Внедрение OAuth 2.0 в существующие и новые приложения

Версия 1.0

Содержание

[1. Описание 4](#_Toc57637241)

[2. Внедрение 4](#_Toc57637242)

[2.1 DB 4](#_Toc57637243)

[2.1.1 Установка модуля OAuth 4](#_Toc57637244)

[2.1.2 Регистрация приложения 4](#_Toc57637245)

[2.2 APP 5](#_Toc57637246)

[2.2.1 JepRia GWT 5](#_Toc57637247)

[2.2.2 JepRia REST 6](#_Toc57637248)

[2.2.3 React 7](#_Toc57637249)

[2.3 Настройка окружения Apache HTTPD + Tomcat 8](#_Toc57637250)

[2.3.1 Источники данных 8](#_Toc57637251)

[2.3.2 Время жизни токенов 8](#_Toc57637252)

[2.3.3 Настройка балансировщика 8](#_Toc57637253)

[2.3.4 Apache Tomcat 9](#_Toc57637254)

[2.4 Управление сессиями 10](#_Toc57637255)

[2.5 Управление ключами доступа 10](#_Toc57637256)

Лист изменений

|  |  |  |  |
| --- | --- | --- | --- |
| **Дата** | **Версия** | **Описание** | **Автор** |
| 27.11.2020 | 1.0 | Создание документа | Савенков А.В. |

# Описание

[OAuth 2.0](http://tools.ietf.org/html/draft-ietf-oauth-v2) — протокол авторизации, позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя на другом сервисе. Протокол избавляет от необходимости доверять приложению логин и пароль, а также позволяет выдавать ограниченный набор прав, а не все сразу.

Общая схема работы приложения, использующего **OAuth**, такова:

* получение авторизации
* обращение к защищенным ресурсам

Результатом авторизации является *access token* — некий ключ (обычно просто набор символов), предъявление которого является пропуском к защищенным ресурсам. Обращение к ним в самом простом случае происходит по HTTPS с указанием в заголовках или в качестве одного из параметров полученного *access token*'а.

Все приложения, использующие **OAuth**, в обязательном порядке регистрируются в системе. Приложению присваивается *client\_id* и случайно сгенерированное секретное слово *client\_secret*. Эти значения позволяют уникально идентифицировать клиентское приложение.

Так же приложению задается *client\_name* и *client\_name\_en*, для двух соответствующих локалей.

Приложения могут одного из возможных поддерживаемых типов (*application\_type*).

* **web** – приложения, имеющие серверную часть.
* **browser** – приложения, исполняющие весь свой код в браузере пользователя (e.g. Single Page Application).
* **native** – десктопные и мобильные приложения.
* **service** – offline-сервисы, вызывающие защищенные API через технические учетные записи, а не от лица конкретного пользователя.

В протоколе описано несколько вариантов авторизации (**Grant types**), подходящих для различных ситуаций:

* **Authorization code** – авторизация для приложений, имеющих серверную часть. (чаще всего, это сайты и веб-приложения) (*grantTypes => authorization\_code*)
* **Authorization code PKCE** или **Implicit** – авторизация для полностью клиентских приложений. (мобильные, desktop-приложения, Single Page Applications) (*grantTypes => authorization\_code, implicit*)
* **Client credentials** – авторизация по *client\_id* и *client\_secret* зарегистрированного приложения. Если приложение использует этот тип авторизации, то для него создается оператор в AccessOperator’e, права доступа (роли) задаются через поле *scope* при регистрации приложения. Используется для авторизации по «технической учетной записи». (*grantTypes => client\_credentials*)
* **User credentials** – авторизация по логину и паролю. (*grantTypes => password*)
* **Refresh token** – восстановление предыдущей авторизации. (*grantTypes => refresh\_token*)

Более подробное описание типов авторизации и их алгоритмов, подробную информацию по API сервисов авторизации можно посмотреть в [техническом задании к модулю OAuth в SVN](svn://srvsvn/JEP/Module/OAuth/Trunk/Doc/ТЗ_OAuth_2.0.odt).

# Внедрение

## DB

### Установка модуля OAuth

Установка осуществляется согласно авто-документации к соответствующей версии ПО.

#### Основной источник данных

На основной источник данных (в случае production RFInfo схема information на RFINFOP) устанавливается основная версия DB-части и пакета OAuth из svn://srvsvn/JEP/Module/OAuth/Tag/x.x.x/DB где x.x.x последняя актуальная версия модуля.

