Шлюз управления доступом Access Gateway

Оглавление

Описание
Терминология
Устройство функциональности
  # Предикаты
  # Фильтры
  # Функциональность и настройки фильтров и предикатов
  # Особенности использования SpEl-фильтров
Эксплуатация
  # Конфигурация
  # Аудит

Описание

Access Gateway (шлюз управления доступом) — это инфраструктурный компонент современных ИТ-систем, предназначенный для централизованного управления доступом к прикладным сервисам и данным. Он выступает единой точкой входа для внешних и внутренних запросов, обеспечивая применение политик безопасности до передачи запроса целевым компонентам системы.

В архитектурном смысле шлюз доступа часто реализует паттерн PEP (Policy Enforcement Point), обеспечивая принудительное исполнение решений, принимаемых системой управления политиками доступа, и изолируя прикладные сервисы от прямого взаимодействия с механизмами безопасности. Кроме того, использование такого подхода позволяет реализовать расширенные функции авторизации в ситуации, когда само приложение не реализует требуемой функциональности.

В состав RooX UIDM входит компонент, реализующий данную функциональность. Функциональность используется как для управления доступом в рамках продукта, так и для управления доступом к приложениям заказчика.

Шлюз в составе RooX UIDM позволяет не только выступать в роли фильтра входящих запросов, но также может выполнять дополнительные операции: переадресацию пользователя, преобразование входящего запроса, преобразование ответа, обмен токена (token exchange) и т.д.

Документ описывает возможности данного компонента, особенности работы, его настройку и эксплуатацию.

Терминология

Gateway, Шлюз

> шлюз управления доступом, выполняющий маршрутизацию, авторизацию и вспомогательную обработку запросов;

Spring Cloud Gateway

> реактивный фреймворк в экосистеме Spring для создания API-шлюза в микросервисных архитектурах;

SSO

> Single Sign-On — система единого входа реализованная в RooX UIDM через компонент sso-server;

sso-server

> компонент в составе RooX UIDM, который выполняет функции IdP, AM, SSO и другие

AAL

> Authentication / Authorization Library — библиотека в составе RooX UIDM SDK;

aud (Audience)

> логическое имя защищаемого ресурса;

SpEL

> Spring Expression Language — язык выражений Spring Framework

Устройство функциональности

В RooX UIDM шлюз управление доступом реализован в компоненте `api-gateway-wtl` . Компонент реализован на базе Spring Cloud Gateway. `api-gateway-wtl` интегрирован с другими компонентами в составе RooX UIDM, а также реализует все стандартные для RooX UIDM функциональные возможности, необходимые для эксплуатации решения, такие как поддержка сквозного логирования и сбор метрик для мониторинга.

Компонент работает со следующими логическими сущностями:

Маршруты (routes)

> основная сущность, определяет наборы правил для обработки запросов и ответов.

Предикаты (predicates)

> задают правила по выбору маршрута, который будет выбран для обработки запроса

Фильтры (filters)

> выполняет пре- и пост- обработку запроса и ответа, может прерывать выполнение запроса, формировать и видоизменять ответ

`api-gateway-wtl` расширяет функциональность предоставляемую фреймворком, предоставляя набор дополнительных предикатов и фильтров, подробнее описанных ниже.

Предикаты

Стандартные предикаты Spring Cloud Gateway

`After`

> проверка текущей даты, что она находится после заданного момента времени

* `Before`

> проверка текущей даты, что она находится перед заданным моментом времени

* `Between`

> проверка текущей даты, что она находится в заданном интервале времени

* `Cookie`

> проверка наличия куки со значением удовлетворяющего заданному RegEx выражению

* `Header`

> проверка наличия заголовка со значением удовлетворяющего заданному RegEx выражению

* `Host`

> проверка заголовка Host запроса

* `Method`

> проверка метода HTTP-запроса

* `Path`

> проверка пути в URI HTTP-запроса

* `Query`

> проверка параметров HTTP-запроса

* `RemoteAddr`

> проверка IP-адреса клиента вызвавшего запрос

* `Weight`

> балансирование нагрузки

Предикаты RooX UIDM

CheckBodyForSpel

> Выполняет проверку тела запроса на соответствие заданному SpEl

Фильтры

Стандартные фильтры Spring Cloud Gateway

`AddRequestHeader`

> Добавляет заголовок в исходящий HTTP-запрос.

`AddRequestParameter`

> Добавляет параметр в строку запроса.

`AddResponseHeader`

> Добавляет заголовок в ответ.

