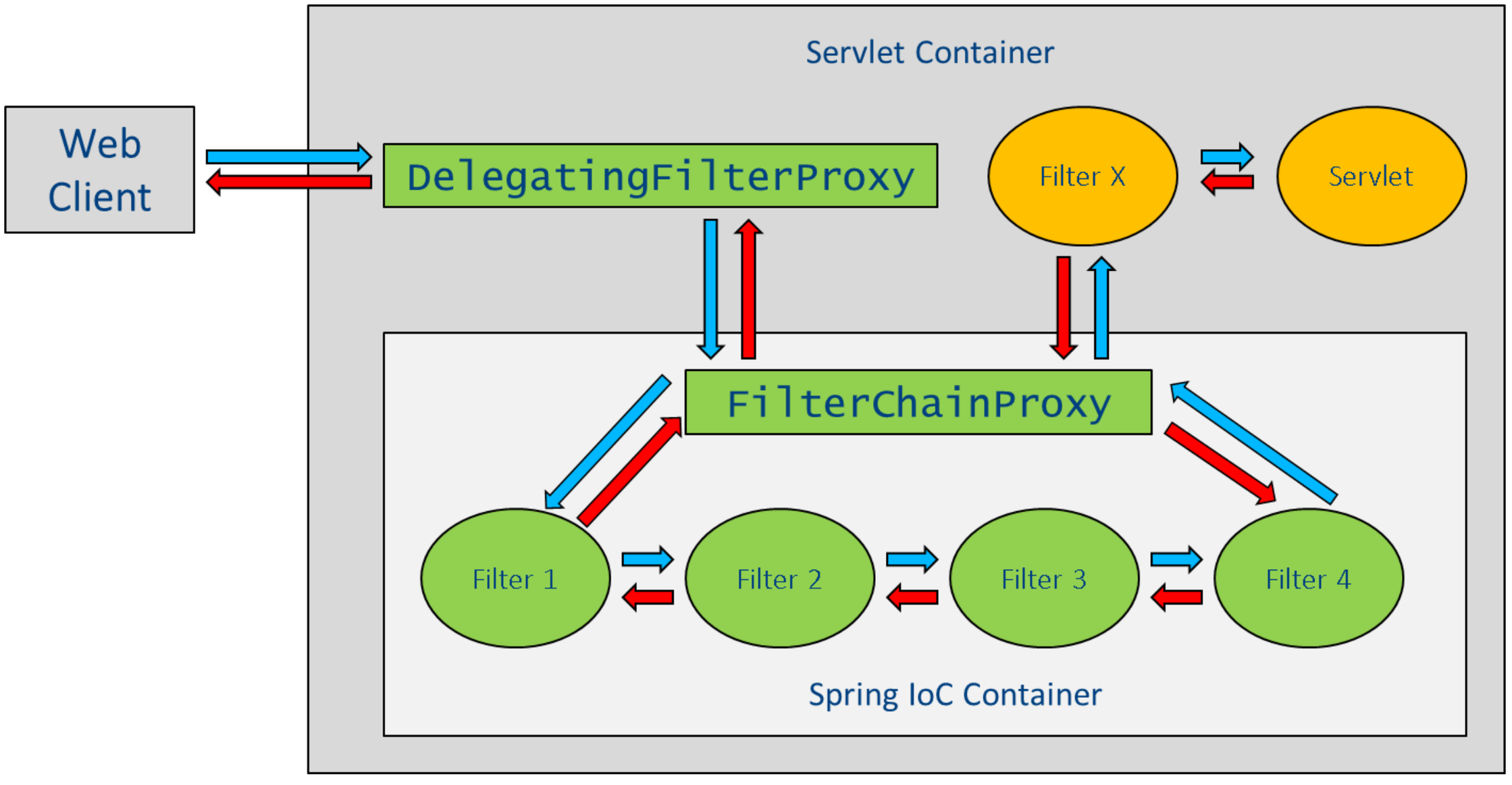
- При написание сложных приложений не всегда можно обойтись namespace-based или Java-based подходами
- Для тонкой настройки приходится использовать классический подход
- Их можно использовать одновременно в конфигурации приложения (зачастую так и бывает)

Вопросы?

Spring Security и Web

- Интеграция Spring Security со средой Web достигается через использование Servlet фильтров
- Фильтры объединены в цепочки
- Каждый фильтр реализует какой-то аспект механизма безопасности
- Важна последовательность фильтров в цепочке

Spring Security и Web



DelegatingFilterProxy

- DelegatingFilterProxy
  - Интегрирет цепочки фильтров в последовательность обработки HTTP запроса
  - Единственная задача – вызов цепочки фильтров Spring Security

DelegatingFilterProxy

```
@EnableWebSecurity
@Configuration
public class ConfigClass extends
                WebSecurityConfigurerAdapter {}
```

или

```
@EnableWebSecurity
@SpringBootApplication
public class App {}
```

FilterChain

```
@EnableWebSecurity
@Configuration
public class SpringSecurityConfig extends
        WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .csrf().disable()
                .authorizeRequests();
    }
}
```

Фильтры Spring Security

- Фильтры Spring Security реализует конкретные аспекты безопасности:
  - Совсем служебные
  - Аутентификация
  - Авторизация
  - Обработка ошибок

Фильтры Spring Security

- Цепочка фильтров:
  - В случае Java- based конфигурации создаётся неявно

Фильтры Spring Security

- ChannelProcessingFilter
- ConcurrentSessionFilter
- SecurityContextPersistenceFilter
- Фильтр(ы) аутентификации
- RememberMeAuthenticationFilter
- AnonymousAuthenticationFilter
- ExceptionTranslationFilter
- FilterSecurityInterceptor

Фильтры Spring Security

- ChannelProcessingFilter – если по HTTP, то перенаправляет на HTTPS
- ConcurrentSessionFilter – защита от повторяющихся сессий
- SecurityContextPersistenceFilter – создаёт Security-контекст
- Фильтр(ы) аутентификации – их тьма, пройдём на другом занятии

Фильтры Spring Security

- RememberMeAuthenticationFilter – вторичная аутентификация «Запомни меня»
- AnonymousAuthenticationFilter – тоже вторичная аутентификация
- ExceptionTranslationFilter – обрабатывает исключения
- FilterSecurityInterceptor – а вот он как раз занимается безопасностью

Вопросы?

