Spring Security

**Это и остальное с полным описанием всех возможных элементов со всеми атрибутами и их параметрами (написанными в конце в Приложении) находится по адресу:**

[**https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#getting-started**](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#getting-started)

**и все существующие параметры для всех существующих атрибутов смотреть в последней главе - “Приложение".**

**Пример создания с объяснениями от официалов - https://spring.io/guides/gs/securing-web/**

* *4 основных стадии работы со Spring Security для Spring Boot - 2*
* [*Предисловие*](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#preface) *- 2*
* [*Начало*](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#getting-started) *работы - 3*
  + [1. Введение](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#introduction) - 3
    - [1.1. Что такое Spring Security?](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#what-is-acegi-security) - 3
    - [1.2. История](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#history) - 5
    - [1.3. Нумерация выпусков](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#release-numbering) - 6
    - [1.4. Получение Spring Security](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#get-spring-security) - 6
  + [2. Что нового в Spring Security 3.2](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#new) и в Безопасности Spring Boot 2 - 10
  + [3. Конфигурация Java](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc) (настройка безопасности в коде Java) - 12
    - [3.1. Конфигурация Java Hello Web Security](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#hello-web-security-java-configuration) - 10
    - [3.2. HttpSecurity](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc-httpsecurity) - 15
    - [3.3. Конфигурация Java и вход в форму](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc-form) - 17
    - [3.4. Запросы на авторизацию](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#authorize-requests) - 19
    - 3.5. Обработка выхода из системы - 19
    - [3.6. Аутентификация](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc-authentication) - 21
    - [3.7. Несколько HttpSecurity](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#multiple-httpsecurity) - 23
    - 3.8. Переход на HTTPS протокол - 24
    - [3.9. Безопасность метода](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc-method) - 28
    - [3.10. Настроенные объекты постобработки](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#post-processing-configured-objects) - 29
    - 3.11 Загрузка файлов (css, jpg и т.д.) из папок на титульную и др. страницы до авторизации - 30
  + [4. Конфигурация пространства имен безопасности](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-config) (настройка безопасности в ***security.xml***) - 30
    - [4.1. Введение](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#introduction-2) - 30
    - [4.2. Начало работы с конфигурацией пространства имен безопасности](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-getting-started) - 32
    - 4.2.1 web.xml Конфигурация - 32
    - 4.2.2 Минимальная конфигурация <http> - 32
    - 4.2.3 Форма и основные параметры входа  - 34
    - 4.2.4 Обработка выхода -  36
    - 4.2.5 Использование других провайдеров аутентификации  - 36
    - 4.2.6 Добавление кодировщика паролей - 37
    - [4.3. Расширенные возможности Интернета](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-web-advanced) - 38
    - 4.3.1 Remember-me – шифрование куков - 38
    - 4.3.2 Добавление безопасности канала HTTP / HTTPS - 39
    - 4.3.3 Управление сессией - 40
    - Обнаружение тайм-аутов - 40
    - Управление одновременным сеансом - 41
    - Защита от атак фиксации сеанса - 42
    - 4.3.4 Поддержка OpenID - 42
    - 4.3.5 Заголовки ответа для безопасности - 43
    - 4.3.6 Добавление собственных фильтров - 52
    - [4.4. Безопасность метода](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-method-security) - 55
    - [4.5. Менеджер AccessDecisionManager по умолчанию](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-access-manager) - 57
    - [4.6. Диспетчер аутентификации и пространство имен](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-auth-manager) - 58
  + [5. Примеры приложений](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#sample-apps) - 59
    - [5.1. Учебный образец](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#tutorial-sample) - 59
    - [5.2. Контакты](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#contacts-sample) - 59
    - [5.3. Пример LDAP](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ldap-sample) - 60
    - [5.4. Образец OpenID](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#openid-sample) - 60
    - [5.5. Образец CAS](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#cas-sample) - 61
    - [5.6. Образец JAAS](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jaas-sample) - 61
    - [5.7. Образец предварительной аутентификации](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#preauth-sample) - 61
    - [6. Тесты чтобы убедиться, что наша конфигурация безопасности применяется правильно](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#sample-apps)  - 61

Spring Security - это мощная и настраиваемая среда аутентификации и контроля доступа. Это де-факто стандарт защиты приложений на основе Spring.

**Как я понял существует 4 основных стадии работы со Spring Security для Spring Boot:**

1. Базовая полностью настроенная Spring Security - добавляем только зависимость в Maven. Это будет включать класс SecurityAutoConfiguration, содержащий начальную / стандартную конфигурацию безопасности (будут имя и пароль реально проверяться, созданы страница для аутентификации, страница где можно “разлогиниться” по адресу *logout*, пользователь с name=user, password=произвольно сгенерированный пароль, role=всем все)  Все значения по умолчанию см. в разделе свойств безопасности на справочной странице [общих свойств приложения Spring Boot](https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html) .
2. Базовая с небольшими изменениями – вносим доп. строчки, касающиеся изменений, в файл ***application.properties*** *или* ***application.yml*** (например spring.security.user.name=username, spring.security.user.password=password),
3. Собственная конфигурация безопасности с небольшими изменениями от стандартной/базовой – используется механизм Java Security (с Java кодом без XML). Мы в наш класс экстендим класс [***WebSecurityConfigurerAdapter***](https://www.baeldung.com/java-config-spring-security)*,* добавляем над классом аннотации @Configuration и @EnableWebSecurity и переопределяем в классе auth и http, с обязательным добавлением для **Spring Boot 2** и **Spring 5** PasswordEncoder. Но нет нормально написанной полной документации – только отрывочные статьи и JavaDoc.

!!! Класс WebMvcConfigurerAdapter который находился в интерфейсе WebMvcConfigurer и в котором “как в прокладке” находились все необходимые методы, объявлен устаревшим и все методы вынесены в сам интерфейс. И если мы используем Spring версии 5 (или Spring Boot 2), то появится предупреждение о “deprecated”. Поэтому выше должно быть написано “extends WebMvcConfigurer”

1. Собственная конфигурация безопасности со значительными изменениями и добавлениями функционала - используется механизм Пространства имен Безопасности. Используются XML файлы и нормально написанная полная документация (ссылка выше).

Предисловие

Spring Security предоставляет комплексное решение безопасности для корпоративных программных приложений на основе Java EE. Изучая это справочное руководство, вы обнаружите, что мы постарались предоставить вам полезную и легко настраиваемую систему безопасности.

Безопасность - это постоянно меняющаяся цель, и важно придерживаться комплексного общесистемного подхода. В кругах безопасности мы рекомендуем вам использовать «уровни безопасности», чтобы каждый уровень пытался быть как можно более безопасным сам по себе, а последовательные уровни обеспечивали дополнительную безопасность. Чем «жестче» безопасность каждого уровня, тем надежнее и безопаснее будет ваше приложение:

- На нижнем уровне вам нужно будет решить такие проблемы, как безопасность транспорта и идентификация системы, чтобы предотвратить атаки типа «злоумышленник в середине».

- Затем вы обычно используете брандмауэры, возможно, с VPN или IP-безопасностью, чтобы гарантировать, что только авторизованные системы могут пытаться подключиться. В корпоративных средах вы можете развернуть DMZ, чтобы отделить общедоступные серверы от серверной базы данных и серверов приложений.

- Ваша операционная система также будет играть важную роль, решая такие проблемы, как запуск процессов от имени непривилегированных пользователей и максимизируя безопасность файловой системы. Операционная система обычно также имеет собственный брандмауэр.

- Надеюсь, что где-то в процессе вы будете пытаться предотвратить отказ в обслуживании и атаки грубой силы на систему.

- Система обнаружения вторжений также будет особенно полезна для мониторинга атак и реагирования на них, поскольку такие системы могут принимать защитные меры, такие как блокирование вредоносных адресов TCP / IP в режиме реального времени.

- При переходе на более высокие уровни ваша виртуальная машина Java, надеюсь, будет настроена таким образом, чтобы минимизировать разрешения, предоставляемые различным типам Java, а затем ваше приложение добавит свою собственную конфигурацию безопасности для конкретной проблемной области.

Конечно, вам нужно будет правильно рассмотреть все уровни безопасности, упомянутые выше, вместе с управленческими факторами, которые охватывают каждый уровень. Неполный список таких управленческих факторов будет включать мониторинг бюллетеней безопасности, исправления, проверку персонала, аудит, контроль изменений, системы инженерного управления, резервное копирование данных, аварийное восстановление, сравнительный анализ производительности, мониторинг нагрузки, централизованное ведение журнала, процедуры реагирования на инциденты и т. д.

Поскольку Spring Security ориентирован на то, чтобы помочь вам с уровнем безопасности корпоративных приложений, вы обнаружите, что существует столько же различных требований, сколько и доменов бизнес-задач. Банковское приложение имеет другие потребности, чем приложение электронной коммерции. У приложения электронной коммерции другие потребности, чем у инструмента автоматизации корпоративных продаж. Эти специальные требования делают безопасность приложений интересной, сложной и полезной.

Пожалуйста, полностью прочтите « [Начало работы»](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#getting-started) . Это познакомит вас с фреймворком и системой конфигурации на основе пространства имен, с которой вы можете быстро приступить к работе. Чтобы лучше понять, как работает Spring Security, и некоторые классы, которые могут вам понадобиться, вам следует прочитать [Архитектура и реализация.](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#overall-architecture) Остальные части этого руководства построены в более традиционном справочном стиле, предназначенном для чтения по мере необходимости. Мы также рекомендуем вам прочитать как можно больше о проблемах безопасности приложений в целом. Spring Security - не панацея, которая решит все проблемы безопасности. Важно, чтобы приложение с самого начала разрабатывалось с учетом требований безопасности. Попытка модернизировать это - не лучшая идея. В частности, если вы создаете веб-приложение, вы должны знать о многих потенциальных уязвимостях, таких как межсайтовые сценарии, подделка запросов и захват сеанса, которые вы должны учитывать с самого начала. Веб-сайт OWASP ( [http://www.owasp.org/](https://www.owasp.org/)) содержит список десяти основных уязвимостей веб-приложений, а также много полезной справочной информации.

Мы надеемся, что это справочное руководство окажется для вас полезным, и будем рады вашим отзывам и [предложениям](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jira) .

Наконец, добро пожаловать в [сообщество](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#community) Spring Security .

Начало работы

В последующих частях этого руководства дается подробное обсуждение архитектуры фреймворка и классов реализации, которые вам необходимо понять, если вы хотите внести какие-либо серьезные изменения. В этой части мы представим Spring Security 3.0, дадим краткий обзор истории проекта и немного более осторожно рассмотрим, как начать использовать фреймворк. В частности, мы рассмотрим конфигурацию пространства имен, которая обеспечивает гораздо более простой способ защиты вашего приложения по сравнению с традиционным подходом к Spring bean-компонентам, в котором вам нужно подключить все классы реализации по отдельности.

Мы также рассмотрим доступные примеры приложений. Стоит попробовать запустить их и немного поэкспериментировать с ними еще до того, как вы прочитаете последующие разделы - вы можете вернуться к ним по мере того, как ваше понимание фреймворка будет расти. Также посетите [веб-сайт проекта, так](http://static.springsource.org/spring-security/site/index.html) как на нем есть полезная информация о создании проекта, а также ссылки на статьи, видео и учебные пособия.

1. Введение

1.1. Что такое Spring Security?

Spring Security предоставляет комплексные услуги безопасности для корпоративных программных приложений на основе Java EE. Особое внимание уделяется поддержке проектов, созданных с использованием Spring Framework, ведущего решения Java EE для разработки корпоративного программного обеспечения. Если вы не используете Spring для разработки корпоративных приложений, мы настоятельно рекомендуем вам присмотреться к нему. Некоторое знакомство со Spring - и в особенности с принципами внедрения зависимостей - поможет вам быстрее освоить Spring Security.

Люди используют Spring Security по многим причинам, но большинство из них обращается к проекту после того, как обнаруживает, что функциям безопасности Java EE Servlet Specification или EJB Specification не хватает глубины, необходимой для типичных сценариев корпоративных приложений. Упоминая эти стандарты, важно понимать, что они не переносимы на уровне WAR или EAR. Следовательно, если вы переключаете серверные среды, обычно требуется много работы по перенастройке безопасности вашего приложения в новой целевой среде. Использование Spring Security решает эти проблемы, а также предоставляет вам множество других полезных настраиваемых функций безопасности.

Как вы, наверное, знаете, двумя основными областями безопасности приложений являются «аутентификация» и «авторизация» (или «контроль доступа»). Это две основные области, на которые нацелена Spring Security. «Аутентификация» - по логину и паролю дается роль. «Авторизация» -ролям даются доступы к страницам. Чтобы прийти к точке, где требуется решение об авторизации, личность принципала уже была установлена ​​в процессе аутентификации. Эти концепции являются общими и никак не относятся к Spring Security.

На уровне аутентификации Spring Security поддерживает широкий спектр моделей аутентификации. Большинство из этих моделей аутентификации либо предоставляются третьими сторонами, либо разрабатываются соответствующими органами по стандартизации, такими как Инженерная группа Интернета. Кроме того, Spring Security предоставляет собственный набор функций аутентификации. В частности, Spring Security в настоящее время поддерживает интеграцию аутентификации со всеми этими технологиями:

* HTTP BASIC authentication headers (an IETF RFC-based standard)
* HTTP Digest authentication headers (an IETF RFC-based standard)
* HTTP X.509 client certificate exchange (an IETF RFC-based standard)
* LDAP (a very common approach to cross-platform authentication needs, especially in large environments)
* Form-based authentication (for simple user interface needs)
* OpenID authentication
* Authentication based on pre-established request headers (such as Computer Associates Siteminder)
* JA-SIG Central Authentication Service (otherwise known as CAS, which is a popular open source single sign-on system)
* Transparent authentication context propagation for Remote Method Invocation (RMI) and HttpInvoker (a Spring remoting protocol)
* Automatic "remember-me" authentication (so you can tick a box to avoid re-authentication for a predetermined period of time)
* Anonymous authentication (allowing every unauthenticated call to automatically assume a particular security identity)
* Run-as authentication (which is useful if one call should proceed with a different security identity)
* Java Authentication and Authorization Service (JAAS)
* JEE container autentication (so you can still use Container Managed Authentication if desired)
* Kerberos
* Java Open Source Single Sign On (JOSSO) \*
* OpenNMS Network Management Platform \*
* AppFuse \*
* AndroMDA \*
* Mule ESB \*
* Direct Web Request (DWR) \*
* Grails \*
* Tapestry \*
* JTrac \*
* Jasypt \*
* Roller \*
* Elastic Path \*
* Atlassian Crowd \*
* Your own authentication systems (see below)

(\* Обозначения предоставлены третьей стороной

Многие независимые поставщики программного обеспечения (ISV) принимают Spring Security из-за такого значительного выбора гибких моделей аутентификации. Это позволяет им быстро интегрировать свои решения со всем, что нужно их конечным клиентам, без необходимости выполнять много инженерных работ и не требуя от клиента изменения своей среды. Если ни один из вышеперечисленных механизмов аутентификации вам не подходит, Spring Security - открытая платформа, и довольно просто написать свой собственный механизм аутентификации. Многим корпоративным пользователям Spring Security необходимо интегрироваться с «устаревшими» системами, которые не соответствуют каким-либо конкретным стандартам безопасности, и Spring Security с радостью «хорошо играет» с такими системами.

Независимо от механизма аутентификации Spring Security предоставляет обширный набор возможностей авторизации.

Есть три основных направления, представляющих интерес:

- авторизация веб-запросов,

- проверка возможности вызова методов,

- авторизация доступа к отдельным экземплярам объекта домена.

Чтобы помочь вам понять различия, рассмотрим возможности авторизации, имеющиеся в безопасности веб-шаблона спецификации сервлета, управляемой безопасности EJB-контейнера и безопасности файловой системы соответственно. Spring Security предоставляет широкие возможности во всех этих важных областях, которые мы рассмотрим позже в этом справочном руководстве.

1.2. История

Spring Security началась в конце 2003 года как «Система безопасности Acegi для Spring». В списке рассылки разработчиков Spring был задан вопрос о том, уделялось ли какое-либо внимание реализации безопасности на основе Spring. В то время сообщество Spring было относительно небольшим (особенно по сравнению с сегодняшними размерами!), И действительно, сам Spring как проект SourceForge существовал только с начала 2003 года. Ответ на вопрос был таков: это полезная область, хотя ее и не хватает. времени в настоящее время препятствует его разведке.

Имея это в виду, простая реализация безопасности была построена и не выпущена. Несколько недель спустя другой член сообщества Spring спросил о безопасности, и в то время им был предложен этот код. Последовало еще несколько запросов, и к январю 2004 года код использовали около двадцати человек. К этим новаторским пользователям присоединились другие, которые предположили, что проект SourceForge в порядке, который был должным образом учрежден в марте 2004 года.

В то время у проекта не было собственных модулей аутентификации. В процессе аутентификации использовалась система безопасности, управляемая контейнером, а Acegi Security вместо этого фокусировалась на авторизации. Сначала это было подходящим вариантом, но по мере того, как все больше и больше пользователей запрашивали дополнительную поддержку контейнера, стало очевидным фундаментальное ограничение интерфейсов области аутентификации, специфичных для контейнера. Также существовала связанная проблема с добавлением новых JAR-файлов в путь к классам контейнера, что было обычным источником путаницы и неправильной настройки конечных пользователей.

Впоследствии были представлены службы аутентификации Acegi Security. Примерно через год Acegi Security стал официальным подпроектом Spring Framework. Финальный выпуск 1.0.0 был опубликован в мае 2006 года - после более чем двух с половиной лет активного использования в многочисленных производственных проектах программного обеспечения и многих сотен улучшений и вклада сообщества.

В конце 2007 года Acegi Security стал официальным проектом Spring Portfolio и был переименован в Spring Security.

Сегодня Spring Security пользуется сильным и активным сообществом разработчиков ПО с открытым исходным кодом. На форумах поддержки есть тысячи сообщений о Spring Security. Существует активное ядро ​​разработчиков, которые работают над самим кодом, и активное сообщество, которое также регулярно публикует исправления и поддерживает своих коллег.

1.3. Нумерация выпусков

Полезно понять, как работают номера выпусков Spring Security, поскольку это поможет вам определить усилия (или их отсутствие), связанные с переходом на будущие выпуски проекта.