`CircuitBreaker`

> Реализует шаблон Circuit Breaker с возможностью fallback.

`FallbackHeaders`

> Добавляет заголовки с информацией об ошибке при переходе на fallback.

`Hystrix`

> Оборачивает маршрут в команду Hystrix с поддержкой fallback.

`PrefixPath`

> Добавляет префикс к пути запроса.

`PreserveHostHeader`

> Сохраняет оригинальный заголовок Host при пересылке запроса.

`RedirectTo`

> Выполняет перенаправление на указанный URL с заданным статусом.

`RemoveRequestHeader`

> Удаляет указанный заголовок из запроса.

`RemoveResponseHeader`

> Удаляет указанный заголовок из ответа.

`RemoveHopByHopHeadersFilter`

> Удаляет hop-by-hop заголовки согласно IETF.

`RequestRateLimiter`

> Ограничивает частоту запросов на основе ключа (например, IP).

`RequestSize`

> Ограничивает максимальный размер тела запроса.

`Retry`

> Повторяет запрос при ошибках с заданными условиями.

`RewritePath`

> Переписывает путь запроса с использованием регулярных выражений.

`RewriteResponseHeader`

> Изменяет значения заголовков ответа с помощью регулярных выражений.

`SaveSession`

> Принудительно сохраняет сессию перед пересылкой запроса.

`SecureHeaders`

> Добавляет стандартные заголовки безопасности в ответ.

`SetPath`

> Устанавливает новый путь запроса с использованием шаблонов.

`SetResponseHeader`

> Устанавливает или заменяет заголовок в ответе.

`SetStatus`

> Устанавливает HTTP-статус ответа.

`StripPrefix`

> Удаляет заданное количество сегментов из начала пути запроса.

Фильтры RooX UIDM

`OAuth2Security`

> (старое название — TokenValidation) выполнение авторизации запросов посредством валидации токена доступа.

`PolicyEvaluation`

> выполнение расчета политики и принятие решения на предоставление доступа.

`TokenExchange`

> обмен исходного токена доступа на токен, полученный через механизм UIDM Token Exchange с возможностью расширения полномочия субъекта.

`TokenSupplier`

> (старое название — BearerTokenSupplier) простановка токена в заголовок Authorization по схеме Bearer или в параметры тела запроса. Поддерживает различные источники токена.

`MaskByJsonPath`

> маскирование чувствительных данных в ответе.

`MaskPhoneNumber`

> маскирование номера телефона в ответе.

`FormatPhone`

> Форматирование номера телефона в ответе сервиса.

`HeaderToBodyReplacer`

> Перекидывание данных из заголовка ответа сервера в тело ответа клиенту.

`WhiteListJsonAttribute`

> Фильтрация JSON тела ответа по белому списку атрибутов.

`SystemAuth`

> Получение системного токен и подстановка в запрос в заголовок `Authorization` по схеме `Bearer`.

`RemoveJsonAttributes`

> Удаляет из ответа json-поля с заданным именем.

`SpelDecision`

> Локальное вычисление политики — принимает решение о предоставлении доступа на основании вычисляемого SpEl-выражения.

`SetRequestHeaderWithSpel`

> Устанавливает заголовок запроса, значение которого вычисляется через SpEl-выражение.

Функциональность и настройки фильтров и предикатов

В данном разделе собрано подробное описание логики работы и настроек фильтров и предикатов RooX UIDM.

Настройки маршрутов `api-gateway-wtl` задаются в формате `yaml` и содержат, для каждого маршрута: * `id` — уникальный идентификатор машрута * `uri` — URI назначения, куда проксируется запрос * `predicates` — набор предикатов * `filters` — набор фильтров

Шлюз выбирает маршрут, если все его предикаты были вычислены как `true`. Для выбранного маршрута выполняются все фильтры над полученным запросом, запрос проксируется на указанный URI, после чего выполняется набор фильтров над полученным от сервера ответом. Результирующий ответ передается клиенту.

Подробнее смотри документацию на Sping Cloud Gateway.

Специфичные для данного маршрута параметры предикатов и фильтров задаются в yaml-файле. Кроме того, некоторые фильтры и предикаты используют настройки, заданные стандартным для RooX UIDM способом в properties-файлах.

Общие для компонента в целом особенности настройки приведены в разделе Конфигурация.