Упражнение

- Упражнение № 2 смотрим в действии

Вопросы?

Домашнее задание

Нет

Вопросы?

Пожалуйста, пройдите опрос

https://otus.ru/polls/1754/

Спасибо
за внимание!

Security Вам!

OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

19 – Spring Security, Аутентификация

Дворжецкий Юрий

Как меня слышно && видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели:

- Строить приложения с аутентификацией
- Разбираться в механизмах аутентифкации

Планы:

- Basic Auth
- Form-based auth
- ДЗ есть

Совсем чуть-чуть орг.вопросов:

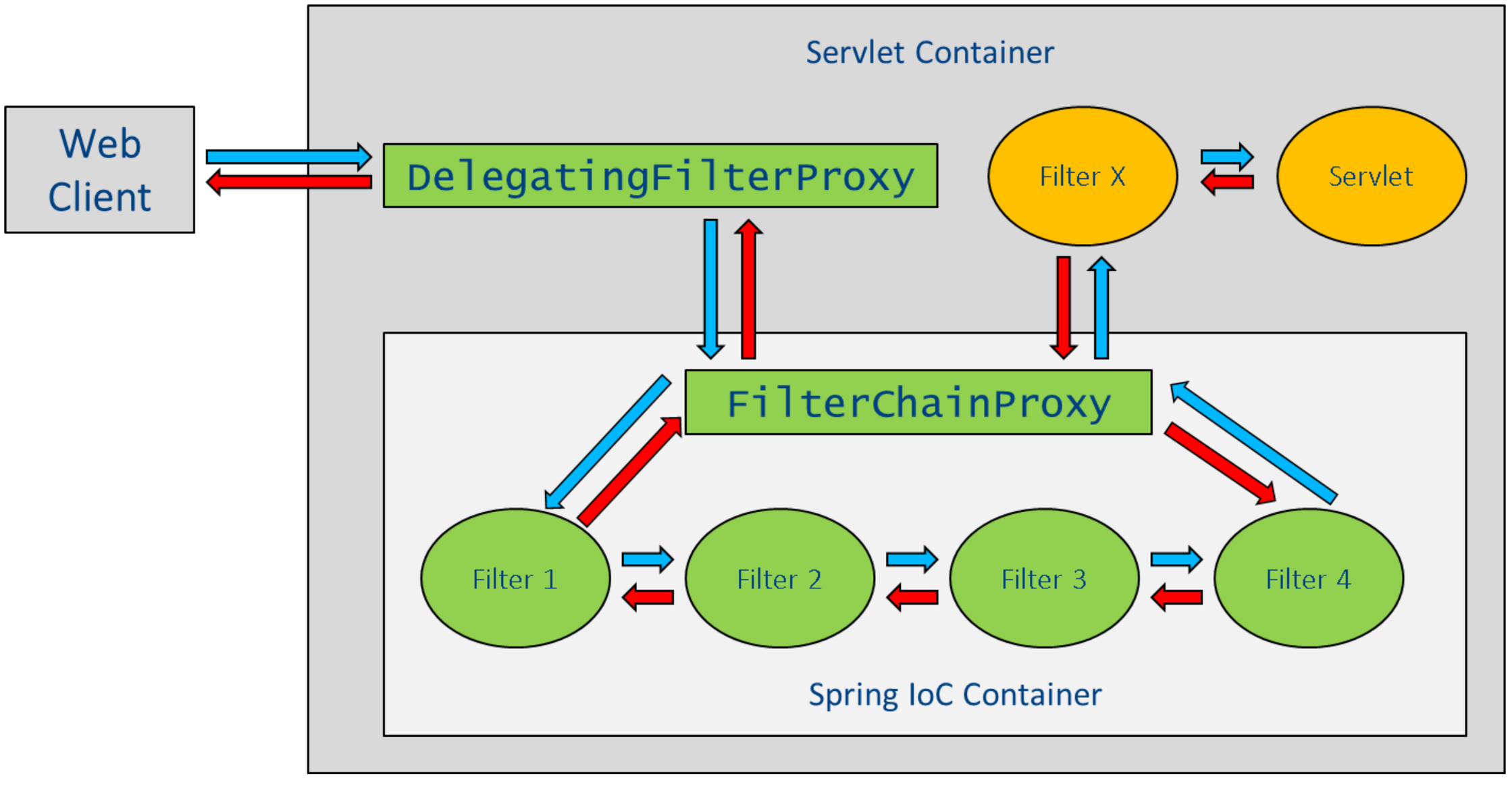
- Я проверяю!
- Мне осталось совсем немного.
- Опрос по перерыву в слаке.
- Со всех отзыв.

OTUS

Поехали?

Вспомним архитектуру Spring Security

Spring Security и Web



Фильтры Spring Security

- ChannelProcessingFilter
- ConcurrentSessionFilter
- SecurityContextPersistenceFilter
- Фильтр(ы) аутентификации
- RememberMeAuthenticationFilter
- AnonymousAuthenticationFilter
- ExceptionTranslationFilter
- FilterSecurityInterceptor

Фильтры создаются неявно с помощью DSL

```
@EnableWebSecurity
@Configuration
public class SpringSecurityConfig extends
        WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .csrf().disable()
                .authorizeRequests();
    }
}
```

Фильтры

- С некоторого момента оперируют Authentication в Security Context
- Кладут существующие данные – потом заменяют объект Authentication

Вопросы?