В каждом выпуске используется стандартная тройка целых чисел: MAJOR . MINOR . PATCH.  Намерение состоит в том, что ОСНОВНЫЕ (MAJOR) версии являются несовместимыми крупномасштабными обновлениями API. Младшие (MINOR) версии должны в основном сохранять исходную и двоичную совместимость со старыми второстепенными версиями, хотя могут быть некоторые изменения дизайна и несовместимые обновления. Уровень PATCH должен быть полностью совместим, вперед и назад, с возможным исключением изменений, которые предназначены для исправления ошибок и дефектов.

Степень влияния изменений на вас будет зависеть от того, насколько тесно интегрирован ваш код. Если вы выполняете много настроек, это может повлиять на вас больше, чем при использовании простой конфигурации пространства имен.

Вы всегда должны тщательно тестировать свое приложение перед развертыванием новой версии.

1.4. Получение Spring Security

Вы можете получить Spring Security несколькими способами:

- загрузить пакетный дистрибутив с главной страницы [Spring Security](https://spring.io/spring-security) ,

- загрузить отдельные jar-файлы из репозитория Maven Central (или репозитория SpringSource Maven для моментальных снимков и промежуточных выпусков),

- самостоятельно собрать проект из исходного кода.

1.4.1. Использование с Maven

Минимальный набор зависимостей Spring Security Maven обычно выглядит следующим образом:

**pom.xml**

<dependencies>

<!-- ... other dependency elements ... -->

<dependency>

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

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

<version>3.2.5.RELEASE</version>

</dependency>

<dependency>

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

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

<version>3.2.5.RELEASE</version>

</dependency>

</dependencies>

Если вы используете дополнительные функции, такие как LDAP, OpenID и т.д., Вам также необходимо включить соответствующие [модули проекта](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#modules) .

Репозитории Maven

Все выпуски GA (то есть версии, оканчивающиеся на .RELEASE) развернуты в Maven Central, поэтому не нужно объявлять дополнительные репозитории Maven в вашем pom.

Если вы используете версию SNAPSHOT, вам необходимо убедиться, что у вас есть репозиторий Spring Snapshot, как показано ниже:

**pom.xml**

<repositories>

<!-- ... possibly other repository elements ... -->

<repository>

<id>spring-snapshot</id>

<name>Spring Snapshot Repository</name>

<url>http://repo.springsource.org/snapshot</url>

</repository>

</repositories>

Если вы используете веху или версию-кандидат на выпуск, вам необходимо убедиться, что у вас есть репозиторий Spring Milestone, как показано ниже:

**pom.xml**

<repositories>

<!-- ... possibly other repository elements ... -->

<repository>

<id>spring-milestone</id>

<name>Spring Milestone Repository</name>

<url>http://repo.springsource.org/milestone</url>

</repository>

</repositories>

Использование Spring 4 и Maven

Spring Security строится на основе Spring Framework 3.2.8.RELEASE, но также тестируется на Spring Framework 4.0.2.RELEASE. Это означает, что вы можете использовать Spring Security 3.2.5.RELEASE с Spring Framework 4.0.2.RELEASE.  Проблема, с которой столкнутся многие пользователи, заключается в том, что транзитивные зависимости Spring Security разрешают Spring Framework 3.2.8.RELEASE, вызывая всевозможные странные проблемы с путями к классам.

Один (утомительный) способ обойти эту проблему - включить все модули Spring Framework в раздел [<dependencyManagement>](https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Dependency_Management) вашего файла pom.

Альтернативный подход - включить spring-framework-bom в свой <dependencyManagement> раздел, pom.xml как показано ниже:

**pom.xml**

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-framework-bom</artifactId>

<version>4.0.2.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Это гарантирует, что все транзитивные зависимости Spring Security будут использовать модули Spring 4.0.2.RELEASE.

|  |  |
| --- | --- |
|  | Этот подход использует концепцию «ведомости материалов» (BOM) Maven и доступен только в Maven 2.0.9+. Дополнительные сведения о том, как разрешаются зависимости, см. [В разделе Maven Introduction to the Dependency Mechanism documentation](https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html) . |

1.4.2. Gradle

Минимальный набор зависимостей Spring Security Gradle обычно выглядит следующим образом:

**build.gradle**

dependencies {

compile 'org.springframework.security:spring-security-web:3.2.5.RELEASE'

compile 'org.springframework.security:spring-security-config:3.2.5.RELEASE'

}

Если вы используете дополнительные функции, такие как LDAP, OpenID и т. Д., Вам также необходимо включить соответствующие [модули проекта](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#modules) .

Репозитории Gradle

Все выпуски GA (т.е. версии, заканчивающиеся на .RELEASE) развертываются в Maven Central, поэтому для выпусков GA достаточно использовать репозиторий mavenCentral ().

**build.gradle**

repositories {

mavenCentral()

}

Если вы используете версию SNAPSHOT, вам необходимо убедиться, что у вас есть репозиторий Spring Snapshot, как показано ниже:

**build.gradle**

repositories {

maven { url 'https://repo.spring.io/snapshot' }

}

Если вы используете веху или версию-кандидат на выпуск, вам необходимо убедиться, что у вас есть репозиторий Spring Milestone, как показано ниже:

**build.gradle**

repositories {

maven { url 'https://repo.spring.io/milestone' }

}

Использование Spring 4 и Gradle

По умолчанию Gradle будет использовать самую новую версию при разрешении переходных версий. Это означает, что часто при запуске Spring Security 3.2.5.RELEASE с Spring Framework 4.0.2.RELEASE дополнительная работа не требуется. Однако иногда могут возникать проблемы, поэтому лучше всего смягчить их с помощью [Gradle ResolutionStrategy,](http://www.gradle.org/docs/current/dsl/org.gradle.api.artifacts.ResolutionStrategy.html) как показано ниже:

**build.gradle**

configurations.all {

resolutionStrategy.eachDependency { DependencyResolveDetails details ->

if (details.requested.group == 'org.springframework') {

details.useVersion '4.0.2.RELEASE'

}

}

}

Это гарантирует, что все транзитивные зависимости Spring Security будут использовать модули Spring 4.0.2.RELEASE.

|  |  |
| --- | --- |
|  | В этом примере используется Gradle 1.9, но могут потребоваться модификации для работы в будущих версиях Gradle, поскольку это инкубационная функция в Gradle. |

1.4.3. Модули проекта

В Spring Security 3.0 кодовая база была разделена на отдельные jar-файлы, которые более четко разделяют различные функциональные области и сторонние зависимости. Если вы используете Maven для сборки своего проекта, то это те модули, которые вы добавите в свой pom.xml. Даже если вы не используете Maven, мы рекомендуем вам ознакомиться с pom.xmlфайлами, чтобы получить представление о сторонних зависимостях и версиях.  В качестве альтернативы хорошей идеей является изучение библиотек, включенных в примеры приложений.

Ядро - spring-security-core.jar

Содержит Базовую аутентификацию и классы и интерфейсы управления доступом, поддержку удаленного взаимодействия и базовые API инициализации. Требуется любым приложением, использующим Spring Security. Поддерживает автономные приложения, удаленных клиентов, безопасность методов (уровня услуг) и инициализацию пользователей JDBC. Содержит пакеты верхнего уровня:

* org.springframework.security.core
* org.springframework.security.access
* org.springframework.security.authentication
* org.springframework.security.provisioning

Удаленное взаимодействие - spring -security-remoting.jar

Обеспечивает интеграцию с Spring Remoting.  Вам это не нужно, если вы не пишете удаленный клиент, который использует Spring Remoting.  Основной пакет есть org.springframework.security.remoting.

Интернет - spring -security-web.jar

Содержит фильтры и соответствующий код инфраструктуры веб-безопасности. Все, что связано с зависимостью API сервлета. Он вам понадобится, если вам требуются службы веб-аутентификации Spring Security и управление доступом на основе URL.  Основной пакет есть org.springframework.security.web.

Конфиг - spring-security-config.jar

Содержит код анализа пространства имен безопасности. Он вам понадобится, если вы используете для конфигурации пространство имен Spring Security XML.  Основной пакет есть org.springframework.security.config. Ни один из классов не предназначен для прямого использования в приложении.

LDAP - spring-security -ldap.jar

Код аутентификации и инициализации LDAP. Требуется, если вам нужно использовать аутентификацию LDAP или управлять записями пользователей LDAP. Пакет верхнего уровня - это org.springframework.security.ldap.

ACL - spring-security -acl.jar

Реализация ACL для специализированных доменных объектов. Используется для применения безопасности к конкретным экземплярам объекта домена в вашем приложении. Пакет верхнего уровня - это org.springframework.security.acls.

CAS - пружина-безопасность-cas.jar

Интеграция клиента CAS Spring Security. Если вы хотите использовать веб-аутентификацию Spring Security с сервером единого входа CAS.  Пакет верхнего уровня - это org.springframework.security.cas.

OpenID - spring-security -openid.jar

Поддержка веб-аутентификации OpenID. Используется для аутентификации пользователей на внешнем сервере OpenID. org.springframework.security.openid. Требуется OpenID4Java.

1.4.4. Проверка источника

Поскольку Spring Security - это проект с открытым исходным кодом, мы настоятельно рекомендуем вам проверить исходный код с помощью git. Это даст вам полный доступ ко всем примерам приложений, и вы сможете легко создать самую последнюю версию проекта. Наличие исходного кода проекта также очень помогает при отладке. Трассировки стека исключений больше не скрывают проблемы с черным ящиком, но вы можете сразу перейти к строке, которая вызывает проблему, и разобраться, что происходит.  Источник - это основная документация для проекта и часто самое простое место, чтобы узнать, как что-то на самом деле работает.

Чтобы получить исходный код проекта, используйте следующую команду git:

git clone https://github.com/spring-projects/spring-security.git

Это даст вам доступ ко всей истории проекта (включая все выпуски и ветки) на вашем локальном компьютере.

2. Что нового в *Spring Security 3.2*

и в

*Безопасности Spring Boot 2*

В выпуске Spring Security 3.2 [разрешено более 150 обращений](https://jira.springsource.org/issues/?jql=project%20%3D%20SEC%20AND%20fixVersion%20in%20(%223.2.0.RC2%22%2C%20%223.2.0%22%2C%20%223.2.0.RC1%22%2C%20%223.2.0.M2%22%2C%20%223.2.0.M1%22)%20ORDER%20BY%20priority%20DESC%2C%20issuetype%20ASC%2C%20key%20DESC) . Ниже приведены основные характеристики новых функций Spring Security 3.2.

* [Поддержка конфигурации Java](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc)
* [Защита от подделки межсайтовых запросов (CSRF)](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#csrf) (механизм, при котором с каждой сессией пользователя ассоциируется дополнительный секретный уникальный ключ, предназначенный для выполнения запросов. Секретный ключ не должен передаваться в открытом виде, например, для [POST](https://ru.wikipedia.org/wiki/POST_(HTTP))-запросов ключ следует передавать в теле запроса, а не в адресе страницы. Браузер пользователя посылает этот ключ в числе параметров каждого запроса, и перед выполнением каких-либо действий сервер проверяет этот ключ.)

В случае применения на клиенте, для передачи информации на сервер, методов POST,PUT,DELETE(а не только GET) (кроме того есть еще метод передачи информации через ***формы***), чтобы эти методы CSRF-защита (автоматически запускающаяся с spring security) пропустила, у официалов написано, что нужно дописать следующий код в каждый такой JavaScript-метод и html-лист:

//!!!Не работающий метод разрешения CSRF-защитой механизмов POST,PUT,DELETE на страницах!!!!

// Эту строку дописать html-лист <meta name="csrf-token" content="{{ csrf\_token() }}">  
// Эти строки в скрипт  
//var csrf\_token = $('meta[name="csrf-token"]').attr('content');  
//  
// function csrfSafeMethod(method) {  
// // these HTTP methods do not require CSRF protection  
// return (/^(GET|HEAD|OPTIONS)$/.test(method));  
// }  
  
// $.ajaxSetup({  
// beforeSend: function(xhr, settings) {  
// if (!csrfSafeMethod(settings.type) && !this.crossDomain) {  
// xhr.setRequestHeader("anti-csrf-token", csrf\_token);  
// }  
// }  
// });

Можно все передавать через разрешенный метод GET, но в нем вся инфа передается в URl, а не в теле запроса

* [Нажмите Защита от взлома](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-frame-options)
* [Заголовки ответа HTTP безопасности](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers)
* Дополнительная интеграция [Spring MVC](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#mvc)
  + Автоматическое разрешение Authentication.getPrincipal()с помощью [@AuthenticationPrincipal](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#mvc-authentication-principal)
  + Автоматическая [интеграция Spring MVC Async](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#mvc-async)
  + [Интеграция Spring MVC и CSRF](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#mvc-csrf)
* [Поддержка параллелизма](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#concurrency)
* [Servlet 3+ Интеграция](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#servletapi-3) и [Servlet 3.1+ интеграции](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#servletapi-31)
* Расширенная возможность [разрешать имена параметров](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#el-pre-post-annotations-arguments) методов для обеспечения безопасности на основе методов
  + Поддержка стандартного отражения JDK 8
  + Поддержка разрешения на основе аннотаций
  + Включает разрешение имен параметров на интерфейсах
  + Автоматическая интеграция с @Paramтегом Spring Data
* Дополнительные RequestMatcherреализации
  + [MediaTypeRequestMatcher](https://docs.spring.io/spring-security/site/docs/3.2.x-SNAPSHOT/apidocs/org/springframework/security/web/util/matcher/MediaTypeRequestMatcher.html) - позволяет сопоставлять запросы с помощью согласования содержимого.
  + OrRequestMatcher- позволяет передавать несколько экземпляров RequestMatcher в конструктор. Если один возвращает истину, то результат истинный.
  + AndRequestMatcher- позволяет передавать несколько экземпляров RequestMatcher в конструктор. Если все они возвращают истину, тогда результат будет истинным.
  + NegatedRequestMatcher- разрешает заполнение экземпляра RequestMatcher. Если результат делегата ложный, результат истинный.
* DebugFilter теперь выводит заголовки запросов
* Документация
  + Приступили к созданию ориентированных на задачи [руководств](https://docs.spring.io/spring-security/site/docs/3.2.x-SNAPSHOT/guides/)
  + Добавлено 10+ [образцов безопасности Spring](https://github.com/spring-projects/spring-security/tree/master/samples)
  + Конвертировал всю документацию в [Asciidoctor](http://asciidoctor.org/)
* Интеграция сонара для сборки

**Безопасность Spring Boot 2 против безопасности Spring Boot 1**

По сравнению с Spring Boot 1, **Spring Boot 2 значительно упростил автоконфигурацию**.

* В Spring Boot 2, если нам нужна собственная конфигурация безопасности, мы можем просто добавить настраиваемый [***WebSecurityConfigurerAdapter***](https://www.baeldung.com/java-config-spring-security)*.*Это отключит автоконфигурацию по умолчанию и включит нашу настраиваемую конфигурацию безопасности.
* Spring Boot 2 использует большинство настроек Spring Security по умолчанию. Из-за этого **некоторые конечные точки, которые по умолчанию не были защищены в Spring Boot 1, теперь защищены по умолчанию**. Эти конечные точки включают статические ресурсы, такие как / **css / \*\*, / js / \*\*, / images / \*\*, / webjars / \*\*, /\*\*/favicon.ico**, и конечную точку ошибки. Если нам нужно разрешить неаутентифицированный доступ к этим конечным точкам, мы можем это явно настроить.

3. Конфигурация Java

Общая поддержка [конфигурации Java](https://docs.spring.io/spring/docs/3.1.x/spring-framework-reference/html/beans.html#beans-java) была добавлена ​​в платформу Spring в Spring 3.1. Начиная с Spring Security 3.2 появилась поддержка Spring Security Java Configuration, которая позволяет пользователям легко настраивать Spring Security без использования какого-либо XML. Но если базовая настройка не подходит (нужны доп. “навороты”), то лучше использовать пространство имен с XML. Т.к. там нормально написана полная документация (с объяснениями и примерами), а не просто урезанный JavaDoc или розрозненные статьи с разными кусками.