Предикат CheckBodyForSpel

Назначение: проверка тела запроса на соответствие заданному SpEl

Конфигурационные аргументы:

`inClass` - полное имя класса, в который преобразуется тело запроса перед тем как прогнать его через spel

`spelExpression` - SpEl выражение, которое должно вернуть значение типа Boolean

Пример конфигурации:

```
- id: idm-users-write
  uri: http://localhost:8498/identity-storage-1.0/scim/v2/Users
  predicates:
    - Path=/identity-storage-1.0/scim/v2/Users,/identity-storage-1.0/scim/v2/Users/*
    - Method=POST,PUT,PATCH
    - name: CheckBodyForSpel
      args:
        inClass: java.util.LinkedHashMap
        spelExpression:
'@{data.get("urn:ietf:params:scim:schemas:extension:roox:case:2.0:$Shared") == null}'
```

Данный предикат отлавливает запросы, которые пришли от клиента, анализирует тело и принимает запрос если в теле отсутствует поле `urn:ietf:params:scim:schemas:extension:roox:case:2.0:$Shared`.

Основные сценарии использования

Для использования api-gateway-wtl в мультидоменной конфигурации рекомендуется для каждого маршрута указывать предикат Host, чтобы исключить слишком широкого срабатывания предикатов.

Маршруты с более узкими правилами в предикатах следует располагать выше чем маршруты с более широкими предикатами.

Пример:

```
routes:
  - id: staff_employee
    uri: http://127.0.0.1:8998
    predicates:
      - Host=staff.example.com
      - Path=/employee/**
      - Method=GET
  - id: staff_management_reports
    uri: http://127.0.0.1:8998
    predicates:
      - Host=staff.example.com
      - Path=/management/reports/**
      - Method=GET
  - id: staff_static
    uri: http://127.0.0.1:8998
    predicates:
      - Host=staff.example.com
      - Path=/static/**
      - Method=GET
  - id: staff
    uri: http://127.0.0.1:8998
    predicates:
      - Host=staff.example.com
      - Path=/**
    filters:
      - name: OAuth2Security
        args:
          aud: staff
          on-fail: authorize
          redirect-response-headers:
            Cache-Control: "no-cache, no-store, must-revalidate"
            Pragma: "no-cache"
            Expires: "0"
```

В примере представлены маршруты для приложения staff, которое размещено на домене staff.example.com. При этом приложение защищено аутентификацией через OAuth2Security фильтр. Но для ряда GET-запросов сделаны исключения (в целях снизить нагрузку), таким образом через аутентификацию не будут прогоняться запросы к урлам `/employee/**`, `/management/reports/**`, `/static/**`. Здесь `**` — это широкий wildcard, который позволяет применить данный придикат к любому количеству сегментов в пути запроса (по аналогии с тем как работает AntPattern в Spring Security).

Фильтр `Path=/**` в маршрут `staff` добавлен для наглядности и по сути не обязателен, так как его шаблон применим к любому запросу. Но если api-gateway-wtl будет работать в одно-доменной среде и фильтр `Host` будет отсутствовать, то для реализации "дефолтного" маршрута его нужно будет добавить, так как не допустимо иметь маршруты без предикатов.

Также три маршрута `staff_employee`, `staff_management_reports` и `staff_static` можно объединить в один, так как предикат `Path` принимает список в значения аргумента `patterns`:

```
routes:
  - id: staff_bypass
    uri: http://127.0.0.1:8998
    predicates:
      - Host=staff.example.com
      - Path=/employee/**,/management/reports/**,/static/**
      - Method=GET
```

Фильтр OAuth2Security (старое название — TokenValidation)

Назначение: выполнение авторизации запросов через валидацию токенов средствами AAL (Authentication/Authorization Library) из RooX UIDM SDK

Дополнительные возможности:

- обмен кода авторизации на токен доступа с выставлением кук,
- в случае отсутствия токена доступа или истечении срока его действия может выполняться обновление токена доступа через OAuth2 Refresh Token Flow
  - из конфига `com.rooxteam.aal.{aud}.sso.token.refresh.cookie.name` вычитывается имя куки с рефреш-токеном (если конфиг не задан — пропустить процедуру обновления)
  - вычитывается из запроса кука с рефреш токеном (если кука отсутствует — пропустить процедуру обновления)
  - выполняется обновление куки доступа через OAuth2 Refresh Token Flow
  - в случае успешного обновления новые значения токенов доступа и рефреша выставляются в куки (см. Конфигурация TokenCookieManager)
- можно задать действие в случае неуспешной авторизации запроса:
  - возврат ответа со статусом ошибки 401 Unauthorized (по-умолчанию)
  - редирект на страницу аутентификации
  - редирект на страницу ошибки