Аутентификация

Аутентификация

- Аутентификация – проверка что пользователь, тот за кого себя выдаёт

Аутентификация

- Аутентификация - один из основных механизмов Spring Security
- Есть и свои собственные
- А есть и уже существующие и известные

Механизмы аутентификации

- HTTP Basic authentication
- Form-based authentication
- HTTP X.509
- OpenID
- LDAP, AD

Аутентификация

- Стандартный сценарий аутентификации:
  - У пользователя запрашивается имя и пароль
  - Система проверят что пароль правильный
  - Система загружает информацию о пользователе
  - Устанавливается контекст безопасности
  - Пользователь обращается к ресурсам

в Spring Security

- Полученные имя и пароль с помощью Authenticationфильтра помещаются в UsernamePasswordAuthenticationToken (реализует Authentication) в SecurityContext

- Созданный AuthToken передаётся для проверки в AuthenticationManager через другие фильтры

- AuthenticationManager возвращает полностью инициализированный объект Authentication (последний FilterSecurityInterceptor)

в Spring Security

- Устанавливается контекст безопасности посредством вызова SecurityContextHolder. getContext().setAuthentication(...)

- С этого момента пользователь считается аутентифицированным

- Кстати достать можно в Spring MVC контроллере: SecurityContextHolder.getContext().getAuthentication()

Кладём (код уже готов)

```
String name = "<USERNAME>";
String password = "<PASSWORD>";

AuthenticationManager am = new SampleAuthenticationManager();
try {
    Authentication request =
            new UsernamePasswordAuthenticationToken(name, password);
    Authentication result = am.authenticate(request);

    SecurityContextHolder.getContext().setAuthentication(result);
} catch (AuthenticationException e) {
    System.out.println("Authentication failed.");
}
```

Проверяем (уже тоже готов)

```
class SampleAuthenticationManager implements AuthenticationManager {
    public Authentication authenticate(Authentication auth)
    throws AuthenticationException {

        Collection<GrantedAuthority> roles = Collections.emptyList();
        if (auth.getName().equals(auth.getCredentials())) {
            return new UsernamePasswordAuthenticationToken(
                    auth.getName(), auth.getCredentials(), roles);
        }

        throw new BadCredentialsException("Bad Credentials");
    }
}
```

Аутентификация в Web

- Клиент переходит по ссылке на фотку в контакте
- Запрос уходит на сервер, сервер решает, что незарегистрированный пользователь не может смотреть
- Сервер отправляет ответ, что нужна аутетификация

Аутентификация в Web

- Браузер в зависимости от механизма аутентификации:
  - переходит на Login Page
  - или другим способом собирает информацию о клиенте (BASIC Auth, X.509)
- Форма отправляет данные на сервер через HTTP тело или HTTP заголовки

Аутентификация в Web

- Сервер проверят Credentials:

- Сервер направляет клиента на первоначальную страницу вконтакте.

Вопросы?

Form-based аутентификация

- UsernamePasswordAuthenticationFilter – на сервер
- На клиенте реализуется web формой, содержащей поля:
  - Имя пользователя
  - Пароль
- Эта форма есть стандартная, а можете свою написать

Form-based аутентификация

```
@Override
protected void configure(
        HttpSecurity http) {
http
    .formLogin()
    .loginProcessingUrl(
        "/j_spring_security_check"
    )
    .usernameParameter("j_username")
    .passwordParameter("j_password")
    .loginPage("/public/login.jsp")
}
```

Демо

- https://github.com/ydvorzhetskiy/spring-framework-19

Вопросы?

Пара слов

- Логин форма настраивается
- Вы можете использовать свою
- Стратегии (куда перенаправлять – тоже настриваются)
- Это одна из самых гибких частей)
- Если не хочется использовать готовую – то можно свой метод- вполне реально пишется

Пример (+ костыль)

```
    @PostMapping("/auth")
    public AuthenticationResponseDto authenticate(
        @ApiParam(name = "Тело запроса", value = "Запрос на аутентификацию", required = true)
        @RequestBody AuthenticationRequestDto requestDto,
        HttpServletRequest request, HttpServletResponse response
    ) {
        String login = requestDto.getLogin();
        String password = requestDto.getPassword();
        Authentication authRequest = new UsernamePasswordAuthenticationToken(login, password);
        Authentication auth = authenticationManager.authenticate(authRequest);
        UserDetailsAdapter userDetails = (UserDetailsAdapter) auth.getPrincipal();
        rememberMeServices.loginSuccess(request, response, auth);
        return new AuthenticationResponseDto(
            userDetails.getUsername(), userDetails.getName(),
        );
    }
```

Упражнение

- Настроить аутентфикацию через веб-форму, чтобы при логине она перенаправляла на success, если не было страницы изначально

- * URL и страницу при failure

Вопросы?

HTTP Basic

- HTTP Basic Access аутентификация – это способ представления имени пользователя и пароля браузером или иным HTTP клиентом
- Является стандартной для протокола HTTP
- Определена в RFC-1945 (Hypertext Transfer Protocol – HTTP/1.0)
- Суть – передача закодированных в Base64 имени/пароля в виде HTTP заголовка Authorization

HTTP Basic

- Преимущества:
  - Простота механизма
  - Поддержка всеми браузерами
- Недостатки:
  - Имя пользователя и пароль передаются в открытом виде (необходимо использовать HTTPS)

HTTP Basic

- BasicAuthenticationFilter
- BasicAuthenticationFilter читает из заголовка HTTP запроса имя и пароль и использует для аутентификации
- BasicAuthenticationFilter соответствует RFC-1945

HTTP Basic

```
@Override

protected void configure(http) {
    http.httpBasic();

}



// В spring-boot-starter-web-security это
уже есть
```

Вопросы?