#### Дополнительные источники данных

TBD

### Регистрация приложения

Все настройки системы осуществляются только через основной источник данных. Дополнительные источники, обеспечивающие отказоустойчивость системы, реплицируют данные из основного.

#### Скрипт

Предпочтительным способом регистрации клиентского приложения является прямая установка скриптом в базу. Установка выполняется в основной источник данных OAuth в рамках текущего окружения.

Регистрация осуществляется вызовом хранимой процедуры:

*pkg\_OAuth.createClient(*

*clientShortName => ?,*

*clientName => ?,*

*clientNameEn => ?,*

*applicationType => ?,*

*grantTypeList => ?,*

*roleShortNameList => ?,*

*loginModuleUri => ?,*

*operatorId => ?)*

* *clientShortName –* уникальное имя приложения (аналогично roleShortName в AccessOperator). Задается латиницей и цифрами.
* *clientName –* полное наименование приложения, отображается на странице авторизации при русской локали.
* *clientNameEn –* полное наименование приложения на английском, отображается во всех локалях кроме русской.
* *applicationType –* тип приложения.
* *grantTypeList* – список типов авторизаций через запятую.
* *roleShortNameList –* список ролей, присваиваемых приложению, актуален, только если *grantTypeList* содержит *client\_credentials.*
* *loginModuleUri – URL*логин модуля, если *null*, то аутентификация проходит через AccessOperator.
* *operatorId –* operator\_id выполняющего скрипт установки.

Если клиентское приложение имеет UI-часть, то нужно добавить URL(‘ы) переадресации. В стандарте OAuth клиентскому приложению соответствует white-list ссылок, на которые сервер может переадресовать пользователя после прохождения авторизации.

Для внутрибанковских приложений задаются относительные ссылки. (/Application/callback) Для внешних клиентских приложений абсолютные, допустимы только ссылки с защищенным соединением HTTPS. (https://server/Application/callback)

Стандартный формат URL’ов для GWT-приложений: “/ContextPath”.

Стандартный формат URL’ов для React-приложений (при использовании jfront-oauth): “/ContextPath/oauth”.

Формат ссылок неограничен и может быть изменен по необходимости.

Добавление ссылки выполняется вызовом хранимой процедуры:

*pkg\_OAuth.createClientUri(*

*clientShortName => ?,*

*clientUri => ?,*

*operatorId =>?)*

* *clientShortName –* уникальное имя приложения
* *clientUri –* добавляемый URL
* *operatorId –* operator\_id выполняющего скрипт установки.

Рекомендуется помечать в техническом задании, что модуль был переведен на OAuth. И какое значение было задано в *clientShortName.*Помечать, в map.xml зависимость от модуля OAuth.

#### Настройка через интерфейс

Все параметры приложения можно добавить/изменить через UI клиентских приложения.  
Он доступен в Навигации по пути: Администрирование => OAuth => Приложения.

## APP

### JepRia GWT

Для перехода с механизма SsoUi на OAuth требуется:

* Обновить версию jepria-gwt >12.1.0
* Добавить константу OAuth client\_id в web.xml приложения:

<context-param>

<param-name>oauthClientId</param-name>

<param-value>OAuthClient</param-value>

</context-param>

* заменить в web.xml:

<security-constraint>

<web-resource-collection>

<web-resource-name>Application entry points</web-resource-name>

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

</web-resource-collection>

<auth-constraint>

<role-name>\*</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>FORM</auth-method>

<form-login-config>

<form-login-page>/WEB-INF/security/Login.jsp</form-login-page>

<form-error-page>/WEB-INF/security/Login.jsp</form-error-page>

</form-login-config>

</login-config>

* На:

<filter>

<filter-name>SecurityFilter</filter-name>

<filter-class>com.technology.jep.jepria.server.security.servlet.oauth.OAuthEntrySecurityFilter</filter-class>

<init-param>

<param-name>security-constraint</param-name>

<param-value>\*</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>SecurityFilter</filter-name>

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

</filter-mapping>

* Для того чтобы добавить исключение из авторизации, нужно добавить еще одно объявление фильтра, но без init-param security-constraint:

<filter>

<filter-name>Public</filter-name>

<filter-class>com.technology.jep.jepria.server.security.servlet.oauth.OAuthEntrySecurityFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>Public</filter-name>

<url-pattern>/public/example</url-pattern>

</filter-mapping>

* Для подключения OAuth для сервисов на базе Java Servlet'ов рекомендуется использовать

com.technology.jep.jepria.server.security.servlet.oauth.OAuthServletSecurityFilter.