Конфигурационные аргументы:

`aud` — audience (логическое имя) прикрываемого ресурса (см. подробнее в ▸)

`on-fail` — действие при неуспешной валидации токена

`redirect-response-headers` — список заголовков добавляемых к ответу в случае выполнения редиректа

Пример конфигурации:

```
filters:
  - name: OAuth2Security
    args:
      aud: arm_cc
      on-fail: authorize
      redirect-response-headers:
        Cache-Control: "no-cache, no-store, must-revalidate"
        Pragma: "no-cache"
        Expires: "0"
```

Используются базовые конфиги AAL, которые могут использоваться в aud-based варианте. Наиболее востребованные для фильтра:

```
com.rooxteam.aal.{aud}.sso.endpoint=http://demo.uidm.ru:15018/sso
com.rooxteam.aal.{aud}.sso.token.cookie.name=at

# задает источник данных принципала (используется при валидации токена доступа)
# - JWT - офлайн валидация токена доступа по OIDC
# - TOKENINFO (по-умолчанию) - онлайн валидация токена доступа через запрос на OAuth2 TokenInfo Endpoint в RooX UIDM SSO Server
# - USERINFO - онлайн валидация токена доступа через запрос на OIDC Userinfo Endpoint
com.rooxteam.aal.{aud}.filter.principal_provider_type=JWT

# список валидаторов JWT токена при использовании principal_provider_type=JWT
com.rooxteam.aal.{aud}.jwt.validators=com.rooxteam.sso.aal.validation.jwt.impl.RsSignatureValidator,com.rooxteam.sso.aal.validation.jwt.impl.AudValidator

# URL к JKWS-эндпоинту с ключами для валидации JWT-токена
# ВНИМАНИЕ! текущая версия SDK не умеет самостоятельно высчитывать JWKS урл, поэтому для мультирилмовых окружений потребуется настройка под каждый aud использующий офлайн-валидацию токена
com.rooxteam.aal.{aud}.jwks.url=http://demo.uidm.ru:15018/sso/realms/customer/as/oauth2/idp/jwks.json
```

также добавлены дополнительные ключи конфига для реализации функции фильтра (они как бы тоже AAL-ые, но реализованы только в api-gateway-wtl):

```
# URL эндпоинта получения токена OAuth2
com.rooxteam.aal.{aud}.oauth2.token_endpoint=https://demo.uidm.ru/sso/realms/{realm}/as/oauth2/idp/access_token

# URL эндпоинта авторизации OAuth2
com.rooxteam.aal.{aud}.oauth2.authorization_endpoint=https://demo.uidm.ru/sso/realms/{realm}/as/oauth2/idp/authorize

# Значение параметра response_type для запроса на авторизацию OAuth2 (deprecated с 25.1)
com.rooxteam.aal.{aud}.oauth2.authorization_response_type=code mpt

# URL OAuth2-консьюмера RooX UIDM (deprecated с 25.2)
com.rooxteam.aal.{aud}.oauth2.oauth2_consumer_endpoint=https://demo.uidm.ru/oauth2-consumer/authorize

# URL страницы перенаправления в случае ошибки
com.rooxteam.aal.{aud}.oauth2.error_page=https://demo.uidm.ru/uidm-admin-panel/error

# имя куки из которой получают refresh токен, если не задано - то обновление токена доступа не выполняется
com.rooxteam.aal.{aud}.sso.token.refresh.cookie.name=reft

# имя параметра в котором передан код авторизации (для каких-нибудь странных случаев, когда IDP вставляет в кастомный параметр)
com.rooxteam.aal.{aud}.sso.authorization-code.param=code
```

Варианты ответа при неуспешной аутентификации (управляется аргументом `on-fail` фильтра).

`error` — сервис отвечает со статусом 401 (Unauthorized)

`redirect` — сервис отвечает со статусом 302 — редирект на страницу ошибки

страница ошибки указывается конфигом `com.rooxteam.aal.{aud}.oauth2.error_page`

`authorize` — сервис отвечает со статусом 302 — редирект на страницу аутентификации

URL в этом случае формируется следующим образом:

1. берет URL эндпоинта авторизации из конфига `com.rooxteam.aal.{aud}.oauth2.authorization_endpoint`
2. добавляет следующие параметры:
   a. `client_id` — заполняется из конфига `com.rooxteam.aal.{aud}.auth.client`
   b. `realm` — заполняется из конфига `com.rooxteam.aal.{aud}.auth.realm`
   c. `response_type` — заполняется из конфига `com.rooxteam.aal.{aud}.oauth2.authorization_response_type` (по-умолчанию: `code`)
   d. `redirect_uri` — формирует следующим образом:
      i. если задан конфиг с урлом консьюмера (`com.rooxteam.aal.{aud}.oauth2.oauth2_consumer_endpoint`) - то добавляет ему параметр state сформированный из следующих значений (в виде URL-энкоженной строки `key1=value1&key2=value2`):
         A. `client_id` — заполняется из конфига `com.rooxteam.aal.{aud}.auth.client`
         B. `realm` — заполняется из конфига `com.rooxteam.aal.{aud}.auth.realm`
         C. goto — текущий URL запроса
         D. gotoOnFail — значение из конфига `com.rooxteam.aal.{aud}.oauth2.error_page`
      ii. если конфига с консьюмером нет — то просто текущий урл запроса

Замена oauth2-consumer

Данный фильтр может заменить использование компонента `oauth2-consumer` (исключен из состава RooX UIDM, начиная с релиза 25.2) для реализации методов `authorize` и `refresh`.

Пример конфигурации маршрута для замены `oauth2-consumer/authorize`:

```
- id: uidm-selfservice_auth
  predicates:
    - Path=/uidm-selfservice/auth
  filters:
    - name: OAuth2Security
      args:
        aud: self_service
        on-fail: authorize
    - RedirectTo=302, https://uidm.ru/uidm-selfservice
```

Пример конфигурации маршрута для замены `oauth2-consumer/refresh`

```
- id: uidm-selfservice_refresh
  predicates:
```

```
    - Path=/uidm-self-service/refresh
  filters:
    - name: OAuth2Security
      args:
        aud: self_service
        on-fail: error
    - name: SetStatus
      args:
        status: 200
```

В случае успешного прохождения фильтра OAuth2Security выполнится редирект на URL целевого сервиса или возврат пустого ответа с кодом `200`

Атрибуты Exchange устанавливаемые фильтром

После успешного прохождения фильтра выставляются следующие атрибуты в текущий Exchange:

- `com.rooxteam.gateway.oauth2Security.aud` — aud текущего маршрута
- `com.rooxteam.gateway.oauth2Security.realm` — рилм соответствующий aud`у текущего запроса
- `com.rooxteam.gateway.principal` — модель принципала полученная при валидации токена

Фильтр PolicyEvaluation

Назначение: выполнение расчета политики и принятие решения на предоставление доступа.

ВАЖНО

На вычисление политик отправляется токен провалидированный фильтром OAuth2Security, поэтому в маршруте обязательно должен присутствовать фильтр OAuth2Security перед фильтром PolicyEvaluation.

Пример конфигурации:

```
filters:
  - name: OAuth2Security
    args:
      aud: audience
  - name: PolicyEvaluation
    args:
      aud: audience
      resource: /api/user
      action: GET
      env:
        param1: value1
        param2: value2
```

Конфигурационные аргументы:

`aud` — audience, используется для выбора соответствующей конфигурации,

`resource` — имя ресурса для которого выполняется расчет политики,

`action` — действие выполняемое при обращении к прикрываемому ресурсу, для которого выполняется расчет политики,

`env` — дополнительные атрибуты принимающие участие в расчете политики.

Настройки окружения

Настройки окружения которые управляют поведением фильтра можно задать через конфиг с использованием `aud`. См. AUD-based конфигурация AAL.

```
# базовый URL SSO-Server
com.rooxteam.aal.{aud}.sso.endpoint=http://demo.uidm.ru:15018/sso

# тип авторизации
# - JWT (по-умолчанию) - через запрос к Policy Evaluation Endpoint в sso-server
# - OPA (deprecated) - через запрос к Open Policy Agent (OPA)
com.rooxteam.aal.{aud}.authorization_type=JWT
```

Атрибуты Exchange устанавливаемые фильтром

После успешного прохождения фильтра выставляются следующие атрибуты в текущий Exchange:

- `com.rooxteam.gateway.policy_evaluation_positive=true`

Фильтр TokenExchange

Назначение: обмен исходного токена доступа на токен, полученный через механизм RooX UIDM Token Exchange (/token) с возможностью расширения полномочия субъекта.

Выполняется запрос к RooX UIDM SSO Server по протоколу OAuth 2.0 Token Exchange, согласно RFC 8693.