Упражнение

- Прикрутить basic-аутентификацию

Anonymous Authentication

- Есть открыте ресурсы
- Часто такие ресурсы исключаются из обработки цепочкой фильтров
- В результате не для всех запросов подсистема Spring Security будет не доступен заполненный SecurityContext
- А иногда код использует SecurityContext
- И получаются exceptions

Anonymous Authentication

- Anonymous Authentication – это типа паттерн NullObject
- (уже не Optional, скорее NaN в мире double)
- Основная идея – иметь пользователя с самыми дохлыми правами
- Такое есть в FTP, Гость в ОС и Chrome и т.д.

Anonymous Authentication

- Теперь можно ограничить доступ просто
- Иметь полностью доступную инфраструктуру Spring Security для всех запросов к системе

Anonymous Authentication

- AnonymousAuthenticationToken
  - Реализация интерфейса Authentication для анонимной аутентификации
  - Хранит список GrantedAuthority для анонимной аутентификации

Anonymous Authentication

- AnonymousAuthenticationFilter
  - Аутентифицирует пользователя как анонимного в случае если отсутствует другая аутентификация
  - Находится в цепочке после всех настроенных фильтров аутентификации

Anonymous Authentication

```
@Override
protected void configure(HttpSecurity http)
{
    http.httpBasic()
        .and()
        .anonymous()
        .authorities("ROLE_ANONYMOUS")
        .principal("anonymous");
}
```

Вопросы?

Упражнение

- Anonymous аутентификация (/public), anonymous – логин для анонима

- Вставтье код выводящий логин из authenticated

Вопросы?

Remember Me Authentication

- Remember Me аутентификация позволяет сохранять информацию о пользователе между HTTP сессиями

- Использование Remember Me аутентификации позволяет при очередном обращении к приложению использовать ранее введённые имя и пароль

- Обычно реализуется через сохранение cookie в браузере

Remember Me Authentication

- Simple Hash-Based Token подход (кстати переписываем обычно на JWT)
- Persistent Token подход

Simple Hash-Based Token

- После удачной аутентификации сохраняется cookie
- Cookie содержит закодированную в Base64 строку с:
  - Именем пользователя
  - Временем экспирации токена
  - Md5 от имени + пароль + время экспирации + ключ

Simple Hash-Based Token

```
base64(
username + ":" +  expirationTime + ":" +
md5Hex(
        username + ":"   +
        expirationTime + ":" + password
        + ":" + key
))
```

???

```
base64(
username + ":" +  expirationTime + ":" +
md5Hex(
        username + ":"   +
        expirationTime + ":" + password
        + ":" + key
))
```

Simple Hash-Based Token

```
.rememberMe()
.key("myAppKey")
.tokenValiditySeconds(60)
```

Persistent Token подход

- Альтернатива подходу Simple Hash-Based Token
- Основное отличие - содержимое Remember-Me cookie
- Для работы необходимо хранилище для Remember-Me токенов
- Токены в хранилище доступны по суррогатным ключам (значения из cookie)

На практике

- Хранить выданные токены иногда совсем затратно
- И не хочется лазить в базу за каждым запросом
- Мы лично использовали JWT (аналог simple-hash-based) чтобы проверять прямо в сервисах
- Но это не решает проблему логаута – поэтмоу хранили отозванные токены (пока не истечёт их реальный срок)

Вопросы?

Упражнение

- Simple hash-token-based

- И можете убрать сессию (раскоммнетировать вверху)

Вопросы?

Домашнее задание

В существующее CRUD-приложение добавить мехнизм Form-based аутентификации.

UsersServices реализовать самостоятельно.

Вопросы?

Пожалуйста, пройдите опрос

https://otus.ru/polls/1775/

Спасибо
за внимание!

Security Вам!

ОНЛАЙН-ОБРАЗОВАНИЕ

20 – Spring Security, Авторизация

Дворжецкий Юрий

Как меня слышно && видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели:

- Строить приложения с ролевой модель
- Разбираться в авторизации в Spring Security

Планы:

- Ролевая модель в Spring Security
- Основные классы
- Авторизация на основе урлов
- Авторизация на основе вызовов методов

Совсем чуть-чуть орг.вопросов:

- Осталось совсем чуть-чуть работ из завала☺
- Мы больше так не будем ☺
- Опрос по перерыву в слаке был – 14ого делаем перерыв.
- ДЗ сегодня нет.
- Со всех отзыв.

Поехали?

Вспомним Spring Security

Фильтры Spring Security

- ChannelProcessingFilter
- ConcurrentSessionFilter
- SecurityContextPersistenceFilter
- Фильтр(ы) аутентификации
- RememberMeAuthenticationFilter
- AnonymousAuthenticationFilter
- ExceptionTranslationFilter
- FilterSecurityInterceptor – вот здесь всё

Авторизация

- Авторизация - проверка/предоставление пользователю прав доступа к объекту

Авторизация

- Может осуществляться только после аутентификации
- Не обязательно она будет происходить сразу, она может происходить по мере обработки запроса

Авторизация

- Вот определение, которое было – бесполезное.
- Непонятно на каком основании принимается решение о доступе.
- И, собственно, к чему даётся доступ – тоже непонятно.

Права доступа

Spring Security поддерживает две модели прав доступа (на которых выдаётся разрешение). Эти модели и являются самыми распространёнными в мире IT

- Ролевая модель - у пользователей есть роли – сегодня её и пройдём.