Если вы знакомы с [конфигурацией пространства имен безопасности,](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-config) вы должны найти довольно много общего между ней и поддержкой конфигурации безопасности Java.

|  |  |
| --- | --- |
|  | Spring Security предоставляет [множество примеров приложений,](https://github.com/spring-projects/spring-security/tree/master/samples) которые заканчиваются -jcдемонстрацией использования конфигурации Java Spring Security. |

3.1. Конфигурация Java Hello Web Security

Первым шагом является создание нашей конфигурации Java Spring Security. Конфигурация создает фильтр сервлетов, известный как фильтр springSecurityFilterChain, который отвечает за всю безопасность (защита URL-адресов приложений, проверка отправленных имени пользователя и паролей, перенаправление в форму входа и т.д.) в вашем приложении. Вы можете найти самый простой пример конфигурации Java Spring Security ниже:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.\*;

import org.springframework.security.config.annotation.authentication.builders.\*;

import org.springframework.security.config.annotation.web.configuration.\*;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth

.inMemoryAuthentication() // задает что пользователь хранится в памяти, а не в БД и не в LDAP

.withUser("user").password(passwordEncoder().encode("password")).roles("USER");

}

}

|  |  |
| --- | --- |
|  | Имя метода configureGlobal не имеет значения.  Однако важно настраивать AuthenticationManagerBuilder только в классе, помеченном аннотацией @EnableWebSecurity, @EnableWebMvcSecurity(который в отличаи от @EnableWebSecurity, при использовании Thymeleaf 2.1+ должен начать автоматически вводить токен CSRF с помощью Spring Security 3.2+, без необходимости дописывания в формы отправки на сервер блока кода со скрытым полем в котором передается токен для включенной по умолчанию защиты CSRF), @EnableGlobalMethodSecurity или @EnableGlobalAuthentication. Иначе результат будет непредсказуем. Если ее нет, приложение не запустится. Аннотация является необязательной, только если мы просто переопределяем поведение по умолчанию с помощью WebSecurityConfigurerAdapter. |

В этой конфигурации действительно немного написано, но она многое делает. Вы можете найти краткое изложение функций ниже:

* Требует аутентификации для каждого URL-адреса в вашем приложении
* Создает для вас форму входа
* Разрешает пользователю с **именем***пользователя* и **паролем-***паролем* аутентифицироваться с аутентификацией на основе формы
* Разрешает пользователю выйти из системы
* Добавляет кодировщик паролей
* Предотвращает [CSRF-атак](https://en.wikipedia.org/wiki/Cross-site_request_forgery) (механизм, при котором с каждой сессией пользователя ассоциируется дополнительный секретный уникальный ключ, предназначенный для выполнения запросов. Секретный ключ не должен передаваться в открытом виде, например, для [POST](https://ru.wikipedia.org/wiki/POST_(HTTP))-запросов ключ следует передавать в теле запроса, а не в адресе страницы. Браузер пользователя посылает этот ключ в числе параметров каждого запроса, и перед выполнением каких-либо действий сервер проверяет этот ключ.)
* Защищает от [фиксации сеанса](https://en.wikipedia.org/wiki/Session_fixation) ( пытаются [использовать](https://en.wikipedia.org/wiki/Exploit_(computer_security)) уязвимость системы, которая позволяет одному человеку зафиксировать (найти или установить) [идентификатор сеанса](https://en.wikipedia.org/wiki/Session_ID) другого человека , принимаемые из [URL-адресов](https://en.wikipedia.org/wiki/Uniform_Resource_Locator) ( [строки запроса](https://en.wikipedia.org/wiki/Query_string) ) или данных POST.)
* Добавляет в ответ заголовки безопасности:
  + [HTTP Strict Transport Security](https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security) для безопасных запросов
  + Интеграция [X-Content-Type-Options](https://msdn.microsoft.com/en-us/library/ie/gg622941(v=vs.85).aspx)
  + Управление кешем (позже может быть переопределено вашим приложением, чтобы разрешить кеширование ваших статических ресурсов)
  + Интеграция [X-XSS-Protection](https://msdn.microsoft.com/en-us/library/dd565647(v=vs.85).aspx)
  + Интеграция X-Frame-Options для предотвращения [кликджекинга](https://en.wikipedia.org/wiki/Clickjacking) (Принцип основан на том, что поверх видимой страницы располагается невидимый слой, в который и загружается нужная злоумышленнику страница, при этом элемент управления (кнопка, ссылка), необходимый для осуществления требуемого действия, совмещается с видимой ссылкой или кнопкой, нажатие на которую ожидается от пользователя)
* Интегрируйте со следующими методами API сервлетов
  + [HttpServletRequest # getRemoteUser ()](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#getRemoteUser())
  + [HttpServletRequest.html # getUserPrincipal ()](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#getUserPrincipal())
  + [HttpServletRequest.html # isUserInRole (java.lang.String)](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#isUserInRole(java.lang.String))
  + [HttpServletRequest.html # login (java.lang.String, java.lang.String)](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#login(java.lang.String,%20java.lang.String))
  + [HttpServletRequest.html # logout ()](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#logout())

**Начиная с Spring 5, для добавления кодировщика паролей с использованием кодирования ключа медленным хэш алгоритмом** bcrypt **нам в файле настройки безопасности понадобится добавить bean-компонент**[***PasswordEncoder***](https://www.baeldung.com/spring-security-5-default-password-encoder)**:**

**@Bean**

**public PasswordEncoder passwordEncoder() {**

**return new BCryptPasswordEncoder();**

**}**

**@Autowired**

**private PasswordEncoder passwordEncoder;**

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

**auth.jdbcAuthentication()**

**.dataSource(dataSource)**

**.passwordEncoder(passwordEncoder)**

----------------------------------------------------

**И в сервисном файле или файле контроллера добавить:**

**в классе**

@Autowired  
private PasswordEncoder passwordEncoder;

**в самом методе**

user.setActive(true);  
user.setRoles(Collections.*singleton*(Role.*USER*));  
user.setPassword(passwordEncoder.encode(user.getPassword()));

registrationRepository.save(user);

**Если по какой-либо причине мы не хотим кодировать настроенный пароль**, мы можем использовать [NoOpPasswordEncoder](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/password/NoOpPasswordEncoder.html):

auth

.inMemoryAuthentication()

.withUser(**"spring"**).password(**"{noop}secret"**).roles(**"USER"**);

**Обновление существующих паролей до рекомендованных стандартов Spring Security 5:**

* Обновление паролей, сохраненных в виде обычного текста, с их закодированными значениями:

String encoded = **new** BCryptPasswordEncoder().encode(plainTextPassword);

* Приставка к хешированным сохраненным паролям их известным идентификатором кодировщика:

{bcrypt}$2a$10$MF7hYnWLeLT66gNccBgxaONZHbrSMjlUofkp50sSpBw2PJjUqU.zS

{sha256}97cde38028ad898ebc02e690819fa220e88c62e0699403e94fff291cfffaf8410849f27605abcbc0

**Шифрование паролей должно быть только с использованием односторонних методов(хеш-функций).**

Это связано с тем, что, за исключением некоторых конкретных сценариев (в основном, связанных с устаревшей интеграцией), *нет абсолютно никаких причин для расшифровки пароля* . Если вы зашифруете свои пароли с помощью шифрования на основе паролей (двусторонний метод) и злоумышленник узнает ваш пароль шифрования, все ваши пароли пользователей будут раскрыты (и, возможно, все одновременно). Если у вас нет такого пароля (или ключа) шифрования, который можно было бы расшифровать, этот риск исчезнет, ​​и злоумышленнику придется полагаться на грубую силу или аналогичные стратегии.

**«Но что, если один из моих пользователей потеряет свой пароль? Разве я не могу напомнить ему / ей?**

**Ответ - громкое и ясное НЕТ** . Вы не только не можете напоминать им их пароли, но на самом деле у вас даже не должно быть возможности читать / узнавать / видеть пароли ваших пользователей, независимо от того, являетесь ли вы системным администратором! *Если один из ваших пользователей потеряет свой пароль, просто установите новое значение и отправьте ему сообщение на подтвержденный адрес электронной почты с новым паролем с просьбой изменить его как можно скорее* .

3.1.1. АннотацияБезопасностьWebApplicationInitializer

Следующим шагом будет регистрация springSecurityFilterChain в war-архиве.

Это можно сделать в конфигурации Java с поддержкой [Spring WebApplicationInitializer](https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-container-config)  в среде Servlet 3.0+.  Spring Security предоставляет базовый класс AbstractSecurityWebApplicationInitializer, который гарантирует, что springSecurityFilterChain будет зарегистрирован за вас.

Использование AbstractSecurityWebApplicationInitializer различается в зависимости от того, используем ли мы уже Spring или Spring Security является единственным компонентом Spring в нашем приложении.

3.1.1.1 AbstractSecurityWebApplicationInitializer без использования Spring

Вам нужно будет передать SecurityConfig объект в суперкласс, чтобы убедиться, что конфигурация подобрана. Вы можете найти пример ниже:

import org.springframework.security.web.context.\*;

public class SecurityWebApplicationInitializer

extends AbstractSecurityWebApplicationInitializer {

public SecurityWebApplicationInitializer() {

super(SecurityConfig.class);

}

}

SecurityWebApplicationInitializer будет делать следующее:

* Автоматически регистрировать фильтр springSecurityFilterChain для каждого URL-адреса в вашем приложении,
* Добавляет ContextLoaderListener, который загружает [SecurityConfig](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc-hello-wsca).

3.1.1.2 AbstractSecurityWebApplicationInitializer с Spring MVC

Если бы мы использовали Spring где-то еще в нашем приложении, у нас, вероятно, уже был бы WebApplicationInitializer загружающий нашу конфигурацию Spring (*раньше использовался* DispatcherServlet *с* xml-*файлом настройки*).  Если мы воспользуемся предыдущей конфигурацией, мы получим ошибку. Вместо этого мы должны зарегистрировать Spring Security с существующим ApplicationContext.

Например, если бы мы использовали Spring MVC, наш SecurityWebApplicationInitializer выглядел бы примерно так:

import org.springframework.security.web.context.\*;

public class SecurityWebApplicationInitializer

extends AbstractSecurityWebApplicationInitializer {}

Это просто зарегистрирует фильтр springSecurityFilterChain для каждого URL-адреса в вашем приложении.

После этого мы должны убедиться, что SecurityConfig загружен в наш существующий ApplicationInitializer.

Например, если бы мы использовали Spring MVC, он был бы добавлен в getRootConfigClasses()

public class MvcWebApplicationInitializer extends

AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class<?>[] getRootConfigClasses() {

return new Class[] { SecurityConfig.class };

}

// ... other overrides ...

}

3.2. HttpSecurity

Пока что наш [SecurityConfig](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#jc-hello-wsca) содержит только информацию о том, как аутентифицировать наших пользователей.  Как Spring Security узнает, что мы хотим требовать аутентификации всех пользователей?  Как Spring Security узнает, что мы хотим поддерживать аутентификацию на основе форм?  Причина этого в том, что  WebSecurityConfigurerAdapter (*который мы, как показано выше, “’экстендим” в класс SecurityConfig*) предоставляет конфигурацию по умолчанию в методе configure(HttpSecurity http), который выглядит так:

protected void **configure**(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.formLogin()

.and()

.httpBasic();

}

Конфигурация по умолчанию выше:

* Гарантирует, что любой запрос к нашему приложению требует аутентификации пользователя
* Позволяет пользователям аутентифицироваться с помощью входа в систему на основе формы
* Позволяет пользователям проходить аутентификацию с помощью Базовой аутентификации HTTP (см. на сайте глава “Безопасность веб приложений” п.4.1), которая передает пароль в виде обычного текста и поэтому должна использоваться только в HTTPS. При использовании HTTP протокола должна использоваться, более “геморойная” в настройке, дайджест-аутентификация (см. на сайте глава “Безопасность веб приложений” п.4.2).

**Все доступные методы, аналогичные authorizeRequests(),formLogin() и httpBasic(), смотри в методах для класса HttpSecurity. А подметоды для этих методов искать нажимая на классы в первом столбике таблицы.**

Вы заметите, что эта конфигурация очень похожа на конфигурацию пространства имен XML:

<http use-expressions="true">

<intercept-url pattern="/\*\*" access="authenticated" />

<form-login />

<http-basic />

</http>

Конфигурация Java, эквивалентная закрытию тега XML, выражается в использовании метода and(), который позволяет нам продолжить настройку родительского элемента. Я хочу настроить авторизованные запросы  *и*  настроить форму входа в систему  *и*  настроить базовую аутентификацию HTTP.

Однако конфигурация Java имеет другие URL-адреса и параметры по умолчанию. Помните об этом при создании пользовательских страниц входа. В результате наши URL-адреса более RESTful.

Кроме того, не так очевидно, что мы используем Spring Security, который помогает предотвратить [утечку информации](https://www.owasp.org/index.php/Information_Leak_(information_disclosure)) . Например:

* GET / login отображает страницу входа вместо / spring\_security\_login
* POST / login аутентифицирует пользователя вместо / j\_spring\_security\_check
* Параметр username по умолчанию - username вместо j\_username
* Параметр пароля по умолчанию - пароль вместо j\_password.

Если бы мы использовали **не Spring Boot, а просто Spring**, то чтобы использовать определенную выше конфигурацию Spring Security, нам нужно прикрепить ее к веб-приложению. Мы будем использовать *WebApplicationInitializer*, поэтому нам не нужно предоставлять какой-либо *web.xml****:***

**public** **class** AppInitializer **implements** WebApplicationInitializer {

@Override

**public** **void** onStartup(ServletContext sc) {

AnnotationConfigWebApplicationContext root = **new** AnnotationConfigWebApplicationContext();

root.register(SecSecurityConfig.class);

sc.addListener(**new** ContextLoaderListener(root));

sc.addFilter(**"securityFilter"**, **new** DelegatingFilterProxy(**"springSecurityFilterChain"**))

.addMappingForUrlPatterns(**null**, **false**, **"/\*"**);

}

}

3.3. Конфигурация Java и вход в форму

Вам может быть интересно, откуда взялась форма входа, когда вам было предложено войти в систему, поскольку мы не упоминали какие-либо файлы HTML или JSP.

Поскольку конфигурация Spring Security по умолчанию явно не устанавливает URL-адрес для страницы входа в систему, Spring Security генерирует его автоматически на основе включенных функций и с использованием стандартных значений для URL-адреса, который обрабатывает отправленный логин, целевой URL-адрес по умолчанию, который пользователь будет отправлено после входа в систему и т. д.

Хотя автоматически сгенерированную страницу входа в систему удобно начать и быстро запустить, большинство приложений захотят предоставить свою собственную страницу входа. Для этого мы можем обновить нашу конфигурацию, как показано ниже:

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll();

.logout()

.permitAll();

}

|  |  |
| --- | --- |
|  | .loginPage("/login") означает, что при попытке зайти на закрытую паролем страницу (*не авторизовавшись до этого*) нас автоматически, внутри сервера, перебросит с метода контроллера, отвечающего за обработку этой страницы, в метод “/login” контроллера, который выдает на выходе страницу “login.html” и клиенту открывается страница с требованием ввести логин и пароль.  В обновленной конфигурации указано расположение страницы входа в систему. |
|  | Мы должны предоставить всем пользователям (т.е. пользователям, не прошедшим проверку подлинности) доступ к нашей странице входа в систему.  Метод .permitAll()позволяет предоставить доступ всем пользователям вне зависимости от роли. |

Метод .logout().permitAll() определяет что “разлогиниться” могут все.

Также для .formLogin() существуют параметры:

расширим указанную выше конфигурацию для входа в форму и выхода из нее:

@Override

**protected** **void** **configure**(**final** HttpSecurity http) **throws** Exception {

http

// ...

.and()

.formLogin()

.loginPage("/login.html")

.loginProcessingUrl("/perform\_login")

.defaultSuccessUrl("/homepage.html", **true**)

.failureUrl("/login.html?error=true")

.failureHandler(authenticationFailureHandler())

.and()

.logout()

.logoutUrl("/perform\_logout")

.deleteCookies("JSESSIONID")

.logoutSuccessHandler(logoutSuccessHandler());

}

* **always-use-default-target** - если установлено значение true, пользователь всегда будет начинать со значения, заданного default-target-url, независимо от того, как они попали на страницу входа. Сопоставляется со свойством alwaysUseDefaultTargetUrl UsernamePasswordAuthenticationFilter. Значение по умолчанию - false.
* **authentication-details-source-ref** - ссылка на AuthenticationDetailsSource, который будет использоваться фильтром аутентификации.
* **authentication-failure-handler-ref** - может использоваться в качестве альтернативы URL-адресу ошибки-аутентификации, давая вам полный контроль над потоком навигации после сбоя аутентификации. Значением должно быть имя bean-объекта AuthenticationFailureHandler в контексте приложения.
* **authentication-failure-url** - сопоставляется со свойством authenticationFailureUrl объекта UsernamePasswordAuthenticationFilter. Определяет URL-адрес, на который браузер будет перенаправлен при ошибке входа в систему. По умолчанию это ***/ spring\_security\_login? Login\_error***, который будет автоматически обрабатываться автоматическим генератором страницы входа, повторно отображая страницу входа с сообщением об ошибке.
* **authentication-success-handler-ref** - это можно использовать как альтернативу default-target-url и always-use-default-target, давая вам полный контроль над навигационным потоком после успешной аутентификации. Значение должно быть именем bean-объекта AuthenticationSuccessHandler в контексте приложения. По умолчанию используется реализация SavedRequestAwareAuthenticationSuccessHandler, которая вводится с целевым URL-адресом по умолчанию.
* **default-target-url -** сопоставляется со свойством defaultTargetUrl объекта UsernamePasswordAuthenticationFilter. Если не установлен, значение по умолчанию - ***«/»*** (корень приложения). Пользователь будет перенаправлен на этот URL-адрес после входа в систему, при условии, что его не попросили войти в систему при попытке доступа к защищенному ресурсу, когда они будут перенаправлены на первоначально запрошенный URL-адрес.
* **login-page** URL-адрес, который следует использовать для отображения страницы входа. Сопоставляется со свойством loginFormUrl объекта LoginUrlAuthenticationEntryPoint. По умолчанию "/ spring\_security\_login".
* **login-processing-url** Сопоставляется со свойством filterProcessesUrl объекта UsernamePasswordAuthenticationFilter. Значение по умолчанию - «***/ j\_spring\_security\_check***».
* **password-parameter** Имя параметра запроса, содержащего пароль. По умолчанию "***j\_password***".
* **username-parameter** Имя параметра запроса, который содержит имя пользователя. По умолчанию "***j\_username***".

Пример страницы входа "/login", реализованной с помощью JSP для нашей текущей конфигурации, можно увидеть ниже:

|  |  |
| --- | --- |
|  | На странице входа ниже представлена ​​наша текущая конфигурация. Мы могли бы легко обновить нашу конфигурацию, если некоторые из значений по умолчанию не соответствуют нашим требованиям. |

<c:url value="/login" var="loginUrl"/> //устанавливаем значение переменной

<form action="${loginUrl}" method="post"> //указываем адрес программы или документа, который будет обрабатывает данные отправленные из этой формы

<c:if test="${param.error != null}"> //где “test” это служебное слово после которого должно идти выражение с какой-то проверкой. А через служебный объект “param” *(который хранит все передаваемые странице параметры)* получаем из запроса значения параметра “error”

<p>

Invalid username and password.

</p>

</c:if>

<c:if test="${param.logout != null}">

<p>

You have been logged out.

</p>

</c:if>

<p>

<label for="username">Username</label>

<input type="text" id="username" name="username"/>

</p>

<p>

<label for="password">Password</label>

<input type="password" id="password" name="password"/>

</p>

<input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}"/> // включаем токен CSRF в форме входа в систему как скрытый параметр, для CSRF-защиты включенной по умолчанию

<button type="submit" class="btn">Log in</button>

</form>

Тут после нажатия на кнопку "submit" на сервер методом "post" будет отправлено:

- текст или “Invalid username and password.” и/или “You have been logged out.”,

- переменная "username" = “введенное значение”,

- переменная "password" = “введенное значение”,

- отправляем токен для включенной по умолчанию защиты CSRF.

|  |  |
| --- | --- |
|  | POST на URL-адрес “/ login” будет пытаться аутентифицировать пользователя. |
|  | Если параметр запроса errorсуществует, попытка аутентификации не удалась. |
|  | Если параметр запроса logoutсуществует, пользователь успешно вышел из системы. |
|  | Имя пользователя должно присутствовать как параметр HTTP с именем *username.* |
|  | Пароль должен присутствовать как параметр HTTP с именем *password.* |
|  | Мы [включаем токен CSRF.](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#csrf-include-csrf-token) *Чтобы узнать больше, прочтите раздел справки о*[*подделке межсайтовых запросов (CSRF).*](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#csrf) |

Также для .formLogin() существуют параметры:

**.loginProcessingUrl**(**"/perform\_login"**)- URL для отправки имени пользователя и пароля

**.defaultSuccessUrl**(**"/page.html"**, **true**) - целевая страница после успешного входа в систему

**.failureUrl**(**"/login.html?error=true"**)- целевая страница после неудачного входа в систему

**.failureHandler**(authenticationFailureHandler())- пользовательский выход

3.4. Запросы на авторизацию

В наших примерах требуется только аутентификация пользователей, и это сделано для каждого URL-адреса в нашем приложении. Мы можем указать индивидуальные требования для наших URL-адресов, добавив в метод http.authorizeRequests() несколько дочерних элементов. Например:

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/resources/\*\*", "/signup", "/about").permitAll()

.antMatchers("/admin/\*\*").hasRole("ADMIN")

.antMatchers("/db/\*\*").access("hasRole('ROLE\_ADMIN') and hasRole('ROLE\_DBA')")

.anyRequest().authenticated()

.and()

// ...