Пример конфигурации:

```
filters:
  - name: TokenExchange
    args:
      aud: audience
      scope: telephoneNumber cid cn
      roles: ROLE_PROVISION
```

Конфигурационные аргументы:

`aud` — audience, используется для выбора соответствующей конфигурации, `scope` — запрашиваемые области доступа (scope), `roles` — запрашиваемые роли

`roles`, `aud`, `scope` — значения будут проставлены в новый токен

Настройки для формирования запроса в Token Exchange (clientId и client secret) можно задать через конфиг с использованием `aud`

```
com.rooxteam.aal.{aud}.sso.endpoint=http://demo.uidm.ru:15018/sso
com.rooxteam.aal.{aud}.auth.client=test_oauth2m2m
com.rooxteam.aal.{aud}.auth.password=password
```

При этом если значение aud-based конфига не установлено, то используется значение конфига без `aud`, таким образом можно задавать общие AAL-конфиги и оверрайдить их конфигом по `aud`.

В случае некорректной конфигурации фильтра (а также зависимых настроек AAL) и получения ошибки в ходе выполнения запроса на обмен токена, фильтр прерывает работу и возвращает ответ со статусом 502 Bad Gateway.

Фильтр TokenSupplier (старое название — BearerTokenSupplier)

Назначение: простановка токена в заголовок Authorization по схеме Bearer или в параметры тела запроса. Поддерживает различные источники токена через аргумент `provider`

Конфигурационные аргументы:

- `provider` — указывает источник токена. Варианты значения:
  - `principal` — при выполнении после фильтра OAuth2Security проставляет токен авторизованный для данного запроса;
  - `cookie` — без валидации использует токен из авторизационной куки для заданного `aud`.
- `supplier` — указывает куда будет подставлен токен. Варианты значения:
  - `bearer` (по-умолчанию) — токен подставляется в заголовок `Authorization` по схеме `Bearer`;
  - `x_www_form_urlencoded_param` — токен подставляется в тело запроса как параметр по схеме `x-www-form-urlencoded`
- `token-param` — имя параметра для подстановки токена в тело запроса по схеме `x-www-form-urlencoded` опционально, по-умолчанию: `access_token`
- `aud` — audience (логическое имя) прикрываемого ресурса, используется для определения имени куки, из которой нужно достать токен

Пример конфигурации — вариант 1:

```
filters:
  - name: OAuth2Security
    args:
      aud: arm_cc
  - name: TokenSupplier
    args:
      provider: principal
```

Пример конфигурации — вариант 2:

```
filters:
  - name: TokenSupplier
    args:
      provider: cookie
      aud: arm_cc
      supplier: x_www_form_urlencoded_param
```

Конфиги AAL в aud-based варианте:

```
com.rooxteam.aal.{aud}.sso.token.cookie.name=at
```

Замена oauth2-consumer

Данный фильтр может заменить использование компонента `oauth2-consumer` для реализации запроса `tokens/@current`

Пример конфигурации маршрута для замены `oauth2-consumer/tokens/@current`:

```
- id: tokens-current
  uri: http://demo.uidm.ru:15018/sso/oauth2/tokeninfo
  predicates:
    - Path=/uidm-adminpanel-api/tokens/@current
    - Method=POST
  filters:
    - SetPath=/sso/oauth2/tokeninfo
    - name: TokenSupplier
      args:
        aud: arm_cc
        provider: cookie
        supplier: x_www_form_urlencoded_param
    - name: WhiteListJsonAttribute
      args:
        allowed: accessToken, amr, "com.rooxteam.sso.auth.delegate.authMethod",
contactEmail, displayName, expires_in, roles, sub, telephoneNumber, user_name, deviceId,
executionId
```

Фильтр MaskByJsonPath

Назначение: маскирование чувствительных данных в ответе

Конфигурационные аргументы:

- `jsonPaths` — список JSON-путе к маскируемым полям
- `mask` — значение маски на которую заменяется значение

  Опционально. По-умолчанию: `******`

Пример конфигурации:

```
filters:
  - name: MaskByJsonPath
    args:
      jsonPaths: $..auth.clientId, $..auth.accessToken
      mask: '******'
```

Заменяет в данных ответа значения полей auth.clientId и auth.accessToken на звездочки

Фильтр MaskPhoneNumber

Назначение: маскирование номера телефона в ответе.

Фильтр похож на MaskByJsonPath, но более узкоспециализированный, рассчитанный на маскирование номеров телефонов. Позволяет выполнять замену через RegEx. При этом его можно задействовать и для других данных, например email, или ФИО.