- ACL (Access Control List) – очень навороченная модель, на основе бизнес-сущностей, ей посвятим целое занятие (аналогичная правам доступа к файлам и папкам в ОС)

Права доступа

- Они не являются взаимоисключающими – в приложении можете использовать обе.
- На самом деле есть ещё разные модели (больше теоретические), Spring Security из не поддерживает.
- Но всегда есть интерфейсы для реализации данных моделей.
- Обычно веб-приложение ограничиваются ролями, а если сложное приложение, то ещё ACL-ами.

Объект доступа

В Spring Security разграничение доступа происходит по:

- URL-ам страницы (API)
- Методам сервисов, контроллеров и т.д.
- Бизнес-сущностям (документам, пользвателям, счетам и т.д.) – это про ACL-ы

Опять же, Вы можете использовать их все сразу.

Вопросы?

Ролевая модель

Ролевая модель

- Роли представляют собой строчки (хотя есть возможность сделать и сложный объект)

- Когда пользователь вошёл, и где-то в БД они были записаны и вернулись в User Details (смотрим в IDEA), то роли теперь называются Granted Authority (так и называется интерфейс)

- По идее они динамические и записав в базу роль мы добавим права пользователя до следующей аутентификации.

Ролевая модель

Роли (строчки) в приложении называют так:

- ROLE_ADMIN, ROLE_ANONYMOUS, ROLE_USER
- ADMIN, USER… (вот такое пишется в некоторых служебных классах, ROLE_ добавляется автоматически)
- ROLE_CAN_PACK_BOX – в крутых приложениях

Ролевая модель

- Роли в Spring Security (только для ролевой модели) – Flat, т.е. не иерархичные и смысл мы в них вкладываем какой хотим (собственно строчки)

- Т.е. у Вас могут быть ROLE_ANONYMOUS, ROLE_USER, ROLE_ADMIN, но для Spring Security они ваще независимы.

- А какой смысл мы в это вкладывает – решаем мы сами.

Ролевая модель

- Это касается ролевой модели, а вот ACL – как раз наследуется.

Вопросы?

Авторизация в Spring Security

Авторизация

- Проверка прав осуществляется в AccessDecisionManager
- AccessDecisionManager при проверке прав опирается на GrantedAuthoritys
- Смотрим)

Авторизация

- Вот здесь можете как раз реализовать свою ролевую модель в AccessDecisionManager

- Реализации (это голосовалки)))):
  - AffirmativeBased
  - ConsensusBased
  - UnanimousBased - “default”

Они опрашивают AccessDecisionVoter для реализации проверки

Упражнение

- Придумайте где требуется
  - AffirmativeBased
  - ConsensusBased
  - UnanimousBased
  - Ваш Custom Voter

Авторизация

- Можно самостоятельно реализовывать логику проверки в виде AccessDecisionVoter
- Spring Security предлагает готовые реализации:
  - RoleVoter
  - AuthenticatedVoter
  - AclEntryVoter

Авторизация

- Класс RoleVoter голосует так:
  - есть роль – доступ разрешен
  - нет роли – доступ запрещен
- Класс AuthenticatedVoter выносит вердикт на основе типа аутентификации (обычная, remember-me, anonymous)

Авторизация в Web приложении

- В момент получения управления объектом FilterSecurityInterceptor пользователь уже аутентифицирован
- По умолчанию доступ есть у всех на все ресурсы
- Все паттерны ходят «сверху-вниз», срабатывает первый подошедший

DEMO

- https://github.com/ydvorzhetskiy/spring-framework-20

- Демо

Вопросы?

Упражение

- Давайте доабвим все проверки

Вопросы?

Авторизация для методов

- MethodSecurityInterceptor
- Это всё AOP – со всеми вытекающими последствиями (класс должен быть в контексте, внутренние обёртки и т.д.)

Авторизация для методов класса

```
@EnableGlobalMethodSecurity
@EnableWebSecurity
@Configuration
public class MySecurityConfig extends
GlobalMethodSecurityConfiguration {
    @Bean
    public MethodSecurityInterceptor
methodSecurityInterceptor() {
        return interceptor;
    }
}
```

Авторизация для методов класса

- 3 способа
- Аннотацию @Secured - простая
- Аннотации JSR-250
- Spring Security аннотации позволяющие поддерживающие EL выражения (самая круть)

Авторизация для методов класса

@EnableGlobalMethodSecurity есть атрибуты:

- pre-post-annotations
- jsr250-annotations
- secured-annotations

Все отключены по умолчанию.

Авторизация для методов класса

Spring Security аннотации, поддерживающие EL:

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

Авторизация для методов класса

```
@PreAuthorize("hasRole('ROLE_USER')")
public List<News> getNews() {
      ...
}

@Secured({"ROLE_USER", "ROLE_ADMIN"})
public List<News> getNews() {
      ...
}
```

Наворот для ACL
Угадайте что делает?

```
@PostFilter("hasPermission(filterObject, 'READ')")
public List<BankAccount> getBankAccounts(
                         Customer customer) {
    ...
}
```

Демо

- https://github.com/ydvorzhetskiy/spring-framework-20

Вопросы?

Упражнение

- Давайте сделаем доступ на уровне методов контроллера
- Не забудем дать админу досутуп к страницам пользователя

Вопросы?

Домашнее задание

НЕТ

Вопросы?

Пожалуйста, пройдите опрос

https://otus.ru/polls/1789/

Спасибо
за внимание!

Security Вам!

OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

21 – Spring Security, ACL

Дворжецкий Юрий

А я знаю, что меня видно и слышно)

Цели:

- Строить приложения с моделью авторизации на основе бизнес-сущностей
- Ну и, конечно, разбираться в модели ACL Spring.