.formLogin();

}

Обратите внимание, что порядок элементов antMatchers ()  имеет большое значение - **сначала должны быть более конкретные правила, а за ними - более общие**.

У метода http.authorizeRequests() есть несколько дочерних элементов, каждое сопоставление рассматривается в том порядке, в котором они были объявлены:

- Мы указали несколько шаблонов URL, к которым может получить доступ любой пользователь. В частности, любой пользователь может получить доступ к запросу, если URL-адрес **начинается** с «/resources /», **равен** «/ signup» или равен «/ about».

- Поскольку мы вызываем метод hasRole, нам не нужно указывать префикс «ROLE» и любой URL, начинающийся с «/admin/», будет доступен только пользователям с ролью «ROLE\_ADMIN».

- Любой URL, который начинается с «/db/», требует, чтобы у пользователя были **и** «ROLE\_ADMIN» **и** «ROLE\_DBA».

- Любой URL, который еще не был сопоставлен *(так же как и в первом пункте)* требует только аутентификации пользователя.

3.5 Обработка выхода из системы

При использовании [WebSecurityConfigurerAdapter](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/config/annotation/web/configuration/WebSecurityConfigurerAdapter.html)автоматически применяются возможности выхода из системы. По умолчанию доступ к URL-адресу /logoutприведет к выходу пользователя из системы:

* Аннулирование HTTP-сеанса
* Очистка любой настроенной аутентификации RememberMe
* Очистка SecurityContextHolder
* Перенаправить на /login?logout

Однако, как и при настройке возможностей входа в систему, у вас также есть различные возможности для дальнейшей настройки требований выхода из системы:

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http

.logout()

.logoutUrl("/my/logout")

.logoutSuccessUrl("/my/index")

.logoutSuccessHandler(logoutSuccessHandler)

.invalidateHttpSession(true)

.addLogoutHandler(logoutHandler)

.deleteCookies(cookieNamesToClear)

.and()

...

}

, где:

***logout()*** - обеспечивает выход из системы. Применяется автоматически при использовании WebSecurityConfigurerAdapter.

***logoutUrl()*** - URL-адрес, запускающий выход из системы (по умолчанию /logout). Если защита CSRF включена (по умолчанию), то запрос также должен быть POST.

***logoutSuccessUrl()*** - URL-адрес для перенаправления после выхода из системы. По умолчанию это /login?logout.

***logoutSuccessHandler()*** - Давайте укажем custom LogoutSuccessHandler. Если это указано, logoutSuccessUrl() игнорируется.

***invalidateHttpSession()*** - Укажите, следует ли аннулировать HttpSession при выходе из системы. Это верно по умолчанию..

***addLogoutHandler()*** - Добавляет LogoutHandler. по умолчанию SecurityContextLogoutHandler добавляется последним LogoutHandler.

***deleteCookies()*** - Позволяет указать имена файлов cookie, которые будут удалены при успешном выходе из системы.

Как правило, для настройки функции выхода из системы вы можете добавить реализации LogoutHandler и / или LogoutSuccessHandler. Для многих распространенных сценариев эти обработчики применяются незаметно при использовании Fluent API.

**3.5.1 LogoutHandler**

Как правило, [LogoutHandler](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/authentication/logout/LogoutHandler.html) реализации указывают классы, которые могут участвовать в обработке выхода из системы. Ожидается, что они будут вызваны для выполнения необходимой очистки. Таким образом, они не должны создавать исключения. Предусмотрены различные реализации:

* [PersistentTokenBasedRememberMeServices](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/authentication/rememberme/PersistentTokenBasedRememberMeServices.html)
* [TokenBasedRememberMeServices](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/authentication/rememberme/TokenBasedRememberMeServices.html)
* [CookieClearingLogoutHandler](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/authentication/logout/CookieClearingLogoutHandler.html)
* [CsrfLogoutHandler](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/csrf/CsrfLogoutHandler.html)
* [SecurityContextLogoutHandler](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/authentication/logout/SecurityContextLogoutHandler.html)

За подробностями обратитесь к [Раздел 17.4, «Интерфейсы и реализации Remember-Me»](https://docs.spring.io/spring-security/site/docs/4.2.x/reference/html/remember-me.html#remember-me-impls) .

Вместо того, чтобы предоставлять LogoutHandlerреализации напрямую, свободный API также предоставляет ярлыки, которые предоставляют соответствующие LogoutHandlerреализации под обложками. Например, deleteCookies()позволяет указать имена одного или нескольких файлов cookie, которые будут удалены при успешном выходе из системы. Это ярлык по сравнению с добавлением файла CookieClearingLogoutHandler.

**3.5.2 LogoutSuccessHandler**

LogoutSuccessHandlerВызывается после успешного завершения сеанса по LogoutFilter, для обработки , например , переадресацию или перенаправление на соответствующий пункт назначения. Обратите внимание, что интерфейс почти такой же, как у, LogoutHandlerно может вызывать исключение.

Предоставляются следующие реализации:

* [SimpleUrlLogoutSuccessHandler](https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/authentication/logout/SimpleUrlLogoutSuccessHandler.html)
* HttpStatusReturningLogoutSuccessHandler

Как упоминалось выше, вам не нужно указывать SimpleUrlLogoutSuccessHandler напрямую. Вместо этого свободный API предоставляет ярлык, задав для logoutSuccessUrl(). Это настроит SimpleUrlLogoutSuccessHandler. Указанный URL-адрес будет перенаправлен после выхода из системы. По умолчанию это /login?logout.

HttpStatusReturningLogoutSuccessHandlerможет быть интересен в сценариях типа REST API. Вместо перенаправления на URL-адрес при успешном выходе из системы это LogoutSuccessHandler позволяет вам предоставить простой код состояния HTTP для возврата. Если не настроен, по умолчанию будет возвращен код состояния 200.

3.6. Аутентификация

До сих пор мы рассмотрели только самую простую конфигурацию аутентификации. Давайте взглянем на несколько более продвинутых параметров настройки аутентификации.

3.6.1. Аутентификации в памяти

Мы уже видели пример настройки аутентификации в памяти для одного пользователя. Ниже приведен пример настройки нескольких пользователей:

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth

.inMemoryAuthentication()

.withUser("user").password(passwordEncoder().encode("password")).roles("USER")

.**and()**

.withUser("admin").password(passwordEncoder().encode("password")).roles("USER", "ADMIN");

}

3.6.2. JDBC аутентификация

Вы можете найти обновления для поддержки аутентификации на основе JDBC.

1. В приведенном ниже примере предполагается, что вы уже определили DataSourceв своем приложении. [JDBC-х образец](https://github.com/spring-projects/spring-security/tree/master/samples/jdbc-jc) обеспечивает полный пример использования проверки подлинности на основе JDBC.

@Autowired

private DataSource dataSource;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth

.jdbcAuthentication()

.dataSource(dataSource)

.withDefaultSchema()

.withUser("user"). password(passwordEncoder().encode("password")).roles("USER").and()

.withUser("admin"). password(passwordEncoder().encode("password")).roles("USER", "ADMIN");

}

Конечно, в обоих приведенных выше примерах нам также необходимо определить bean-компонент PasswordEncoder, как описано в разделе 3.1.

2. Пример для случая, когда логин/пароль самостоятельно создаются новыми клиентами и хряться в БД, где создаются две таблицы *(одна для ролей(подчиненная), а вторая для всего остального(главная) - в этом примере это соответственно “AUTHORITIES” и “* *USER ”)*:

  @Override

    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

      auth

.jdbcAuthentication()

.dataSource(dataSource)

 .usersByUsernameQuery("SELECT USERNAME, PASSWORD, ENABLED FROM USER WHERE

USERNAME=?")

          .authoritiesByUsernameQuery("SELECT U.USERNAME, A.AUTHORITY FROM AUTHORITIES A, USER U WHERE U.USERNAME = A.USERNAME AND U.USERNAME = ?");

    }

Кроме того, чтобы определить доступ определенной роли к конкретному контроллеру, в котором находятся необходимые методы обработки, нужно (например для запуска механизма допуска к методам данного контроллера только с ролью "ADMIN”):

- в файле контроллера над классом (или только над необходимыми методами) дописать аннотацию **@PreAuthorize("hasAuthority('ADMIN')")**

- в файле настройки безопасности над классом дописать аннотацию **@EnableGlobalMethodSecurity(prePostEnabled = true)** .

3.6.3. LDAP аутентификация

Вы можете найти обновления для поддержки аутентификации на основе LDAP. [LDAP-х образец](https://github.com/spring-projects/spring-security/tree/master/samples/ldap-jc) обеспечивает полный пример использования проверки подлинности на основе LDAP.

@Autowired

private DataSource dataSource;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth

.ldapAuthentication()

.userDnPatterns("uid={0},ou=people")

.groupSearchBase("ou=groups");

}

В приведенном выше примере используется следующий LDIF и встроенный экземпляр LDAP Apache DS.

**users.ldif**

dn: ou = groups, dc = springframework, dc = org

объектный класс: верх

объектный класс: организационная единица

ou: группы

dn: ou = люди, dc = springframework, dc = org

объектный класс: верх

объектный класс: организационная единица

ou: люди

dn: uid = admin, ou = people, dc = springframework, dc = org

объектный класс: верх

объектный класс: человек

объектный класс: organizationPerson

объектный класс: inetOrgPerson

CN: Род Джонсон

сн: Джонсон

uid: admin

userPassword: пароль

dn: uid = user, ou = people, dc = springframework, dc = org

объектный класс: верх

объектный класс: человек

объектный класс: organizationPerson

объектный класс: inetOrgPerson

CN: Дайан Эму

sn: Эму

uid: пользователь

userPassword: пароль

dn: cn = пользователь, ou = groups, dc = springframework, dc = org

объектный класс: верх

объектный класс: groupOfNames

cn: пользователь

uniqueMember: uid = admin, ou = people, dc = springframework, dc = org

uniqueMember: uid = user, ou = people, dc = springframework, dc = org

dn: cn = admin, ou = groups, dc = springframework, dc = org

объектный класс: верх

объектный класс: groupOfNames

cn: admin

uniqueMember: uid = admin, ou = people, dc = springframework, dc = org

3.7. Несколько HttpSecurity

Мы можем настроить несколько экземпляров HttpSecurity так же, как мы можем иметь несколько <http>блоков. Ключевым моментом является многократное расширение адаптера WebSecurityConfiguration Adapter. Например, ниже приведен пример другой конфигурации URL-адресов, начинающихся с /api/.

@Configuration

@EnableWebSecurity

public class MultiHttpSecurityConfig {

//Стандартный блок Аутентификации

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) {

auth

.inMemoryAuthentication()

.withUser("user").password("password").roles("USER").and()

.withUser("admin").password("password").roles("USER", "ADMIN");

}

//Блок Авторизации

@Configuration

@Order(1)

public static class ApiWebSecurityConfigurationAdapter extends

WebSecurityConfigurerAdapter {

protected void configure(HttpSecurity http) throws Exception {

http

.antMatcher("/api/\*\*")

.authorizeRequests()

.anyRequest().hasRole("ADMIN")

.and()

.httpBasic();

}

}

@Configuration

public static class FormLoginWebSecurityConfigurerAdapter extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.formLogin();

}

}

}

|  |  |
| --- | --- |
| 1. | Настройте аутентификацию как обычно |
| 2. | Создайте экземпляр WebSecurityConfigurerAdapter, содержащий @Order, чтобы указать, какой WebSecurityConfigurerAdapter следует рассматривать в первую очередь. |
| 3. | В http.antMatcher указано, что этот HttpSecurity будет применяться только к URL-адресам, начинающимся с “/ api /”. |
| 4. | Создайте еще один экземпляр WebSecurityConfigurerAdapter. Если URL-адрес не начинается с “/api/”, будет использоваться эта конфигурация. Эта конфигурация рассматривается после ApiWebSecurityConfigurationAdapter, поскольку она имеет значение @Order после 1 (по умолчанию @Order не является последним). |

3.8. Переход на HTTPS протокол

3.8.1. Первоначальная настройка без защиты канала

Доступ контролируется следующей конфигурацией:

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/anonymous\*").anonymous();

http

.authorizeRequests()

.antMatchers(**"/login\*"**).permitAll();

http

.authorizeRequests()

.anyRequest().authenticated();

Или через XML:

<**http** use-expressions=**"true"**>

<**intercept-url** pattern=**"/anonymous\*"** access=**"isAnonymous()"**/>

<**intercept-url** pattern=**"/login\*"** access=**"permitAll"**/>

<**intercept-url** pattern=**"/\*\*"** access=**"isAuthenticated()"**/>

</**http**>

На данный момент страница входа доступна по адресу:

<http://localhost:8080/spring-security-login/login.html>

Веб-приложение позволяет пользователям получить доступ:

1. */anonymous.html* без аутентификации,
2. */login.html* и
3. другие страницы ( */homepage.html* ) после успешного входа в систему.

Пользователи могут аутентифицироваться через HTTP, однако это небезопасно, так как пароли будут отправляться в виде обычного текста.

3.8.2. Конфигурация сервера HTTPS

Для того, чтобы только доставить на страницу входа через HTTPS ***-* ваш веб-сервер должен иметь возможность обслуживать HTTPS страницы** . Для этого требуется, чтобы была включена поддержка [SSL / TLS](https://en.wikipedia.org/wiki/Transport_Layer_Security) .

Обратите внимание, что вы можете использовать действующий сертификат или, в целях тестирования, создать свой собственный.

Допустим, мы используем Tomcat и накручиваем собственный сертификат. Сначала нам нужно создать *хранилище ключей* с самозаверяющим сертификатом.

Создать хранилище ключей можно с помощью следующей команды в терминале:

keytool -genkey -alias tomcat -keyalg RSA -storepass changeit -keypass changeit -dname **'CN=tomcat'**

Это создаст закрытый ключ и самозаверяющий сертификат в хранилище ключей по умолчанию для вашего профиля пользователя в вашей домашней папке.

Следующий шаг - отредактировать *файл conf / server.xml,* чтобы он выглядел так (для Spring Boot нужно дописывать в файл **application.properties**(см. мой файл “Spring Boot – руководство.docx” + там немного по другому выглядит команда на создание самозаверяющегося сертификата)):

<**Connector** port=**"8080"** protocol=**"HTTP/1.1"**

connectionTimeout=**"20000"**

redirectPort=**"8443"** />

<**Connector** port=**"8443"** protocol=**"org.apache.coyote.http11.Http11NioProtocol"**

maxThreads=**"150"** SSLEnabled=**"true"** scheme=**"https"** secure=**"true"**

clientAuth=**"false"** sslProtocol=**"TLS"**

keystoreFile=**"${user.home}/.keystore"** keystorePass=**"changeit"** />

Второй тег SSL / TLS *<Connector>* обычно закомментирован в файле конфигурации, поэтому все, что нужно, - это раскомментировать и добавить информацию о хранилище ключей. Дополнительная информация доступна в [соответствующей документации Tomcat](https://tomcat.apache.org/tomcat-8.0-doc/ssl-howto.html) .

После настройки HTTPS страница входа теперь также может быть обслужена по следующему URL-адресу:

https://localhost:8443/spring-security-login/login.html

Веб-серверы, отличные от Tomcat, потребуют другой, но, вероятно, аналогичной конфигурации.

3.8.3. Настройка безопасности канала

**(*будет настроен протокол HTTPS только для страницы входа*)**

На данный момент мы можем обслуживать страницу входа как по HTTP, так и по HTTPS. В этом разделе объясняется, как разрешить использование HTTPS.

**Чтобы требовать HTTPS для страницы входа,** измените конфигурацию безопасности, добавив следующее:

http

.requiresChannel()

.antMatchers(**"/login\*"**).requiresSecure();

Или добавьте атрибут *requires-channel = ”https”* в конфигурацию XML:

<**intercept-url** pattern=**"/login\*"** access=**"permitAll"** requires-channel=**"https"**/>

После этого пользователи смогут входить в систему только через HTTPS. Все относительные ссылки, например, пересылка на */homepage.html* , унаследуют протокол исходного запроса и будут обслуживаться по HTTPS.

При смешивании запросов HTTP и HTTPS внутри одного веб-приложения необходимо учитывать дополнительные аспекты, которые требуют дополнительной настройки.

3.8.4. Смешивание HTTP и HTTPS

С точки зрения безопасности обслуживание всего через HTTPS - хорошая практика и серьезная цель.

Однако, если использование исключительно HTTPS не является вариантом, мы можем настроить Spring для использования HTTP, добавив в конфигурацию следующее:

http.requiresChannel()

.anyRequest().requiresInsecure();

Или добавьте в XML атрибуты *requires-channel = ”http”* :

<**intercept‐url** pattern=**"/\*\*"** access=**"isAuthenticated()"** requires‐channel=**"http"**/>

Это указывает Spring использовать HTTP для всех запросов, которые явно не настроены для использования HTTPS, но в то же время нарушает исходный механизм входа в систему. В следующих разделах объясняется основная причина.

3.8.5. Пользовательский URL-адрес для обработки входа через HTTPS

Конфигурация безопасности в исходном руководстве по безопасности содержит следующее:

<form-login login-processing-url=**"/perform\_login"**/>

**Без принудительного использования */ perform\_login* использования HTTPS для его варианта HTTP произойдет перенаправление с потерей информации для входа,** отправленной с исходным запросом.

Чтобы преодолеть это, нам нужно настроить Spring на использование HTTPS для URL обработки:

http

.requiresChannel()

.antMatchers(**"/login\*"**, **"/perform\_login"**);

Обратите внимание на дополнительный аргумент */ perform\_login,* переданный методу *antMatchers* .