Конфигурационные аргументы:

- `jsonPath` — JSON-путь к маскируемому полю

- `search` — RegEx-шаблон поиска значения для замены;

  Опционально. По-умолчанию: `\?\d+(\d{2})(\d{2})+`
- `replacement` — значение для подстановки результатов обработки RegEx шаблона

  Опционально. По-умолчанию: `**$1-$2`

Пример конфигурации — вариант 1:

```
filters:
  - name: MaskPhoneNumber
    args:
      jsonPath: $..phoneNumbers[*].value
```

Заменяет в данных ответа IdM User номер телефона с `79012345678` на `**56-78`

Пример конфигурации — вариант 2:

```
filters:
  - name: MaskPhoneNumber
    args:
      jsonPath: $..phoneNumbers[*].value
      search: \+?\d(\d{3})\d+(\d{2})
      replacement: +7 ($1) ***-**-$2
```

Заменяет в данных ответа IdM User номер телефона с `79012345678` на `+7 (901) *--78`

Фильтр FormatPhone

Назначение: Форматирование номера телефона в ответе сервиса. Замена `FormatPhoneRewriteFunction` из api-gateway-arm

Конфигурационные аргументы:

- `fields` — RegEx-шаблон для имени полей, которые нужно форматировать

  опционально, по-умолчанию: `address|phone`
- `search` — RegEx-шаблон для форматирования

  опционально, по-умолчанию: `7?(.{3})(.{3})(.{2})(.{2})`
- `replacement` — шаблон подстановки

  опционально, по-умолчанию: `+7 ($1) $2-$3-$4`

Пример конфигурации (для замены фильтра из api-gateway-arm конфигурация не требуется, дефолты соответствуют)

```
- name: FormatPhone
  args:
    fields: address|phone
    search: 7?(.{3})(.{3})(.{2})(.{2})
    replacement: +7 ($1) $2-$3-$4
```

Фильтр HeaderToBodyReplacer

Назначение: Перекидывание данных из заголовка ответа бек-сервиса в тело ответа. Полезный как замена `LocationHeaderReplacerRewriteFunction` из api-gateway-arm

Конфигурационные аргументы:

- `headerName` — имя заголовка источника данных

  опционально, по-умолчанию: `Location`
- `fieldName` — имя поля в теле ответа, куда складываются полученные данные

  опционально, по-умолчанию: `id`
- `pattern` — RegEx шаблон извлечения данных из заголовка

  опционально, по-умолчанию: `/principals/v2/by_id/([a-zA-Z0-9\\-._~]+)`
- `statusCode` — Код статуса ответа

  опционально, по-умолчанию: `201`

Пример конфигурации для замены фильтра из api-gateway-arm (дефолты составлены для замены `LocationHeaderReplacerRewriteFunction`):

```
filters:
  - name: HeaderToBodyReplacer
    args:
      headerName: Location
      fieldName: id
      pattern: /principals/v2/by_id/([a-zA-Z0-9\\-._~]+)
      statusCode: 201
```

Фильтр WhiteListJsonAttribute

Назначение: Фильтрация JSON тела ответа по белому списку атрибутов.

Поля могут указываться с учетом иерархии, с разделителем через точку.

Если обернуть значение поля в кавычки, то оно считается единым и не разделяется.

Например:

- `auth.authMethod` разрешает поле `auth` и вложенное в него поле `authMethod`
- `"auth.authMethod"` разрешает поле `auth.authMethod` в корне объекта
- `auth."authMethod.origin"` разрешает поле `auth` и вложенное в него поле `authMethod.origin`

Ограничения:

- не умеет работать с объектами внутри списков

Конфигурационные аргументы:

- `allowed` — перечень разрешенных атрибутов с учетом иерархии

Замена oauth2-consumer

Данный фильтр может заменить использование компонента oauth2-consumer для реализации запроса `tokens/@current`

Пример конфигурации маршрута для замены `oauth2-consumer/tokens/@current`:

```
- id: tokens-current
  uri: http://demo.uidm.ru:15018/sso/oauth2/tokeninfo
  predicates:
    - Path=/uidm-adminpanel-api/tokens/@current
    - Method=POST
  filters:
    - SetPath=/sso/oauth2/tokeninfo
    - name: TokenSupplier
      args:
        aud: arm_cc
        provider: cookie
        supplier: x_www_form_urlencoded_param
    - name: WhiteListJsonAttribute
      args:
        allowed: accessToken, amr, "com.rooxteam.sso.auth.delegate.authMethod",
contactEmail, displayName, expires_in, roles, sub, telephoneNumber, user_name, deviceId,
executionId
```

Фильтр SystemAuth

Назначение: Получить системный токен и подставить в заголовок `Authorization` по схеме `Bearer`.