Планы:

- Пара слов про ACL
- Как они устроены с точки зрения БД
- Основные классы Spring Security для ACL
- Авторизация на основе вызовов методов

OTUS

Поехали?

Вспомним Spring Security

Авторизация для методов класса

Spring Security аннотации, поддерживающие EL выражения для описания прав доступа:

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

Авторизация для методов класса

- @PreAuthorize позволяет определить необходимые права для вызова метода.

Авторизация для методов класса

```
@PreAuthorize("hasPermission(#customer, 'READ')")
public List<BankAccount> getBankAccounts(
                                Customer customer) {
      ...
}
@PreAuthorize("hasRole('ROLE_CUSTOMER')")
public List<BankAccount> getBankAccounts(
            Customer customer) {
      ...
}
```

Авторизация для методов класса

- @PreFilter выполняет фильтрацию коллекции объектов в параметрах метода
- Имя параметра для фильтрации определяется атрибутом filterTarget
- Имя filterObject используется для ссылки на объект фильтрации в EL выражении

Авторизация для методов класса

```
@PreFilter(
    value = "hasPermission(filterObject,'READ')",
    filterTarget = "customers"
)
public List<BankAccount> getBankAccounts(
                                List<Customer> customers) {
      ...
}
```

Авторизация для методов класса

- @PostAuthorize позволяет определить необходимые права на объект, возвращаемый методом
- Для ссылки на результат метода в EL выражении используется имя returnObject

Авторизация для методов класса

```
@PostAuthorize("hasPermission(returnObject, 'WRITE')")
public BankAccount getBankAccount() {
    ...
}
```

Авторизация для методов класса

- @PostFilter выполняет фильтрацию возвращаемого значения-коллекции
- Имя filterObject используется для ссылки на объект фильтрации в EL выражении
- Фильтрация поддерживается только для коллекций (массивы не фильтруются)

Авторизация для методов класса

- Для сравнения возможности аннотации @Secured ограничиваются только указанием списка ролей, необходимых для вызова метода

Авторизация для методов класса

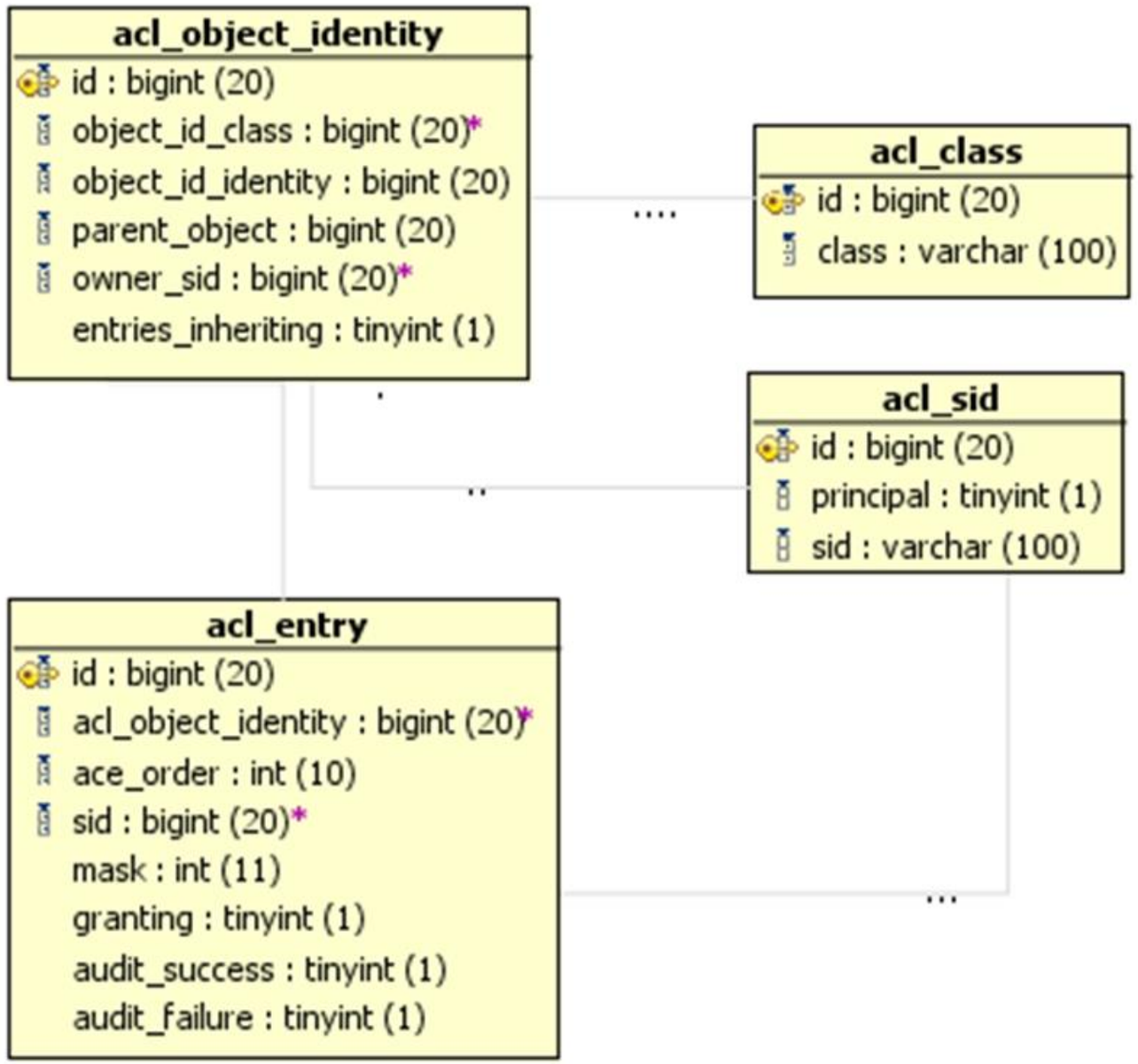
```
@PostFilter("hasPermission(filterObject, 'READ')")
public List<BankAccount> getBankAccounts(
                         Customer customer) {
    ...
}
```

Авторизация для методов класса

```
@Secured({"ROLE_ADMIN", "ROLE_USER"})
public BankAccount getBankAccount() {
     ...
}
```

Вопросы?