Эквивалент в конфигурации XML требует добавления нового элемента *< intercept-url>* в конфигурацию:

<**intercept-url** pattern=**"/perform\_login"** requires-channel=**"https"**/>

Если ваше собственное приложение использует *URL-адрес обработки входа* по умолчанию (это */ login* ), вам не нужно настраивать это явно, поскольку шаблон */ login \** уже охватывает это.

При наличии конфигурации пользователи могут входить в систему, но не иметь доступа к аутентифицированным страницам, например, */homepage.html* по протоколу HTTP, из-за [функции защиты фиксации сеанса Spring](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#ns-session-fixation) .

3.8.6. Отключение защиты от фиксации сеанса

***(это только для ситуации как тут – входим через HTTPS, а дальнейшая работа через HTTP. При HTTPS повсеместно такая настройка не нужна )***

[Фиксация сеанса](https://en.wikipedia.org/wiki/Session_fixation) - это проблема, которую невозможно избежать при переключении между HTTP и HTTPS.

По умолчанию Spring создает новый *идентификатор сеанса* после успешного входа в систему. Когда пользователь загружает страницу входа HTTPS, файл cookie с *идентификатором сеанса* будет помечен как *безопасный.*После входа в систему контекст переключится на HTTP, и [cookie будет утерян,](https://docs.spring.io/spring-security/site/faq/faq.html#faq-tomcat-https-session) поскольку HTTP небезопасен.

Чтобы избежать этого , **S Эттинг**[***сеанса-фиксации-защиты***](https://docs.spring.io/spring-security/site/docs/current/reference/html/ns-config.html#ns-session-fixation)**в этом *никто* не требуется** .

http.sessionManagement()

.sessionFixation()

.none();

Или через XML:

<**session-management** session-fixation-protection=**"none"**/>

**Отключение защиты от фиксации сеанса может иметь последствия для безопасности** , поэтому вам нужно взвесить все за и против, если вас беспокоят атаки на основе фиксации сеанса.

3.8.7. Тест

После применения всех этих изменений конфигурации доступ к */anonymous.html* без входа в систему (с использованием *http: //* или https: //) направит вас на страницу через HTTP.

Открытие других страниц напрямую, например /homepage.html, должно перенаправить вас на страницу входа через HTTPS, а после входа вы будете перенаправлены обратно на /homepage.html с помощью HTTP.

3.8.8. Заключение

В этом руководстве мы рассмотрели, как настроить веб-приложение Spring, которое обменивается данными через HTTP, за исключением механизма входа в систему которое настроено для работы через HTTPS. Однако **новые современные веб-приложения почти всегда должны использовать HTTPS исключительно в** качестве протокола связи. Понижение уровня безопасности или отключение функций безопасности (например, *защиты от фиксации сеанса* ) - плохая идея.

3.9. Безопасность метода

Начиная с версии 2.0, Spring Security значительно улучшил поддержку для добавления безопасности к методам уровня обслуживания. Он обеспечивает поддержку безопасности аннотаций JSR-250, а также исходную аннотацию @Secured фреймворка. Начиная с версии 3.0 вы также можете использовать новые [аннотации на основе выражений](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#el-access) . Вы можете применить безопасность к одному bean-компоненту, используя элемент intercept-methods для declaration объявления bean-компонента, или вы можете защитить несколько bean-компонентов на всем уровне обслуживания, используя точечные вырезки в стиле AspectJ.

3.9.1. EnableGlobalMethodSecurity

Мы можем включить безопасность на основе аннотаций @EnableGlobalMethodSecurity, используя аннотацию для любого экземпляра @Configuration.  Например, следующий код включит аннотацию @Secured Spring Security:

@Configuration

@EnableGlobalMethodSecurity(securedEnabled = true)

public class MethodSecurityConfig {

// ...

}

Добавление аннотации к методу (в классе или интерфейсе) соответственно ограничит доступ к этому методу. Встроенная поддержка аннотаций Spring Security определяет набор атрибутов для метода. Они будут переданы в AccessDecisionManager, чтобы он принял 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

@Configuration

@EnableGlobalMethodSecurity(jsr250Enabled = true)

public class MethodSecurityConfig {

// ...

}

Они основаны на стандартах и ​​позволяют применять простые ограничения на основе ролей, но не имеют собственных аннотаций Power Spring Security.

Чтобы использовать новый синтаксис на основе выражений, вы должны использовать

@Configuration

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class MethodSecurityConfig {

// ...

}

и эквивалентный код Java будет

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);

}

3.9.2. GlobalMethodSecurityConfiguration

Иногда вам может потребоваться выполнить более сложные операции, чем это возможно с помощью аннотации @EnableGlobalMethodSecurity.  Для этих экземпляров вы можете, расширив GlobalMethodSecurityConfiguration и убедившись, что аннотация @EnableGlobalMethodSecurity присутствует в вашем подклассе, написать собственный метод.  Например, если вы хотите создать собственный метод MethodSecurityExpressionHander, вы можете использовать следующую конфигурацию:

@Configuration

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {

@Override

protected MethodSecurityExpressionHandler createExpressionHandler() {

// ... create and return custom MethodSecurityExpressionHandler ...

return expressionHander;

}

}

Дополнительные сведения о методах, которые можно переопределить, см. В документации GlobalMethodSecurityConfigurationJavadoc.

3.10. Настроенные объекты постобработки

Конфигурация Java Spring Security не раскрывает все свойства каждого настраиваемого объекта. Это упрощает настройку для большинства пользователей. В конце концов, если бы все свойства были раскрыты, пользователи могли бы использовать стандартную конфигурацию bean-компонентов.

Хотя есть веские причины не раскрывать все свойства напрямую, пользователям все же могут потребоваться более сложные параметры конфигурации. Для решения этой проблемы Spring Security вводит концепцию объекта, ObjectPostProcessorкоторый можно использовать для изменения или замены многих экземпляров объектов, созданных конфигурацией Java. Например, если вы хотите настроить filterSecurityPublishAuthorizationSuccessсвойство, FilterSecurityInterceptorвы можете использовать следующее:

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.withObjectPostProcessor(new ObjectPostProcessor<FilterSecurityInterceptor>() {

public <O extends FilterSecurityInterceptor> O postProcess(

O fsi) {

fsi.setPublishAuthorizationSuccess(true);

return fsi;

}

});

}

3.11. Загрузка файлов (css, jpg и т.д.) из папок на титульную и др. страницы до авторизации

@Override

public void configure(WebSecurity web) {

web.ignoring().antMatchers(

"/css/\*\*",

"/js/\*\*",

"/fonts/\*\*",

"/images/\*\*"

);

}

4. Конфигурация пространства имен безопасности

4.1. Введение

Конфигурация пространства имен доступна начиная с версии 2.0 инфраструктуры Spring. Он позволяет дополнить традиционный синтаксис контекста приложения bean-компонентов Spring элементами из дополнительной XML-схемы.

Дополнительную информацию можно найти в [справочной документации](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/htmlsingle/spring-framework-reference.html) Spring.

Элемент пространства имен может использоваться просто для обеспечения более краткого способа настройки отдельного компонента или, что более эффективно, для определения альтернативного синтаксиса конфигурации, который более точно соответствует проблемной области и скрывает основную сложность от пользователя. Простой элемент может скрывать тот факт, что в контекст приложения добавляются несколько компонентов и этапов обработки.

Например, добавление следующего элемента из пространства имен безопасности в контекст приложения запустит встроенный сервер LDAP для тестирования использования в приложении:

<security:ldap-server />

Это намного проще, чем подключение эквивалентных компонентов Apache Directory Server.

Наиболее распространенные требования к альтернативной конфигурации поддерживаются атрибутами ldap-server элемента, и пользователю не нужно беспокоиться о том, какие bean-компоненты им нужно создать и каковы имена свойств bean-компонентов. [ [1](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/" \l "_footnote_1" \o "См. Сноску.) ] .

Использование хорошего XML-редактора при редактировании файла контекста приложения должно предоставить информацию о доступных атрибутах и ​​элементах. Мы рекомендуем вам попробовать [SpringSource Tool Suite,](http://www.springsource.com/products/sts) поскольку он имеет специальные функции для работы со стандартными пространствами имен Spring.

Чтобы начать использовать пространство имен безопасности в контексте вашего приложения, вам необходимо:

* 1. добавить в pom.xml зависимость **spring-security-config.jar**.
  2. какой-то **файл контекста**
  3. файл **web.xml**
  4. файл **security.xml** – в котором и будет происходить настройка

Затем все, что вам нужно сделать, это добавить объявление схемы в файл контекста (2.) вашего 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-3.0.xsd

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

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

...

</beans>

Во многих примерах, которые вы увидите (и в образце) приложений, мы часто будем использовать «безопасность» в качестве пространства имен по умолчанию, а не «beans», что означает, что мы можем опустить префикс во всех элементах пространства имен безопасности, сделав содержимое более читаемым. Вы также можете сделать это, если у вас есть контекст вашего приложения, разделенный на отдельные файлы, и большая часть вашей конфигурации безопасности находится в одном из них. (*Короче первые две строчки меняются местами и их начало немного меняются*).

 Файл контекста вашего приложения безопасности будет тогда 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-3.0.xsd

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

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

...

</beans:beans>

Мы предполагаем, что этот синтаксис используется в этой главе с этого момента.

4.1.1. Дизайн пространства имен

Пространство имен предназначено для сбора наиболее распространенных применений платформы и обеспечения упрощенного и краткого синтаксиса для включения их в приложение. Дизайн основан на крупномасштабных зависимостях внутри фреймворка и может быть разделен на следующие области:

1. *AuthenticationProvider* s - механизмы, с помощью которых диспетчер аутентификации аутентифицирует пользователей. Пространство имен обеспечивает поддержку нескольких стандартных параметров, а также средства добавления пользовательских компонентов, объявленных с использованием традиционного синтаксиса.
2. *UserDetailsService* - тесно связан с поставщиками аутентификации, но часто также требуется для других bean-компонентов и предоставляет геттеры, которые гарантируют ненулевое предоставление аутентификационной информации, такой как имя пользователя, пароль, предоставленные полномочия и включение или отключение учетной записи пользователя.
3. *Web / HTTP Security* - самая сложная часть. Устанавливает фильтры и связанные служебные компоненты, используемые для применения механизмов аутентификации, защиты URL-адресов, отображения страниц входа и ошибок и многого другого.
4. *Безопасность бизнес-объекта (метода)* - варианты защиты уровня сервиса.
5. *AccessDecisionManager* - предоставляет решения о доступе для веб-безопасности и безопасности методов. Будет зарегистрирован вариант по умолчанию, но вы также можете выбрать использование настраиваемого, объявленного с использованием обычного синтаксиса компонента Spring.
6. *AuthenticationManager* - обрабатывает запросы аутентификации от других частей фреймворка.

Т.е. скорее всего:

1-2 - блок аутентификации

3 - блок авторизации

4-5 - п.3.7 (безопасность методов)

6 – вмешательство во внутреннюю часть работы самого Spring Security при обращеннии его блоков друг к другу.

Мы увидим, как их настроить в следующих разделах.

4.2. Начало работы с конфигурацией пространства имен безопасности

В этом разделе мы рассмотрим, как вы можете создать конфигурацию пространства имен для использования некоторых основных функций фреймворка. Предположим, вы изначально хотите как можно быстрее приступить к работе и добавить поддержку аутентификации и контроль доступа к существующему веб-приложению с помощью нескольких тестовых входов. Затем мы рассмотрим, как перейти на аутентификацию по базе данных или другому репозиторию безопасности. В следующих разделах мы представим более сложные параметры конфигурации пространства имен.

4.2.1. web.xml Конфигурация

Первое, что вам нужно сделать, это добавить в файл web.xml следующее объявление фильтра:

<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 обеспечивает подключение к веб-инфраструктуре Spring Security. DelegatingFilterProxy- это класс Spring Framework, который делегирует реализацию фильтра, который определяется как компонент Spring в контексте вашего приложения. В этом случае компонент называется «springSecurityFilterChain», который представляет собой компонент внутренней инфраструктуры, созданный пространством имен для обеспечения веб-безопасности. Обратите внимание, что вам не следует использовать это имя bean-компонента самостоятельно. После того, как вы добавили это в свой web.xml, вы готовы приступить к редактированию файла контекста приложения. Сервисы веб-безопасности настраиваются с помощью элемента <http>.

4.2.2. Минимальная конфигурация <http>

**Аутентификация**

Создать файл **security.xml, а** в нем блок аутентификации <auth>:

<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>

Элемент <http> отвечает за создание FilterChainProxy и фильтровальные bean-ы, которые он использует. Распространенные проблемы, такие как неправильный порядок фильтров, больше не являются проблемой, поскольку положения фильтров предопределены.

Элемент <authentication-provider> создает компонент DaoAuthenticationProvider и элемент <user-service> создает InMemoryDaoImpl. Все элементы authentication-provider должны быть дочерними по отношению к элементу <authentication-manager>, который создает ProviderManagerи регистрирует с ним поставщиков аутентификации. Вы можете найти более подробную информацию о bean-компонентах, которые созданы в [приложении пространства имен](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#appendix-namespace) . Это стоит перепроверить, если вы хотите начать понимать, какие классы в фреймворке важны и как они используются, особенно если вы хотите настроить вещи позже.

Приведенная выше конфигурация определяет двух пользователей, их пароли и их роли.

Также можно загрузить информацию о пользователе из стандартного файла свойств с помощью атрибута properties on user-service. См. Раздел об [аутентификации в памяти](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#core-services-in-memory-service) для получения более подробной информации о формате файла.

Использование элемента <authentication-provider> означает, что информация о пользователе будет использоваться менеджером аутентификации для обработки запросов аутентификации.

У вас может быть несколько элементов <authentication-provider> для определения разных источников аутентификации, и каждый из них будет использоваться по очереди.

**Авторизация**

В файл **security.xml** дорописать блок авторизации <http>:

<http>

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

<form-login />

<logout />

</http>

Это говорит о том, что мы хотим:

- чтобы все URL-адреса "/\*\*" в нашем приложении были защищены, требуя для доступа к ним роли ROLE\_USER,

- войти в приложение, используя форму с именем пользователя и паролем (*для получения роли USER обязательно ввести логин и пароль*),

- зарегистрировать URL-адрес выхода, который позволит нам выйти из приложения.

Элемент <http> является родительским для всех функций пространства имен, связанных с Интернетом.

Вы можете использовать несколько <intercept-url> элементов для определения различных требований к доступу для разных наборов URL-адресов, но они будут оцениваться в указанном порядке, и будет использоваться первое совпадение. Таким образом, вы должны помещать наиболее конкретные совпадения вверху.

Атрибут pattern который сопоставляется с URL-адресами входящих запросов с использованием синтаксиса стиля ant path [ [2](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/" \l "_footnote_2" \o "См. Сноску.) ] .

Вы также можете использовать сопоставление регулярных выражений в качестве альтернативы (дополнительные сведения см. В приложении о пространствах имен).

Атрибут access определяет требования доступа для запросов, соответствующих данному шаблону. В конфигурации по умолчанию это обычно список ролей, разделенных запятыми, одна из которых должна быть разрешена пользователю для выполнения запроса.

Префикс «ROLE\_» - это маркер, который указывает, что необходимо провести простое сравнение с полномочиями пользователя. Другими словами, следует использовать обычную проверку на основе ролей. Контроль доступа в Spring Security не ограничивается использованием простых ролей (отсюда и использование префикса для различения различных типов атрибутов безопасности). Позже мы увидим, как может варьироваться интерпретация [ [3](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/" \l "_footnote_3" \o "См. Сноску.) ] .

Атрибут method можете добавить, чтобы ограничить соответствие определенному методу HTTP ( GET, POSTи PUTт.д.

На этом этапе вы сможете запустить свое приложение, и вам потребуется войти в систему, чтобы продолжить. Попробуйте это или попробуйте поэкспериментировать с "учебным" примером приложения, которое поставляется вместе с проектом.

4.2.3. Форма и основные параметры входа

Вам может быть интересно, откуда взялась форма входа, когда вам было предложено войти в систему, поскольку мы не упоминали какие-либо файлы HTML или JSP. Фактически, поскольку мы явно не установили URL-адрес для страницы входа в систему, Spring Security генерирует его автоматически на основе включенных функций и с использованием стандартных значений URL-адреса, который обрабатывает отправленный логин, целевой URL-адрес по умолчанию, который пользователь будет быть отправленным после входа в систему и так далее. Однако пространство имен предлагает множество возможностей для настройки этих параметров. Например, если вы хотите предоставить свою собственную страницу входа в систему, вы можете использовать:

<http>

<intercept-url pattern="/login.jsp\*" access="IS\_AUTHENTICATED\_ANONYMOUSLY"/>

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

<form-login login-page='/login.jsp'/>

</http>

Мы добавили дополнительный элемент intercept-url, чтобы сказать, что любые запросы для страницы входа должны быть доступны анонимным пользователям [ [4](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/" \l "_footnote_4" \o "См. Сноску.) ], . В противном случае запрос будет соответствовать шаблону /\*\*и доступ к самой странице входа в систему будет невозможен! Это распространенная ошибка конфигурации, которая приведет к бесконечному циклу в приложении. Spring Security выдаст предупреждение в журнале, если ваша страница входа окажется защищенной.

**Поддержка анонимного входа.**

По умолчанию Spring создает анонимную роль.

Поэтому, когда вы указываете роль как «ROLE\_ANONYMOUS» или «IS\_AUTHENTICATED\_ANONYMOUSLY», любой анонимный пользователь может просматривать эту страницу:

- В диспетчере принятия решений о присоединении AffirrativedBased RoleVoter предоставляет доступ, когда он видит атрибут доступа, установленный в «ROLE\_ANONYMOUS».

- Аналогичным образом **AuthenticatedVoter** предоставляет доступ, если для атрибута доступа установлено значение «IS\_AUTHENTICATED\_ANONYMOUSLY».

Также возможно, чтобы все запросы, соответствующие определенному шаблону, полностью исключали цепочку фильтров безопасности, путем определения отдельного элемента http для такого шаблона следующим образом:

<http pattern="/css/\*\*" security="none"/>

<http pattern="/login.jsp\*" security="none"/>

<http>

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

<form-login login-page='/login.jsp'/>

</http>

Начиная с Spring Security 3.1 теперь можно использовать несколько элементов http для определения отдельных конфигураций цепочки фильтров безопасности для разных шаблонов запросов. Если в элементе http отсутствует атрибут pattern, он соответствует всем запросам.