При регистрации приложений, использующих AutoLogonServlet, требуется включить “**password**” в список доступных типов авторизации приложения.

### JepRia REST

Для подключения OAuth в существующее приложение потребуется:

* Обновить версию jepria-rest > 12.2.0
* Добавить константу OAuth client\_id в web.xml приложения:

<context-param>

<param-name>oauthClientId</param-name>

<param-value>OAuthClient</param-value>

</context-param>

* Зарегистрировать OAuth фильтр в ApplicationConfig (только если не используется ApplicationConfigBase):

register(JepOAuthDynamicFeature.class);

* Добавить аннотацию @OAuth в нужные JaxRs адаптеры/методы:

@Path("/method")

@OAuth

public class JaxrsAdapter extends JaxrsAdapterBase {

@GET

@Path("/{id}")

@RolesAllowed("...")

@OAuth

public Response method(@PathParam("id") Integer id) {

...

}

}

Аннотация над классом закрывает все методы класса, над методом закрывает только этот метод.

* Если требуется поддержать Http Basic и OAuth одновременно, то следует использовать аннотацию @Protected:

@Path("/method")

@Protected

public class JaxrsAdapter extends JaxrsAdapterBase {

@GET

@Path("/{id}")

@RolesAllowed("...")

@Protected

public Response method(@PathParam("id") Integer id) {

...

}

}

При неудачной авторизации будет возвращен ответ с HTTP статусом 401.

### React

Поддержка OAuth для React SPA реализована в библиотеке jfront-oauth.

#### Установка

С помощью npm.

npm i @jfront/oauth-core --save-dev

npm i @jfront/oauth-context --save-dev

npm i @jfront/oauth-ui --save-dev

С помощью yarn.

yarn add @jfront/oauth-core --dev

yarn add @jfront/oauth-context --dev

yarn add @jfront/oauth-ui –dev

#### Пример использования

Пример простого использования в SPA. Использует [Authorization Code Grant with PKCE extension](https://tools.ietf.org/html/rfc7636) по стандарту. OAuthSecuredFragment позволяет закрывать отдельные части приложения, что дает возможность добавлять публичные интерфейсы при необходимости.

import React from "react"

import { OAuthWebContext, OAuthSecuredFragment } from "@jfront/oauth-ui"

function App() {

return (

<OAuthWebContext

clientId={"243j24h246j242gjikh6kllnn5l"} //client\_id приложения

redirectUri={"/app/oauth"} //Библиотека по умолчанию использует следующий формат URL’а scheme:[//authority]/context\_path/oauth

oauthContextPath={"/oauth/api"} //Ссылка на Context path сервисов oauth

>

<OAuthSecuredFragment>

<div>Secured view</div>

</OAuthSecuredFragment>

</OAuthWebContext>

)

}

## Настройка окружения Apache HTTPD + Tomcat

### Источники данных

Требует наличия настроенного источника данных "jdbc/OAuthDS".

При отсутствии вышеуказанного источника, по умолчанию, использует "jdbc/RFInfoDS".

Имя запасного источника данных можно настроить через переменную окружения "OAUTH\_BACKUP\_DATASOURCE".

### Время жизни токенов

Время жизни токенов настраивается через переменные окружения. Время жизни измеряется в часах.

#### Сессия SSO

**Настройка в context.xml Tomcat:**<Environment name="OAUTH\_SSO\_TOKEN\_LIFE\_TIME" value="24" type="java.lang.String" override="true"/>

Также можно настроить через переменную окружения ОС "OAUTH\_SSO\_TOKEN\_LIFE\_TIME".

Время жизни по умолчанию 24 часа.

#### Access Token

**Настройка в context.xml Tomcat:**<Environment name="OAUTH\_ACCESS\_TOKEN\_LIFE\_TIME" value="8" type="java.lang.String" override="true"/>

Также можно настроить через переменную окружения ОС "OAUTH\_ACCESS\_TOKEN\_LIFE\_TIME".

Время жизни по умолчанию 8 часов.

#### Refresh Token

**Настройка в context.xml Tomcat:**<Environment name="OAUTH\_REFRESH\_TOKEN\_LIFE\_TIME" value="24" type="java.lang.String" override="true"/>

Также можно настроить через переменную окружения ОС "OAUTH\_REFRESH\_TOKEN\_LIFE\_TIME".