Domain Objects Security (ACL)



Domain Objects Security (ACL)

- До сих пор мы ограничивались проверкой только на уровне методов и URI ресурсов
- Spring Security так же поддерживает контроль доступа на уровне сущностей
- Эта возможность реализуется через поддержку ACL (Access Control List) сущностей
- ACL определяет кто и какие права имеет на объект указанного типа с указанным идентификатором

Domain Objects Security (ACL)

Spring Security предоставляет следующее:

- Возможность чтение ACL для всех сущностей
    - Оптимизированный механизм чтения
    - Использование кэширования
- Возможность проверки прав перед вызовом метода
- Возможность проверки прав после вызова метода

Domain Objects Security (ACL)

Spring Security хранит ACL в БД. Таблицы:

ACL_SID – позволяет идентифицировать security identity (роль или пользователь):

- id – суррогатный ключ
- principal – определяет тип security identity (0/1 – роль/имя пользователя)
- sid – содержит security identity

Domain Objects Security (ACL)

ACL_CLASS – идентифицирует тип сущности:

- id – суррогатный ключ
- class – содержит имя Java класса

Domain Objects Security (ACL)

ACL_OBJECT_IDENTITY – содержит информацию о всех сущностях системы

- id – суррогатный ключ
- object_id_class – ссылка на ACL_CLASS
- object_id_identity – идентификатор бизнес сущности
- parent_object – ссылка на родительский
- owner_sid – ссылка на ACL_SID определяющая владельца объекта

Domain Objects Security (ACL)

ACL_ENTRY – содержит права, назначенные для security identity на domain object

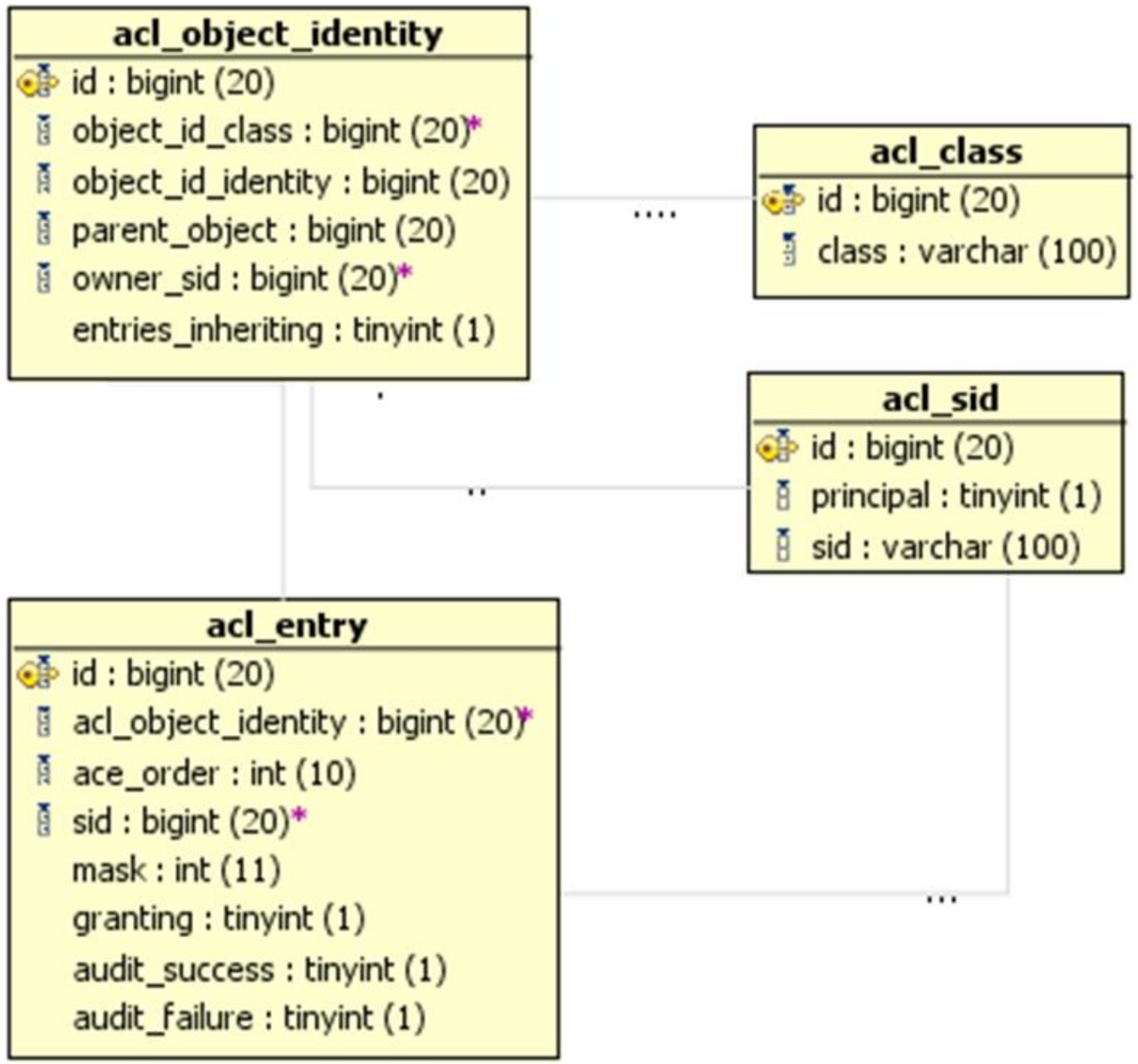
- id – суррогатный ключ
- acl_object_identity – ссылка на таблицу ACL_OBJECT_IDENTITY (что)
- ace_order – порядок применения записи
- sid – ссылка на ACL_SID (кому)
- mask – маска, определяющая права