Создание незащищенного шаблона - простой пример этого синтаксиса, где шаблон отображается в пустую цепочку фильтров [[5](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#_footnote_5)] . Мы рассмотрим этот новый синтаксис более подробно в главе, посвященной [цепочке фильтров безопасности](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#filter-chains-with-ns) .

Важно понимать, что эти незащищенные запросы будут полностью игнорировать любую конфигурацию Spring Security, относящуюся к сети, или дополнительные атрибуты, такие как requires-channel, поэтому вы не сможете получить доступ к информации о текущем пользователе или вызывать защищенные методы во время запроса.

Если вы хотите, чтобы цепочка фильтров безопасности применялась необходимо использовать access='IS\_AUTHENTICATED\_ANONYMOUSLY'.

Если вы хотите использовать базовую аутентификацию вместо входа в систему, измените конфигурацию на

<http>

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

<http-basic />

</http>

В этом случае обычная проверка подлинности будет иметь приоритет и будет использоваться для запроса входа в систему, когда пользователь пытается получить доступ к защищенному ресурсу. Форма входа в систему по-прежнему доступна в этой конфигурации, если вы хотите ее использовать, например, через форму входа, встроенную в другую веб-страницу.

Базовая аутентификации HTTP (см. на сайте глава “Безопасность веб приложений” п.4.1) передает пароль в виде обычного текста и поэтому должна использоваться только в HTTPS. При использовании HTTP протокола должна использоваться, более “геморойная” в настройке, дайджест-аутентификация (см. на сайте глава “Безопасность веб приложений” п.4.2).

Установка URL-адреса по умолчанию после входа в систему

Если при попытке доступа к защищенному ресурсу не появляется запрос на вход в систему,  в игру вступает параметрdefault-target-url. Это URL-адрес, по которому пользователь перейдет после успешного входа в систему, по умолчанию - «/».

Вы также можете настроить так, чтобы пользователь *всегда* попадал на эту страницу (независимо от того, был ли вход в систему «по запросу» или он явно выбрал вход в систему), установив для атрибута always-use-default-target значение «true». Это полезно, если ваше приложение всегда требует, чтобы пользователь начинал с «домашней» страницы, например:

<http pattern="/login.htm\*" security="none"/>

<http>

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

<form-login login-page='/login.htm' default-target-url='/home.htm' always-use-default-target='true' />

</http>

\* Для еще большего контроля над местом назначения вы можете использовать атрибут authentication-success-handler-ref в качестве альтернативы default-target-url. Указанный компонент должен быть экземпляром AuthenticationSuccessHandler. Вы найдете больше об этом в главе « [Основные фильтры»,](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#form-login-flow-handling) а также в приложении к пространству имен, а также информацию о том, как настроить поток при сбое аутентификации.

4.2.4. Обработка выхода

Элемент logout добавляет поддержку для регистрации путем навигации к определенному URL. URL-адрес выхода по умолчанию - это “/j\_spring\_security\_logout”, но вы можете установить для него другое значение с помощью атрибута logout-url.

Также есть еще атрибут logout-success-url **-** целевой URL-адрес, на который пользователь перейдет после выхода из системы. По умолчанию это "/".

<http>

<intercept-url pattern=….

<form-login login-page=…

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

</http>

Более подробную информацию о других доступных атрибутах можно найти в Приложении в Пространстве имен п.2.1.20.

4.2.5. Использование других провайдеров аутентификации

На практике вам понадобится более масштабируемый источник пользовательской информации, чем несколько имен, добавленных в файл контекста приложения с помощью **<authentication-manager>**

**<authentication-provider> <user-service> <user name=………** . Скорее всего, вы захотите хранить информацию о пользователе в чем-то вроде Базы Данных **или** на сервере LDAP. Конфигурация пространства имен LDAP рассматривается в [главе](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ldap) о [LDAP](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ldap) , поэтому мы не будем рассматривать ее здесь.

1. Если у вас есть настраиваемая реализация UserDetailsService Spring Security, называемая например «myUserDetailsService» в контексте вашего приложения, то вы можете выполнить аутентификацию с помощью:

<authentication-manager>

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

</authentication-manager>

2. Если вы хотите использовать Базу Данных, вы можете использовать:

<authentication-manager>

<authentication-provider>

<**jdbc**-user-service data-source-ref="securityDataSource"/>

</authentication-provider>

</authentication-manager>

,где "securityDataSource" - это имя bean-компонента DataSource в контексте приложения, указывающее на базу данных, содержащую стандартные [таблицы пользовательских данных](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#db_schema_users_authorities) Spring Security. !!!Но тут не написано (как в п. 3.6.2.) куда добавлять “.usersByUsernameQuery("SELECT USERNAME, PASSWORD, ENABLED FROM USER WHERE USERNAME=?")” и “.authoritiesByUsernameQuery("SELECT U.USERNAME, A.AUTHORITY FROM AUTHORITIES A, USER U WHERE U.USERNAME = A.USERNAME AND U.USERNAME = ?")” !!!

3. В качестве альтернативы вы можете настроить bean-компонент JdbcDaoImpl Spring Security и указать на него с помощью атрибута 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>

4. Вы также можете использовать стандартные beans AuthenticationProvider следующим образом:

<authentication-manager>

<authentication-provider ref='myAuthenticationProvider'/>

</authentication-manager>

,где 'myAuthenticationProvider' - имя компонента в контексте вашего приложения, который реализует AuthenticationProvider.

5. Вы можете использовать несколько элементов authentication-provider, и в этом случае провайдеры будут опрашиваться в том порядке, в котором они объявлены. Более подробную информацию о authentication-provider можно найти в Приложении в Пространстве имен п.2.2.2.

4.2.6. Добавление кодировщика паролей

Пароли всегда должны кодироваться с использованием безопасного алгоритма хеширования, разработанного для этой цели (а не стандартного алгоритма, такого как SHA или MD5). Это поддерживается элементом <password-encoder>.

С паролями в кодировке **bcrypt**(более медленная работа по сравнению с одной фазой SHA512, что в этом случаи является преимуществом т.к. исключает использование тысяч паралельных потоков для расшифровки) исходная конфигурация провайдера аутентификации будет выглядеть так:

<beans:bean name="bcryptEncoder"

class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>

<authentication-manager>

<authentication-provider>

<password-encoder ref="bcryptEncoder"/>

<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>

Bcrypt - хороший выбор для большинства случаев, если у вас нет устаревшей системы, которая заставляет вас использовать другой алгоритм. Если вы используете простой алгоритм хеширования или, что еще хуже, храните пароли в виде простого текста, вам следует подумать о переходе на более безопасный вариант, такой как bcrypt.

4.3. Расширенные возможности Интернета

4.3.1. Remember-me – шифрование куков

Подробную информацию о всех доступных атрибутах можно найти в Приложении в Пространстве имен п.2.1.26.

 Обзор

Аутентификация «remember-me» означает, что веб-сайты могут запоминать личность участника между сеансами. Обычно это достигается путем отправки файла cookie **в** браузер, который обнару**живается** во время будущих сеансов и вызывает автоматический вход в систему.

Spring Security предоставляет необходимые механизмы для выполнения этих операций и имеет две конкретные реализации «remember-me»:

- использование хеширования для обеспечения безопасности токенов на основе файлов cookie,

- использование базы данных (или другой механизм постоянного хранения) для хранения сгенерированных токенов.

Обратите внимание, что обе реализации требуют UserDetailsService. Если вы используете провайдер аутентификации, который не использует UserDetailsService(например, провайдер LDAP), он не будет работать, если у вас также нет bean-компонента UserDetailsService в контексте вашего приложения.

 Простой подход на основе **хеш-токенов**

Этот подход использует хеширование для достижения полезной стратегии «remember-me». По сути, файл cookie отправляется в браузер после успешной интерактивной аутентификации, при этом файл cookie состоит из следующего:

base64(username + ":" + expirationTime + ":" +

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

*,где*

username: As identifiable to the `UserDetailsService`

password: Это совпадает с полученным UserDetails

expirationTime: Дата и время истечения срока действия токена "remember-me",

выраженное в миллисекундах

key: Закрытый ключ для предотвращения модификации токена "remember-me"

base64() - метод для применения алгоритма кодирования base64

md5Hex() – метод для применения алгоритма хеширования MD5 и получения хеша

Т.е. тут получают хеш с помощью алгоритма MD5, после чего кодируют хеш с доп. параметрами с помощью алгоритма base64.

Запоминающийся токен действителен только в течение указанного периода и при условии, что имя пользователя, пароль и ключ не меняются. Примечательно, что это имеет потенциальную проблему безопасности, поскольку захваченный токен «remember-me» будет использоваться любым пользовательским агентом до тех пор, пока токен не истечет. Это та же проблема, что и при дайджест-аутентификации.

Если принципал знает, что токен был захвачен, он может легко изменить свой пароль и немедленно аннулировать все запоминающиеся токены, о которых идет речь.

Если требуется более серьезная безопасность, вам следует использовать подход, описанный в следующем разделе. В качестве альтернативы сервисы «remember-me» вообще не следует использовать.

Вы можете включить аутентификацию «remember-me» на основе хеш-токенов, просто добавив элемент <remember-me>:

<http>

...

<remember-me key="myAppKey"/>

</http>

key - Должно быть установлено уникальное значение. Если этот параметр не установлен, будет сгенерировано безопасное случайное значение. Поскольку создание безопасных случайных значений может занять некоторое время, явная установка этого значения может помочь сократить время запуска при использовании функции «remember-me».

Обычно UserDetailsService выбирается автоматически.

Если у вас более одного в контексте вашего приложения, вам необходимо указать, какой из них следует использовать с атрибутом user-service-ref, где значением является имя вашего bean-компонента UserDetailsService.

 Подход с **постоянным токеном**

Чтобы использовать этот подход с конфигурацией пространства имен, вы должны предоставить ссылку на источник данных:

<http>

...

<remember-me data-source-ref="someDataSource"/>

</http>

База данных должна содержать persistent\_loginsтаблицу, созданную с использованием следующего SQL (или эквивалента):

create table persistent\_logins (username varchar(64) not null,

series varchar(64) primary key,

token varchar(64) not null,

last\_used timestamp not null)

4.3.2. Добавление безопасности канала HTTP / HTTPS

Если ваше приложение поддерживает как HTTP, так и HTTPS, и вам требуется, чтобы определенные URL-адреса могли быть доступны только через HTTPS, то это напрямую поддерживается с помощью атрибута requires-channel на <intercept-url>:

<http>

<intercept-url pattern="/secure/\*\*" access="ROLE\_USER" requires-channel="https"/>

<intercept-url pattern="/\*\*" access="ROLE\_USER" requires-channel="any"/>

...

</http>

При такой конфигурации, если пользователь пытается получить доступ к чему-либо, совпадающему с шаблоном «/secure/\*\*», используя HTTP, он сначала будет перенаправлен на URL-адрес HTTPS [ [6](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/" \l "_footnote_6" \o "См. Сноску.) ] . Доступные варианты: «http», «https» или «любой». Использование значения «любой» означает, что можно использовать HTTP или HTTPS.

Если ваше приложение использует нестандартные порты для HTTP и / или HTTPS, вы можете указать список сопоставлений портов следующим образом:

<http>

...

<port-mappings>

<port-mapping http="9080" https="9443"/>

</port-mappings>

</http>

Обратите внимание, что для обеспечения полной безопасности приложение не должно вообще использовать HTTP или переключаться между HTTP и HTTPS. Он должен начинаться с HTTPS (с вводом пользователем URL-адреса HTTPS) и использовать защищенное соединение повсюду, чтобы избежать любой возможности атак типа «злоумышленник в середине».

4.3.3. Управление сессией

Обнаружение тайм-аутов

Вы можете настроить Spring Security для обнаружения отправки недопустимого идентификатора сеанса и перенаправления пользователя на соответствующий URL-адрес. Это достигается с помощью элемента session-management:

<http>

...

<session-management invalid-session-url="/invalidSession.htm" />

</http>

Обратите внимание: если вы используете этот механизм для определения тайм-аутов сеанса, он может ложно сообщить об ошибке, если пользователь выйдет из системы, а затем снова войдет в систему, не закрывая браузер. Это связано с тем, что файл cookie сеанса не очищается, когда вы аннулируете сеанс, и будет повторно отправлен, даже если пользователь вышел из системы.

Вы можете явно удалить файл cookie JSESSIONID при выходе из системы, например, используя следующий синтаксис в обработчике выхода:

<http>

<logout delete-cookies="JSESSIONID" />

</http>

К сожалению, это не может гарантировать работу с каждым контейнером сервлетов, поэтому вам нужно будет протестировать его в своей среде.

|  |  |
| --- | --- |
|  | Если вы запускаете свое приложение за прокси-сервером, вы также можете удалить cookie сеанса, настроив прокси-сервер. Например, при использовании mod\_headers Apache HTTPD следующая директива удалит файл cookie JSESSIONID, как с истекшим сроком действия, в ответ на запрос о выходе (при условии, что приложение развернуто по указанному пути /tutorial):  <LocationMatch "/tutorial/j\_spring\_security\_logout">  Header always set Set-Cookie "JSESSIONID=;Path=/tutorial;Expires=Thu, 01 Jan 1970 00:00:00 GMT"  </LocationMatch> |

Управление одновременным сеансом

Если вы хотите наложить ограничения на возможность одного пользователя войти в ваше приложение одновременно только определенное количество раз, Spring Security поддерживает это из коробки со следующими простыми дополнениями. Сначала для того чтобы Spring Security был в курсе событий жизненного цикла сеанса вам нужно добавить в файл web.xml следующего слушателя:

<listener>

<listener-class>

org.springframework.security.web.session.HttpSessionEventPublisher

</listener-class>

</listener>

Затем добавьте следующие строки в контекст вашего приложения:

<http>

...

<session-management>

<concurrency-control max-sessions="1" />

</session-management>

</http>

,это предотвратит многократный вход пользователя в систему - второй вход приведет к тому, что первый вход будет признан недействительным.

Часто вы предпочитаете предотвратить второй вход, и в этом случае вы можете использовать:

<http>

...

<session-management>

<concurrency-control max-sessions="1" error-if-maximum-exceeded="true" />

</session-management>

</http>

Второй логин будет отклонен. Под «отклоненным» мы подразумеваем, что пользователь будет отправлен в систему, authentication-failure-urlесли используется логин на основе формы. Если вторая аутентификация происходит через другой неинтерактивный механизм, такой как «remember me», клиенту будет отправлена ​​«неавторизованная» (401) ошибка. Если вместо этого вы хотите использовать страницу с ошибкой, вы можете добавить атрибут session-authentication-error-urlк session-managementэлементу.

Если вы используете настраиваемый фильтр проверки подлинности для входа на основе формы, вам необходимо явно настроить поддержку управления параллельными сеансами. Более подробную информацию можно найти в [главе «Управление сеансом»](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#session-mgmt) .

Защита от атак фиксации сеанса

Атаки [фиксации сеанса](https://en.wikipedia.org/wiki/Session_fixation) представляют собой потенциальный риск, когда злоумышленник может создать сеанс, получив доступ к сайту, а затем убедить другого пользователя войти в систему с тем же сеансом (отправив ему ссылку, содержащую идентификатор сеанса в качестве параметра).

Spring Security защищает от этого автоматически!!!, создавая новый сеанс или иным образом изменяя идентификатор сеанса при входе пользователя в систему. Если вам не нужна эта защита или она конфликтует с каким-либо другим требованием, вы можете управлять поведением с помощью session-fixation-protectionатрибута on <session-management>, который имеет четыре варианта

* none- Ничего не делай. Исходный сеанс будет сохранен.
* newSession - Создайте новый «чистый» сеанс без копирования данных существующего сеанса (атрибуты, связанные с Spring Security, все равно будут скопированы).
* migrateSession- Создайте новый сеанс и скопируйте все существующие атрибуты сеанса в новый сеанс. Это значение по умолчанию в контейнерах Servlet 3.0 или более старых.
* changeSessionId- Не создавайте новую сессию. Вместо этого используйте защиту от фиксации сеанса, предоставляемую контейнером сервлета ( HttpServletRequest#changeSessionId()). Этот параметр доступен только в Servlet 3.1 (Java EE 7) и более новых контейнерах. Если указать его в старых контейнерах, возникнет исключение. Это значение по умолчанию в контейнерах Servlet 3.1 и более новых.

Когда происходит защита фиксации сеанса, это приводит к публикации SessionFixationProtectionEvent в контексте приложения. Если вы используете changeSessionId, эта защита также приведет к уведомлению любого javax.servlet.http.HttpSessionIdListener s, поэтому будьте осторожны, если ваш код прослушивает оба события. См. Дополнительную информацию в главе « [Управление](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#session-mgmt) сеансом».

4.3.4. Поддержка OpenID

Пространство имен поддерживает вход [OpenID](https://openid.net/) либо вместо, либо в дополнение к обычному входу на основе формы, с простым изменением:

<http>

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

<openid-login />

</http>

Затем вам следует зарегистрироваться у поставщика OpenID (например, myopenid.com) и добавить информацию о пользователе в свою оперативную память <user-service>:

<user name="http://jimi.hendrix.myopenid.com/" authorities="ROLE\_USER" />

Вы должны иметь возможность войти в систему, используя сайт myopenid.com для аутентификации. Также можно выбрать конкретный bean-компонент UserDetailsService для использования OpenID, установив атрибут user-service-ref для элемента openid-login. См. Предыдущий раздел о [поставщиках аутентификации](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-auth-providers) для получения дополнительной информации. Обратите внимание, что мы опустили атрибут пароля в приведенной выше конфигурации пользователя, поскольку этот набор пользовательских данных используется только для загрузки полномочий для пользователя. Случайный пароль будет сгенерирован внутри, чтобы вы не могли случайно использовать эти данные пользователя в качестве источника аутентификации где-либо еще в вашей конфигурации.

Обмен атрибутами

Поддержка [обмена атрибутами](https://openid.net/specs/openid-attribute-exchange-1_0.html) OpenID . Например, следующая конфигурация попытается получить адрес электронной почты и полное имя от поставщика OpenID для использования приложением:

<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>

Атрибут type каждого атрибута OpenID - это URI, определяемый конкретной схемой, в данном случае <http://axschema.org/> .

Атрибут required устанавливается если атрибут должен быть получен для успешной аутентификации.

Точная поддерживаемая схема и атрибуты будут зависеть от вашего провайдера OpenID.