Время жизни по умолчанию 24 часа.

### Настройка балансировщика

Для поддержки отказоустойчивости и горизонтального масштабирования рекомендуется включать балансировщик нагрузки.

#### Пример настройки Apache HTTPD

Пример стандартной настройки обработчиков в worker.properties:

worker.properties:

worker.list=navigation,worker1,worker2

worker.navigation.port=8011

worker.navigation.host=localhost

worker.navigation.type=ajp13

worker.worker1.port=8012

worker.worker1.host=localhost

worker.worker1.type=ajp13

worker.worker2.port=8013

worker.worker2.host=localhost

worker.worker2.type=ajp13

Для подключения OAuth нужно создать новый обработчик с типом load balancer:

worker.lb.type=lb

worker.lb.sticky\_session=True

Включить в него необходимые обработчики (все кроме инстанса с Navigation):

worker.lb.balance\_workers=worker1,worker2

Включить load balancer в общий список обработчиков:

worker.list= navigation,worker1,worker2,lb

Включить oauth в маршрутизацию mod\_jk:

mod\_jk.conf:

JkMount /oauth lb

JkMount /oauth/\* lb

Итоговая конфигурация:

worker.properties:

worker.list= navigation,worker1,worker2,lb

worker.lb.type=lb

worker.lb.balance\_workers=worker1,worker2

worker.lb.sticky\_session=True

worker.navigation.port=8011

worker.navigation.host=localhost

worker.navigation.type=ajp13

worker.worker1.port=8012

worker.worker1.host=localhost

worker.worker1.type=ajp13

worker.worker2.port=8013

worker.worker2.host=localhost

worker.worker2.type=ajp13

mod\_jk.conf:

JkMount /oauth lb

JkMount /oauth/\* lb

**Строго не рекомендуется** устанавливать OAuth и включать в Load Balancer инстанс Tomcat с установленным модулем Navigation. Т.к. это приведет к тому, что Sticky sessions будут приводить все запросы к OAuth на этот инстанс.

### Apache Tomcat

Требуется добавить jvmRoute для поддержки load balancer'a: <Engine name="Catalina" defaultHost="localhost" jvmRoute="worker1"/>

Рекомендуется включить <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>. Для поддержки привязанных к серверам сессий при отказе балансировщика.

#### Установка приложения OAuth

На каждый инстанс Tomcat, содержащий приложение, работающее через OAuth, нужно установить oauth.war из svn://srvsvn/JEP/Module/OAuth/Tag/x.x.x/App/dist, где x.x.x последний актуальный тэг.

Context Path установки должен быть “/oauth”. Каждый инстанс с установленными сервисами OAuth должен быть включен в балансировщик, как в пункте 2.3.3.1.

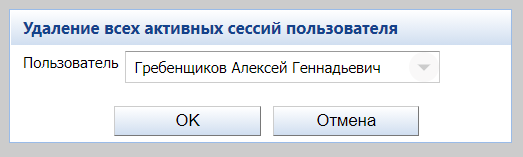
## Управление сессиями

Просмотреть активные сессии можно через интерфейс: Администрирование => OAuth => Сессии.

Интерфейс позволяют блокировать единичные как токены, так и все активные токены пользователя.  
Для удаления единичных записей используется стандартный механизм удаления выделенных на списке записей.  
Для удаления всех токенов пользователя нужно нажать на кнопку «Заблокировать все активные сессии»:



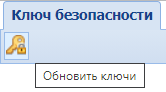
И в диалоговом окне выбрать нужного пользователя в выпадающем списке:

  
Подтверждение удаления токенов осуществляется нажатием кнопки ОК.

## Управление ключами доступа

Для подписи и шифрования токенов используются пары ассиметричных RSA ключей.  
Рекомендуемая частота обновления не реже, чем раз в три месяца. Автоматическое обновление происходит раз в три месяца.  
Обновить ключи можно через интерфейс: Администрирование => OAuth => Ключи безопасности.

Нажатие кнопки «Обновить ключи» сбросит текущую активную пару. **Важно,** сброс ключей выбрасывает всех пользователей из системы.



Также в этом интерфейсе отображается текущий публичный ключ. При необходимости дополнительной валидации токена доступа используется этот ключ. Передается разработчикам по запросу.