Domain Objects Security (ACL)

ACL_ENTRY (продолжение):

- granting – определяет тип назначения (1 – разрешающее, 0 - запрещающее)
- audit_success – определяет будет ли записываться в лог информация об удачном применении ACE
- audit_failure - определяет будет ли записываться в лог информация об не удачном применении ACE

Domain Objects Security (ACL)



Domain Objects Security (ACL)

Spring Security использует битовые маски для обозначения права доступа (permission):

- bit 0 – read permission
- bit 1 – write permission
- bit 2 – create permission
- bit 3 – delete permission
- bit 4 – administer permission

Domain Objects Security (ACL)

Базовые интерфейсы подсистемы ACL:

- Acl
- MutableAcl
- AccessControlEntry
- Permission
- Sid
- ObjectIdentity
- AclService
- MutableAclService

Domain Objects Security (ACL)

- Интерфейс Acl:
  - представляет Access Control List для бизнес сущности
  - содержит список AccessControlEntry
    - имеет реализацию по умолчанию AclImpl
- Интерфейс MutableAcl:
  - расширяет интерфейс Acl предоставляя возможности модификации объекта ACL
  - имеет реализацию по умолчанию AclImpl

Domain Objects Security (ACL)

- Интерфейс AccessControlEntry:
  - представляет единичное разрешение
  - ссылается на Sid и Permission
    - имеет реализацию по умолчанию AccessControlEntryImpl
- Интерфейс Sid:
  - представляет security identity
  - имеет реализации GrantedAuthoritySid и PrincipalSid

Domain Objects Security (ACL)

- PrincipalSid – для формирования ACL для конкретного пользователя
- GrantedAuthoritySid - для формирования ACL для роли
- Интерфейс Permission:
  - представляет конкретное разрешение
  - содержит битовую маску
  - имеет реализации BasePermission и CumulativePermission

Domain Objects Security (ACL)

Константы для Permission:

- BasePermission.READ
- BasePermission.WRITE
- BasePermission.CREATE
- BasePermission.DELETE
- BasePermission.ADMINISTRATION

Domain Objects Security (ACL)

- Интерфейс ObjectIdentity:
  - представляет уникально идентифицируемую бизнес сущность
  - содержит идентификатор и тип сущности
  - реализация по умолчанию ObjectIdentityImpl
- Интерфейс ObjectIdentityGenerator определяет сервис для генерации (create()) ObjectIdentity

Domain Objects Security (ACL)

- Интерфейс ObjectIdentityRetrievalStrategy определяет сервис для получения (get()) ObjectIdentity для сущности
- Интерфейс AclService:
  - определяет сервис для загрузки ACL из хранилища
  - имеет реализацию JdbcAclService загружающую ACL из БД через JDBC

Domain Objects Security (ACL)

- Интерфейс MutableAclService:
  - расширяет интерфейс AclService предоставляя возможно по сохранению изменений в хранилище
    - реализация JdbcMutableAclService сохраняющая ACL в БД через JDBC
- Интерфейс AclCache:
  - определят кэш для ACL
  - используется в MutableAclService

Domain Objects Security (ACL)

- Интерфейс LookupStrategy:
  - определяет стратегию загрузки ACL из БД
  - используется в AclService и MutableAclService
  - реализация по умолчанию BasicLookupStrategy

Создание ACL

```
// создать SID-ы для владельца и пользователя
final Sid owner = new PrincipalSid(
      SecurityContextHolder.getContext()
            .getAuthentication());
final Sid admin = new GrantedAuthoritySid("ROLE_ADMIN");

// создать ObjectIdentity для бизнес сущности
final ObjectIdentity oid = new
ObjectIdentityImpl(BankAccount.class, 10);

// создать пустой ACL
final MutableAcl acl = aclService.createAcl(oid);
```

Создание ACL

```
// определить владельца сущности и права пользователей
acl.setOwner(owner);
acl.insertAce(acl.getEntries().size(),
BasePermission.READ, owner, true);
acl.insertAce(acl.getEntries().size(),
BasePermission.ADMINISTRATION, admin, true);

// обновить ACL в БД
aclService.updateAcl(acl);
```

Проверка ACL

```
// создать ObjectIdentity для бизнес сущности
final ObjectIdentity oid = new
ObjectIdentityImpl(BankAccount.class, 10);

// прочитать ACL бизнес сущности
Acl acl = aclService.readAclById(oid);

// определить какие права и для кого проверять
final List<Permission> permissions =
Arrays.asList(BasePermission.READ);

final List<Sid> sids = Arrays.asList((Sid) new
GrantedAuthoritySid("ROLE_ADMIN"));

// выполнить проверку
if (!acl.isGranted(permissions, sids, false)) {
    throw new RuntimeException("Access denied.");
}
```

Вопросы?

Домашнее задание

Ввести авторизацию на основе URL и/или доменных сущностей

Настроить в приложении авторизацию на уровне URL и/или доменных сущностей.

Вопросы?

Пожалуйста, пройдите опрос

https://otus.ru/polls/1823/

Спасибо
за внимание!

Security Вам!