Значения атрибутов возвращаются как часть процесса аутентификации и могут быть доступны впоследствии с помощью следующего кода:

OpenIDAuthenticationToken token =

(OpenIDAuthenticationToken)SecurityContextHolder.getContext().getAuthentication();

List<OpenIDAttribute> attributes = token.getAttributes();

OpenIDAttribute содержит тип атрибута и извлеченное значение (или значения в случае многозначных атрибутов). Мы увидим больше о том, как используется класс SecurityContextHolder, когда рассмотрим основные компоненты Spring Security в главе [технического обзора](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#core-components).

Также поддерживаются конфигурации обмена несколькими атрибутами, если вы хотите использовать несколько поставщиков удостоверений. Вы можете предоставить несколько элементов <attribute-exchange>, используя для каждого атрибут identifier-matcher. Он содержит регулярное выражение, которое будет сопоставлено с идентификатором OpenID, предоставленным пользователем. См. Пример приложения OpenID в базе кода для примера конфигурации, предоставляющего различные списки атрибутов для поставщиков Google, Yahoo и MyOpenID.

4.3.5. Заголовки ответа для безопасности

В этом разделе обсуждается поддержка Spring Security для добавления в ответ различных заголовков безопасности.

Заголовки безопасности по умолчанию

Spring Security позволяет пользователям легко вводить заголовки безопасности по умолчанию, чтобы помочь защитить свое приложение. Ниже приводится список текущих *заголовков безопасности по умолчанию,* предоставленных Spring Security:

* [Управление кешем](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-cache-control)
* [Параметры типа контента](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-content-type-options)
* [Строгая безопасность транспорта HTTP](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-hsts)
* [X-Frame-Опции](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-frame-options)
* [X-XSS-защита](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-xss-protection)

Хотя каждый из этих заголовков считается передовой практикой, следует отметить, что не все клиенты используют заголовки, поэтому рекомендуется дополнительное тестирование.

По причинам пассивности, если вы используете поддержку пространства имен XML Spring Security, вы должны явно включить заголовки безопасности. Все заголовки по умолчанию можно легко добавить с помощью элемента [<headers](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-headers) /> без дочерних элементов:

|  |  |
| --- | --- |
|  | [SEC-2348](https://jira.springsource.org/browse/SEC-2348) регистрируется, чтобы гарантировать, что конфигурация пространства имен XML Spring Security 4.x по умолчанию включает заголовки безопасности. |

<http>

<!-- ... -->

<headers />

</http>

Кроме того, вы можете явно указать заголовки, которые хотите включить. Например, следующая конфигурация аналогична предыдущей. Удаление любого из элементов приведет к удалению этого заголовка из ответов.

<http>

<!-- ... -->

<headers>

<cache-control />

<content-type-options />

<hsts />

<frame-options />

<xss-protection />

</headers>

</http>

Если вы используете конфигурацию Java Spring Security, а не security.xml, все “Заголовки безопасности по умолчанию” добавляются по умолчанию.

Их можно отключить, используя конфигурацию Java ниже:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers().disable();

}

}

Как только вы укажете какие-либо заголовки, которые должны быть включены, будут включены только эти заголовки. Например, следующая конфигурация будет включать поддержку только [Cache Control](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-cache-control) и [X-Frame-Options](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-frame-options) .

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.cacheControl()

.frameOptions();

}

}

- Управление кешем

В прошлом Spring Security требовала от вас предоставления собственного управления кешем для вашего веб-приложения. В то время это казалось разумным, но кеши браузера эволюционировали и теперь включают кеши для безопасных соединений. Это означает, что пользователь может просмотреть аутентифицированную страницу, выйти из системы, а затем злоумышленник может использовать историю браузера для просмотра кэшированной страницы. Чтобы помочь смягчить эту проблему, Spring Security добавила поддержку управления кешем, которая будет вставлять следующие заголовки в ваш ответ.

Cache-Control: no-cache, no-store, max-age=0, must-revalidate

Pragma: no-cache

Expires: 0

Простое добавление элемента [<headers](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-headers) /> без дочерних элементов автоматически добавит Cache Control и ряд других средств защиты. Однако, если вам нужен только контроль кеша, вы можете включить эту функцию, используя пространство имен XML Spring Security с элементом [<cache-control](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-cache-control) >.

<http>

<!-- ... -->

<headers>

<cache-control />

</headers>

</http>

Точно так же вы можете включить только управление кешем в конфигурации Java со следующим:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.cacheControl();

}

}

Если вы действительно хотите кэшировать определенные ответы, ваше приложение может выборочно вызывать [HttpServletResponse.setHeader (String, String),](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html#setHeader(java.lang.String,java.lang.String)) чтобы переопределить заголовок, установленный Spring Security. Это полезно для обеспечения правильного кеширования таких вещей, как CSS, JavaScript и изображения.

При использовании Spring Web MVC это обычно делается в вашей конфигурации. Например, следующая конфигурация гарантирует, что заголовки кеша установлены для всех ваших ресурсов:

@EnableWebMvc

public class WebMvcConfiguration extends WebMvcConfigurerAdapter {

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry

.addResourceHandler("/resources/\*\*")

.addResourceLocations("/resources/")

.setCachePeriod(31556926);

}

// ...

}

- Параметры типа контента

Исторически браузеры, в том числе Internet Explorer, пытались угадать тип контента запроса, используя [анализ контента](https://en.wikipedia.org/wiki/Content_sniffing) . Это позволило браузерам улучшить взаимодействие с пользователем, угадывая тип контента для ресурсов, для которых не указан тип контента. Например, если браузер обнаружил файл JavaScript, для которого не указан тип содержимого, он сможет угадать тип содержимого и затем выполнить его.

|  |  |
| --- | --- |
|  | Есть много дополнительных вещей, которые можно сделать (например, отображать документ только в отдельном домене, убедиться, что заголовок Content-Type установлен, очистить документ и т.д.), Когда разрешить загрузку контента. Однако эти меры выходят за рамки того, что предоставляет Spring Security. Также важно отметить, что при отключении сниффинга контента вы должны указать тип контента, чтобы все работало правильно. |

Проблема со сниффингом контента заключается в том, что это позволяло злоумышленникам использовать полиглоты (то есть файл, который действителен как несколько типов контента) для выполнения XSS-атак. Например, некоторые сайты могут разрешать пользователям отправлять на сайт действительный документ PostScript и просматривать его. Злоумышленник может создать [документ postscript, который также является допустимым файлом JavaScript,](http://webblaze.cs.berkeley.edu/papers/barth-caballero-song.pdf) и выполнить с ним атаку XSS.

Отслеживание контента можно отключить, добавив к нашему ответу следующий заголовок:

X-Content-Type-Options: nosniff

Как и в случае с элементом управления кешем, директива nosniff добавляется по умолчанию при использовании элемента <headers> без дочерних элементов. Однако, если вам нужен больший контроль над добавлением заголовков, вы можете использовать элемент [<content-type-options](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-content-type-options) >, как показано ниже:

<http>

<!-- ... -->

<headers>

<content-type-options />

</headers>

</http>

Заголовок X-Content-Type-Options добавляется по умолчанию с конфигурацией Java Spring Security. Если вам нужен больший контроль над заголовками, вы можете явно указать параметры типа контента с помощью следующего:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.contentTypeOptions();

}

}

- Строгая безопасность транспорта HTTP (HSTS)

Когда вы вводите веб-сайт своего банка, вы вводите mybank.example.com или [https://mybank.example.com](https://mybank.example.com/)? Если вы опустите протокол https, вы потенциально уязвимы для [атак «Человек посередине»](https://en.wikipedia.org/wiki/Man-in-the-middle_attack) . Даже если веб-сайт выполняет перенаправление на [https://mybank.example.com,](https://mybank.example.com/) злоумышленник может перехватить первоначальный HTTP-запрос и манипулировать ответом (т.е. перенаправить на [https://mibank.example.com](https://mibank.example.com/) и украсть их учетные данные).

Многие пользователи пропускают протокол https, поэтому была создана [строгая безопасность транспорта HTTP (HSTS)](https://tools.ietf.org/html/rfc6797) . После добавления mybank.example.com в качестве [узла HSTS](https://tools.ietf.org/html/rfc6797#section-5.1) браузер может заранее знать, что любой запрос к mybank.example.com следует интерпретировать как [https://mybank.example.com](https://mybank.example.com/) . Это значительно снижает вероятность возникновения атаки «человек посередине».

|  |  |
| --- | --- |
|  | В соответствии с [RFC6797](https://tools.ietf.org/html/rfc6797#section-7.2) заголовок HSTS [вставляется](https://tools.ietf.org/html/rfc6797#section-7.2) только в ответы HTTPS. Чтобы браузер подтвердил заголовок, браузер должен сначала доверять ЦС, подписавшему сертификат SSL, используемый для установления соединения (а не только сертификат SSL). |

Один из способов пометить сайт как хост HSTS - предварительно загрузить этот хост в браузер. Другой - добавить к ответу заголовок «Strict-Transport-Security». Например, следующее может дать браузеру указание рассматривать домен как хост HSTS в течение года (примерно 31536000 секунд в году):

Strict-Transport-Security: max-age=31536000 ; includeSubDomains

Необязательная директива includeSubDomains указывает Spring Security, что поддомены (например, secure.mybank.example.com) также должны рассматриваться как домен HSTS.

Как и в случае с другими заголовками, Spring Security добавляет предыдущий заголовок к ответу, когда элемент <headers> указан без дочерних элементов. Он также автоматически добавляется, когда вы используете конфигурацию Java. Вы также можете использовать только заголовки HSTS с элементом [<hsts](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-hsts) >, как показано ниже:

<http>

<!-- ... -->

<headers>

<hsts />

</headers>

</http>

Точно так же вы можете включить только заголовки HSTS с конфигурацией Java:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.hsts();

}

}

- X-Frame-Опции

Разрешение добавления вашего веб-сайта во фрейм может быть проблемой для безопасности. Например, используя умную стилизацию CSS, можно обманом заставить пользователей щелкнуть то, чего они не собирались делать ( [видео-демонстрация](https://www.youtube.com/watch?v=3mk0RySeNsU) ). Например, пользователь, вошедший в свой банк, может нажать кнопку, которая предоставляет доступ другим пользователям. Этот вид атак известен как http://en.wikipedia.org/wiki/Clickjacking [Clickjacking].

|  |  |
| --- | --- |
|  | Другой современный подход к борьбе с кликджекингом - использование [политики безопасности контента](https://www.w3.org/TR/CSP/) . Spring Security не поддерживает это, поскольку спецификация еще не выпущена, и она немного сложнее. Однако для реализации этого вы можете использовать функцию [статических заголовков](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-static) . Чтобы быть в курсе этой проблемы и узнать, как ее можно реализовать с помощью Spring Security, см. [SEC-2117.](https://jira.springsource.org/browse/SEC-2117) |

Есть несколько способов смягчить атаки кликджекинга. Например, для защиты устаревших браузеров от атак типа «кликджекинг» можно использовать [код разрыва фрейма](https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet#Best-for-now_Legacy_Browser_Frame_Breaking_Script) . Хотя код разбиения фреймов и не идеален, это лучшее, что вы можете сделать для старых браузеров.

Более современный подход к решению проблемы кликджекинга - использование заголовка [X-Frame-Options](https://developer.mozilla.org/en-US/docs/HTTP/X-Frame-Options) :

X-Frame-Options: DENY

Заголовок ответа X-Frame-Options предписывает браузеру запретить отображение любого сайта с этим заголовком в ответе во фрейме. Как и другие заголовки ответа, он автоматически включается, если элемент <headers> не указан без дочерних элементов. Вы также можете явно указать элемент [frame-options,](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-frame-options) чтобы контролировать, какие заголовки добавляются в ответ.

<http>

<!-- ... -->

<headers>

<frame-options />

</headers>

</http>

Точно так же вы можете включить только параметры фрейма в конфигурации Java со следующим:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.frameOptions();

}

}

Если вы хотите изменить значение заголовка X-Frame-Options, вы можете использовать [экземпляр XFrameOptionsHeaderWriter](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#headers-headers-writer) .

- X-XSS-защита

Некоторые браузеры имеют встроенную поддержку для фильтрации [отраженных XSS-атак](https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OWASP-DV-001)) . Это ни в коем случае не является полным доказательством, но помогает в защите от XSS.

Фильтрация обычно включена по умолчанию, поэтому добавление заголовка обычно просто гарантирует, что она включена, и указывает браузеру, что делать при обнаружении XSS-атаки. Например, фильтр может попытаться изменить содержимое наименее инвазивным способом, чтобы все равно отображать все. Иногда такая замена [сама по себе](https://hackademix.net/2009/11/21/ies-xss-filter-creates-xss-vulnerabilities/) может стать [XSS-уязвимостью](https://hackademix.net/2009/11/21/ies-xss-filter-creates-xss-vulnerabilities/) . Вместо этого лучше заблокировать контент, чем пытаться его исправить. Для этого мы можем добавить следующий заголовок:

X-XSS-Protection: 1; mode=block

Этот заголовок включается по умолчанию, если элемент <headers> не указан без дочерних элементов. Мы можем явно указать это, используя элемент [xss-protection,](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-xss-protection) как показано ниже:

<http>

<!-- ... -->

<headers>

<xss-protection />

</headers>

</http>

Точно так же вы можете включить только защиту xss в конфигурации Java со следующим:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.xssProtection();

}

}

Пользовательские заголовки

Spring Security имеет механизмы, позволяющие удобно добавлять в ваше приложение более общие заголовки безопасности. Однако он также предоставляет хуки, позволяющие добавлять собственные заголовки.

- Статические заголовки

Бывают случаи, когда вы хотите добавить в приложение настраиваемые заголовки безопасности, которые не поддерживаются из коробки. Например, возможно, вы хотите получить раннюю поддержку [политики безопасности контента](https://www.w3.org/TR/CSP/) , чтобы гарантировать, что ресурсы загружаются только из одного источника. Поскольку поддержка политики безопасности содержимого еще не завершена, браузеры используют один из двух общих заголовков расширений для реализации этой функции. Это означает, что нам нужно будет дважды ввести политику. Пример заголовков можно увидеть ниже:

X-Content-Security-Policy: default-src 'self'

X-WebKit-CSP: default-src 'self'

При использовании пространства имен XML эти заголовки можно добавить в ответ с помощью элемента [<header](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-header) >, как показано ниже:

<http>

<!-- ... -->

<headers>

<header name="X-Content-Security-Policy" value="default-src 'self'"/>

<header name="X-WebKit-CSP" value="default-src 'self'"/>

</headers>

</http>

Точно так же заголовки могут быть добавлены к ответу с помощью конфигурации Java, как показано ниже:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.addHeaderWriter(new StaticHeaderWriter("X-Content-Security-Policy","default-src 'self'"))

.addHeaderWriter(new StaticHeaderWriter("X-WebKit-CSP","default-src 'self'"));

}

}

- Автор заголовков

Если пространство имен или конфигурация Java не поддерживает нужные вам заголовки, вы можете создать собственный экземпляр HeadersWriter или даже предоставить собственную реализацию HeadersWriter.

Давайте посмотрим на пример использования настраиваемого экземпляра XFrameOptionsHeaderWriter. Возможно, вы хотите разрешить создание контента для одного и того же происхождения. Это легко поддерживается установкой для атрибута [policy значения](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-frame-options-policy) «SAMEORIGIN», но давайте рассмотрим более явный пример с использованием атрибута [ref](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-header-ref) .

<http>

<!-- ... -->

<headers>

<header ref="frameOptionsWriter"/>

</headers>

</http>

<!-- Requires the c-namespace.

See http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/beans.html#beans-c-namespace

-->

<beans:bean id="frameOptionsWriter"

class="org.springframework.security.web.header.writers.frameoptions.XFrameOptionsHeaderWriter"

c:frameOptionsMode="SAMEORIGIN"/>

Мы также могли бы ограничить создание контента одним и тем же источником с конфигурацией Java:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// ...

.headers()

.addHeaderWriter(new XFrameOptionsHeaderWriter(XFrameOptionsMode.SAMEORIGIN));

}

}

- DelegatingRequestMatcherHeaderWriter

Иногда вам может понадобиться написать заголовок только для определенных запросов. Например, возможно, вы хотите защитить только свою страницу входа от фрейма. Вы можете использовать DelegatingRequestMatcherHeaderWriterдля этого. При использовании конфигурации пространства имен XML это можно сделать следующим образом:

<http>

<!-- ... -->

<headers>

<header header-ref="headerWriter"/>

</headers>

</http>

<beans:bean id="headerWriter"

class="org.springframework.security.web.header.writers.DelegatingRequestMatcherHeaderWriter">

<beans:constructor-arg>

<bean class="org.springframework.security.web.util.matcher.AntPathRequestMatcher"

c:pattern="/login"/>

</beans:constructor-arg>

<beans:constructor-arg>

<beans:bean

class="org.springframework.security.web.header.writers.frameoptions.XFrameOptionsHeaderWriter"/>

</beans:constructor-arg>

</beans:bean>

Мы также могли бы предотвратить размещение контента на странице входа в систему, используя конфигурацию java:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

RequestMatcher matcher = new AntPathRequestMatcher("/login");

DelegatingRequestMatcherHeaderWriter headerWriter =

new DelegatingRequestMatcherHeaderWriter(matcher,new XFrameOptionsHeaderWriter());

http

// ...

.headers()

.addHeaderWriter(headerWriter);

}

}

4.3.6. Добавление собственных фильтров

Если вы раньше использовали Spring Security, вы знаете, что фреймворк поддерживает цепочку фильтров для применения своих сервисов. Вы можете добавить свои собственные фильтры в стек в определенных местах или использовать фильтр Spring Security, для которого в настоящее время нет параметра конфигурации пространства имен (например, CAS). Или вы можете захотеть использовать настроенную версию стандартного фильтра пространства имен, такого как фильтр, UsernamePasswordAuthenticationFilterкоторый создается элементом <form-login>, используя преимущества некоторых дополнительных параметров конфигурации, которые доступны при явном использовании bean-компонента. Как вы можете сделать это с помощью конфигурации пространства имен, если цепочка фильтров не отображается напрямую?

При использовании пространства имен всегда строго соблюдается порядок фильтров. Когда создается контекст приложения, компоненты фильтра сортируются по коду обработки пространства имен, и каждый стандартный фильтр Spring Security имеет псевдоним в пространстве имен и известную позицию.

|  |  |
| --- | --- |
|  | В предыдущих версиях сортировка выполнялась после создания экземпляров фильтра во время пост-обработки контекста приложения. В версии 3.0+ сортировка теперь выполняется на уровне метаданных компонента, прежде чем классы будут созданы. Это влияет на то, как вы добавляете собственные фильтры в стек, поскольку весь список фильтров должен быть известен во время синтаксического анализа элемента <http>, поэтому в версии 3.0 синтаксис немного изменился. |

В таблице показаны фильтры, псевдонимы и элементы/атрибуты пространства имен, которые создают фильтры. Фильтры перечислены в том порядке, в котором они встречаются в цепочке фильтров.

| **Таблица 1. Стандартные псевдонимы фильтров и их порядок** | | |
| --- | --- | --- |
| **Псевдоним** | **Класс фильтра** | **Элемент или атрибут пространства имен** |
| CHANNEL\_FILTER | ChannelProcessingFilter | http/intercept-url@requires-channel |
| SECURITY\_CONTEXT\_FILTER | SecurityContextPersistenceFilter | http |
| CONCURRENT\_SESSION\_FILTER | ConcurrentSessionFilter | session-management/concurrency-control |
| HEADERS\_FILTER | HeaderWriterFilter | http/headers |
| CSRF\_FILTER | CsrfFilter | http/csrf |
| LOGOUT\_FILTER | LogoutFilter | http/logout |
| X509\_FILTER | X509AuthenticationFilter | http/x509 |
| PRE\_AUTH\_FILTER | AstractPreAuthenticatedProcessingFilter Подклассы | Нет данных |
| CAS\_FILTER | CasAuthenticationFilter | Нет данных |
| FORM\_LOGIN\_FILTER | UsernamePasswordAuthenticationFilter | http/form-login |
| BASIC\_AUTH\_FILTER | BasicAuthenticationFilter | http/http-basic |
| SERVLET\_API\_SUPPORT\_FILTER | SecurityContextHolderAwareRequestFilter | http/@servlet-api-provision |
| JAAS\_API\_SUPPORT\_FILTER | JaasApiIntegrationFilter | http/@jaas-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 | Нет данных |

Вы можете добавить свой собственный фильтр в стек, используя элемент custom-filter и одно из этих имен, чтобы указать позицию, в которой должен отображаться ваш фильтр:

<http>

<custom-filter position="FORM\_LOGIN\_FILTER" ref="myFilter" />

</http>

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

Вы также можете использовать атрибуты after или before если хотите, чтобы ваш фильтр был вставлен до или после другого фильтра в стеке. Имена «FIRST» и «LAST» могут использоваться с атрибутом position, чтобы указать, что вы хотите, чтобы ваш фильтр отображался до или после всего стека соответственно.

|  |  |
| --- | --- |
|  | **Предотвращение конфликтов позиций фильтров**  Если вы вставляете настраиваемый фильтр, который может занимать ту же позицию, что и один из стандартных фильтров, созданных пространством имен, важно, чтобы вы по ошибке не включили версии пространства имен. Удалите все элементы, которые создают фильтры, функциональность которых вы хотите заменить.  Обратите внимание, что вы не можете заменять фильтры, созданные с помощью самого <http>элемента - SecurityContextPersistenceFilter, ExceptionTranslationFilterили FilterSecurityInterceptor. Некоторые другие фильтры добавляются по умолчанию, но вы можете их отключить. По AnonymousAuthenticationFilterумолчанию добавляется и, если у вас не отключена [защита от фиксации сеанса](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-session-fixation) , в SessionManagementFilterцепочку фильтров также будет добавлен. |

Если вы заменяете фильтр пространства имен, для которого требуется точка входа аутентификации (т.е. когда процесс аутентификации запускается попыткой неаутентифицированного пользователя получить доступ к защищенному ресурсу), вам также потребуется добавить пользовательский компонент точки входа.

Настройка пользовательской аутентификацииEntryPoint

Если вы не используете форму входа в систему, OpenID или базовую аутентификацию через пространство имен, вы можете определить фильтр аутентификации и точку входа, используя традиционный синтаксис bean-компонентов, и связать их с пространством имен, как мы только что видели. Соответствующий AuthenticationEntryPointможет быть установлен с помощью entry-point-refатрибута <http>элемента.

Пример приложения CAS - хороший пример использования настраиваемых bean-компонентов с пространством имен, включая этот синтаксис. Если вы не знакомы с точками входа для аутентификации, они обсуждаются в главе [технического обзора](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#tech-intro-auth-entry-point) .

4.4. Безопасность метода

Начиная с версии 2.0, Spring Security значительно улучшил поддержку для добавления безопасности к методам уровня service. Он обеспечивает поддержку безопасности аннотаций JSR-250, а также исходную аннотацию фреймворка @Secured. Начиная с версии 3.0 вы также можете использовать новые [аннотации на основе выражений](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#el-access) . Вы можете применить безопасность к одному bean-компоненту, используя элемент intercept-methods для украшения объявления bean-компонента, или вы можете защитить несколько bean-компонентов на всем уровне обслуживания, используя точечные вырезки в стиле AspectJ.

Элемент <global-method-security>

Этот элемент используется для включения безопасности на основе аннотаций в вашем приложении (путем установки соответствующих атрибутов в элементе), а также для группировки объявлений **Pointcut** безопасности(*смотри дальше*), которые будут применяться во всем контексте вашего приложения.

Вы должны объявить только один <global-method-security>элемент.

Следующее объявление включит поддержку Spring Security @Secured:

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

Добавление аннотации к методу (в классе или интерфейсе) соответственно ограничит доступ к этому методу. Встроенная поддержка аннотаций Spring Security определяет набор атрибутов для метода. Они будут переданы объекту, чтобы AccessDecisionManager принял 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 ​​позволяют применять простые ограничения на основе ролей, но не имеют собственных аннотаций Power Spring Security. Чтобы использовать новый синтаксис на основе выражений, вы должны использовать

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

и эквивалентный код Java будет

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 - хороший выбор, если вам нужно определить простые правила, которые выходят за рамки проверки имен ролей по списку полномочий пользователя.

|  |  |
| --- | --- |
|  | Аннотированные методы будут защищены только для экземпляров, которые определены как компоненты Spring (в том же контексте приложения, в котором включена безопасность метода). **Если вы хотите защитить экземпляры, которые не созданы Spring (например, с помощью newоператора), вам необходимо использовать AspectJ.** |
|  | **Вы можете включить более одного типа аннотации в одном приложении, но для любого интерфейса или класса следует использовать только один тип, иначе поведение не будет четко определено. Если обнаружены две аннотации, относящиеся к определенному методу, то будет применена только одна из них.** |

Добавление Security Pointcut с помощью protect-pointcut

Использование protect-pointcut является особенно мощным, поскольку позволяет применять безопасность ко многим компонентам с помощью 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 для bean-компонентов, объявленных в контексте приложения:

- классы которых находятся в пакете «com.mycompany»,

- имена классов заканчиваются на «Service»,

- только пользователи с ролью ROLE\_USER смогут вызывать эти методы.

Как и в случае сопоставления URL-адресов, наиболее конкретные совпадения должны быть первыми в списке точек, поскольку будет использоваться первое выражение сопоставления.

Аннотации безопасности имеют приоритет над pointcut.

4.5. Менеджер AccessDecisionManager по умолчанию (дополнительный функционал)

Этот раздел актуален только для людей, которым необходимо выполнить некоторую настройку, чтобы использовать больше, чем простую безопасность на основе ролей.

При использовании конфигурации пространства имен, для вас, по умолчанию, автоматически регистрируется экземпляр AccessDecisionManager, который будет использоваться для принятия решений относительно доступа к методам или вебу – URL. Это будет сделано на основе атрибутов доступа, которые вы указываете при объявлении <intercept-url>, <protect-pointcut> или в аннотациях (*если вы используя методы, защищенные аннотациями*).

Стратегия по умолчанию - использовать AffirmativeBased AccessDecisionManager вместе с RoleVoter и AuthenticatedVoter. Вы можете узнать больше об этом в главе об [авторизации](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#authz-arch) .

Настройка AccessDecisionManager

Если вам нужно использовать более сложную стратегию контроля доступа, легко установить альтернативу как для метода, так и для веб-безопасности.

Для безопасности метода, это можно сделать, установив access-decision-manager-refатрибут global-method-securityк idсоответствующей AccessDecisionManagerфасоли в контексте приложения:

<global-method-security access-decision-manager-ref="myAccessDecisionManagerBean">

...

</global-method-security>

Синтаксис для веб-безопасности такой же, но для элемента http:

<http access-decision-manager-ref="myAccessDecisionManagerBean">

...

</http>

4.6. Диспетчер аутентификации и пространство имен (дополнительный функционал)

Основным интерфейсом, который предоставляет услуги аутентификации в Spring Security, является расширение AuthenticationManager. Обычно это экземпляр класса Spring Security ProviderManager. Это будет рассмотрено позже, в [главе технического обзора](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#tech-intro-authentication) .

Экземпляр компонента регистрируется с использованием элемента authentication-manager пространства имен. Вы не можете использовать настраиваемый AuthenticationManager, если вы используете HTTP или безопасность метода через пространство имен, но это не должно быть проблемой, поскольку у вас есть полный контроль над используемыми AuthenticationProvider.

Вы можете зарегистрировать дополнительные bean-компоненты AuthenticationProvider с помощью ProviderManager, и вы можете сделать это с помощью элемента <authentication-provider> с атрибутом ref, где значение атрибута - это имя компонента-поставщика, который вы хотите добавить. Например:

<authentication-manager>

<authentication-provider ref="casAuthenticationProvider"/>

</authentication-manager>

<bean id="casAuthenticationProvider"

class="org.springframework.security.cas.authentication.CasAuthenticationProvider">

...

</bean>

Другое распространенное требование состоит в том, что другому компоненту в контексте может потребоваться ссылка на AuthenticationManager. Вы можете легко зарегистрировать псевдоним для AuthenticationManagerи использовать это имя где-нибудь в контексте вашего приложения.

<security:authentication-manager alias="authenticationManager">

...

</security:authentication-manager>

<bean id="customizedFormLoginFilter"

class="com.somecompany.security.web.CustomFormLoginFilter">

<property name="authenticationManager" ref="authenticationManager"/>

...

</bean>

5. Примеры приложений

В проекте доступно несколько примеров веб-приложений. Чтобы избежать чрезмерно больших загрузок, в zip-файл дистрибутива включены только образцы «учебника» и «контактов». Остальные могут быть созданы непосредственно из источника, который вы можете получить, как описано во [введении](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#get-source) . Самостоятельно собрать проект легко, а дополнительная информация доступна на веб-сайте проекта [http://spring.io/spring-security/](https://spring.io/spring-security/) . Все пути, упомянутые в этой главе, относятся к исходному каталогу проекта.

5.1. Учебный образец

Учебный пример - хороший базовый пример для начала. Он использует простую конфигурацию пространства имен.

Скомпилированное приложение включается в zip-файл дистрибутива и готово к развертыванию в вашем веб-контейнере ( spring-security-samples-tutorial-3.1.x.war). Механизм проверки подлинности на [основе форм](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#ns-form-and-basic) используется в сочетании с обычно используемым поставщиком проверки подлинности « [запомнить меня»](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#remember-me) для автоматического запоминания входа в систему с помощью файлов cookie.

Мы рекомендуем вам начать с учебного примера, поскольку XML минимален и прост в использовании. Что наиболее важно, вы можете легко добавить этот XML-файл (и соответствующие ему web.xmlзаписи) в существующее приложение. Только когда эта базовая интеграция будет достигнута, мы предлагаем вам попытаться добавить авторизацию метода или безопасность объекта домена.

5.2. Контакты

Пример контактов - это расширенный пример, поскольку он иллюстрирует более мощные функции списков управления доступом к объектам домена (ACL) в дополнение к базовой безопасности приложения. Приложение предоставляет интерфейс, с помощью которого пользователи могут управлять простой базой данных контактов (объектами домена).

Для развертывания просто скопируйте файл WAR из дистрибутива Spring Security в каталог webapps вашего контейнера . Следует вызвать spring-security-samples-contacts-3.1.x.war(добавленный номер версии будет зависеть от того, какой выпуск вы используете).

После запуска контейнера проверьте, может ли приложение загружаться. Посетите [http: // localhost: 8080 / contacts](http://localhost:8080/contacts) (или любой другой URL-адрес, подходящий для вашего веб-контейнера и развернутой WAR).

Затем нажмите «Отладка». Вам будет предложено пройти аутентификацию, и на этой странице будет предложен ряд имен пользователей и паролей. Просто выполните аутентификацию с помощью любого из них и просмотрите полученную страницу. Он должен содержать сообщение об успешном завершении, подобное следующему:

Информация об отладке безопасности

Security Debug Information

Authentication object is of type:

org.springframework.security.authentication.UsernamePasswordAuthenticationToken

Authentication object as a String:

org.springframework.security.authentication.UsernamePasswordAuthenticationToken@1f127853:

Principal: org.springframework.security.core.userdetails.User@b07ed00: Username: rod; \

Password: [PROTECTED]; Enabled: true; AccountNonExpired: true;

credentialsNonExpired: true; AccountNonLocked: true; \

Granted Authorities: ROLE\_SUPERVISOR, ROLE\_USER; \

Password: [PROTECTED]; Authenticated: true; \

Details: org.springframework.security.web.authentication.WebAuthenticationDetails@0: \

RemoteIpAddress: 127.0.0.1; SessionId: 8fkp8t83ohar; \

Granted Authorities: ROLE\_SUPERVISOR, ROLE\_USER

Authentication object holds the following granted authorities:

ROLE\_SUPERVISOR (getAuthority(): ROLE\_SUPERVISOR)

ROLE\_USER (getAuthority(): ROLE\_USER)

Success! Your web filters appear to be properly configured!

Успех! Ваши веб-фильтры настроены правильно!

После успешного получения указанного выше сообщения вернитесь на домашнюю страницу примера приложения и нажмите «Управление». Затем вы можете опробовать приложение. Обратите внимание, что отображаются только контакты, доступные для текущего пользователя, вошедшего в систему, и только пользователям с ROLE\_SUPERVISORдоступом предоставляется доступ для удаления своих контактов. За кулисами MethodSecurityInterceptorосуществляется охрана бизнес-объектов.

Приложение позволяет изменять списки контроля доступа, связанные с разными контактами. Обязательно попробуйте и поймите, как это работает, просмотрев XML-файлы контекста приложения.

5.3. Пример LDAP

Пример приложения LDAP предоставляет базовую конфигурацию и устанавливает как конфигурацию пространства имен, так и эквивалентную конфигурацию с использованием традиционных bean-компонентов, как в одном файле контекста приложения. Это означает, что на самом деле в этом приложении настроены два идентичных поставщика аутентификации.

5.4. Образец OpenID

Пример OpenID демонстрирует, как использовать пространство имен для настройки OpenID и как настроить конфигурации [обмена атрибутами](https://openid.net/specs/openid-attribute-exchange-1_0.html) для поставщиков удостоверений Google, Yahoo и MyOpenID (при желании вы можете поэкспериментировать с добавлением других). Он использует проект [openid-selector](https://code.google.com/p/openid-selector/) на основе JQuery для обеспечения удобной страницы входа в систему, которая позволяет пользователю легко выбрать поставщика, а не вводить полный идентификатор OpenID.

Приложение отличается от обычных сценариев аутентификации тем, что позволяет любому пользователю получить доступ к сайту (при условии успешной аутентификации OpenID). При первом входе в систему вы получите сообщение «Добро пожаловать [ваше имя]». Если вы выйдете из системы и снова войдете в систему (с тем же идентификатором OpenID), это должно измениться на «С возвращением». Это достигается с помощью custom, UserDetailsServiceкоторый назначает стандартную роль любому пользователю и сохраняет идентификаторы внутри карты. Очевидно, что реальное приложение будет использовать вместо этого базу данных. Посмотрите исходную форму для получения дополнительной информации. Этот класс также принимает во внимание тот факт, что различные атрибуты могут возвращаться от разных поставщиков и соответственно строить имя, под которым он обращается к пользователю.

5.5. Образец CAS

В примере CAS требуется, чтобы вы запускали и сервер CAS, и клиент CAS. Он не включен в дистрибутив, поэтому вам следует ознакомиться с кодом проекта, как описано во [введении](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#get-source) . Вы найдете соответствующие файлы в sample/casкаталоге. Там также есть Readme.txtфайл, который объясняет, как запускать и сервер, и клиент прямо из исходного дерева, в комплекте с поддержкой SSL.

5.6. Образец JAAS

Образец JAAS - это очень простой пример использования JAAS LoginModule с Spring Security. Предоставленный LoginModule успешно аутентифицирует пользователя, если имя пользователя совпадает с паролем, в противном случае создается исключение LoginException. AuthorityGranter, используемый в этом примере, всегда предоставляет роль ROLE\_USER. Пример приложения также демонстрирует, как запускаться как объект JAAS, возвращенный LoginModule, путем установки [jaas-api-provision](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#nsa-http-jaas-api-provision) равным «true».

5.7. Образец предварительной аутентификации

В этом примере приложения показано, как подключить компоненты из среды [предварительной проверки подлинности,](https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#preauth) чтобы использовать информацию для входа из контейнера Java EE. Имя пользователя и роли устанавливаются контейнером.

Код находится в формате samples/preauth.

6. Тесты чтобы убедиться, что наша конфигурация безопасности применяется правильно

@RunWith(SpringRunner.class)

@SpringBootTest(webEnvironment = RANDOM\_PORT)

public class BasicConfigurationIntegrationTest {

TestRestTemplate restTemplate;

URL base;

@LocalServerPort int port;

@Before

public void setUp() throws MalformedURLException {

restTemplate = new TestRestTemplate("user", "password");

base = new URL("http://localhost:" + port);

}

@Test

public void whenLoggedUserRequestsHomePage\_ThenSuccess() throws IllegalStateException, IOException {

ResponseEntity<String> response = restTemplate.getForEntity(base.toString(), String.class);

assertEquals(HttpStatus.OK, response.getStatusCode());

assertTrue(response.getBody().contains("Baeldung"));

}

@Test

public void whenUserWithWrongCredentials\_thenUnauthorizedPage() throws Exception {

restTemplate = new TestRestTemplate("user", "wrongpassword");

ResponseEntity<String> response = restTemplate.getForEntity(base.toString(), String.class);

assertEquals(HttpStatus.UNAUTHORIZED, response.getStatusCode());

assertTrue(response.getBody().contains("Unauthorized"));

}

}

Идея состоит в том, что за Spring Boot Security на самом деле стоит Spring Security, поэтому любая конфигурация безопасности, которая может быть выполнена с этим, или любая интеграция, которую он поддерживает, также может быть реализована в Spring Boot.