Конфигурационные аргументы:

`aud` — имя рилма в котором выполняется системная аутентификация,

Пример конфигурации:

```
- name: SystemAuth
  args:
    aud: aud
```

конфигурация `realm`, `clientId`, `clientSecret` задается под `aud`:

```
com.rooxteam.aal.{aud}.auth.realm=employee
com.rooxteam.aal.{aud}.auth.client=test_client1
com.rooxteam.aal.{aud}.auth.password=password
```

В случае некорректной конфигурации фильтра (а также зависимых настроек AAL) и получения ошибки в ходе выполнения запроса на получение системного токена, фильтр прерывает работу и возвращает ответ со статусом 502 Bad Gateway.

Фильтр RemoveJsonAttributes

Назначение: Удаляет из ответа поля с заданным именем.

Конфигурационные аргументы:

- `fieldList` — список имен полей, которые должны быть удалены из ответа
- `deleteRecursively` — если `true` - то будет рекурсивно перебирать поля в JSON-теле ответа и удалять те значения которых указаны в поле `fieldList`

опционально. По-умолчанию: `true`

Пример конфигурации:

```
filters:
  - name: RemoveJsonAttributes
    args:
      fieldList: auth
      deleteRecursively: true
```

Фильтр SpelDecision

Назначение: Локальное вычисление политики — принимает решение о предоставлении доступа на основании вычисляемого SpEl-выражения.

Конфигурационные аргументы:

- `expression` — вычисляемое SpEl выражение, которое должно вернуть `true` или `false`.

SpEl выражения обрамляются в конструкцию: `@{` и `}`

Пример конфигурации:

```
- SpelDecision=@{exchange.attributes["com.rooxteam.gateway.principal"].properties["roles"].contains("ROLE_ADMIN")}
```

Фильтр SetRequestHeaderWithSpel

Назначение: Устанавливает заголовок запроса (к прикрываемому сервису), значение которого вычисляется через SpEl.

Конфигурационные аргументы:

- `name` — имя заголовка
- `value` — значение заголовка, можно использовать SpEl-выражения,

SpEl выражения обрамляются в конструкцию: `@{` и `}`

Пример конфигурации:

```
- name: SetRequestHeaderWithSpel
  args:
    name: X-Principal
    value: '@{exchange.attributes["com.rooxteam.gateway.principal"].properties["sub"]}'
```

Примечание: значение SpEl в этом случае нужно заключать в кавычки, иначе падает парсинг YAML-файла — ему не нравится когда скаляр начинается со спец-символа. Можно обрамлять двойными кавычками, но тогда внутри атрибуты должны быть в одинарных.

Если же @ будет в середине выражения - то кавычки уже не требуются. По этой причине краткая форма записи в одну строку не требует обрамления кавычками:

```
- SetRequestHeaderWithSpel=X-Principal,
@{exchange.attributes["com.rooxteam.gateway.principal"].properties["sub"]}
```

Особенности использования SpEl-фильтров

Для расчета SpEl могут использоваться атрибуты текущей модели обмена, установленные SpringCloudGateway или другими фильтрами. Так фильтр OAuth2Security в случае успешного прохождения устанавливает следующие атрибуты:

- `com.rooxteam.gateway.oauth2Security.aud` — aud текущего маршрута
- `com.rooxteam.gateway.oauth2Security.realm` — рилм соответствующий aud`у текущего запроса
- `com.rooxteam.gateway.principal` — модель принципала полученная при валидации токена

Фильтр PolicyEvaluation в случае успешной авторизации через политики устанавливает следующий атрибут:

- `com.rooxteam.gateway.policy_evaluation_positive=true`

Базовая модель принципала

ВАЖНО

Наполнение модели принципала может отличаться в зависимости от способа валидации (JWT или tokeninfo)

Пример базовой модели (валидация через tokeninfo):

```
{
  "properties": {
    "sub": "sso_____8faebec2-7441-4402-abbb-2dd572fff7d6",
    "telephoneNumber": "+79988877036", // scope telephoneNumber
    "auth_level": "0",
    "contactEmail": "test036@roox.ru",  // scope contactEmail
    "user_name": "test036", // scope user_name
    "sn": "Пушкин", // scope sn
    "givenname,": "Александр", // scope sn
    "displayName,": "Александр Сергеевич Пушкин", // scope displayName
    "amr": [
      "pwd",
      "sms"
    ],
    "roles": [
      "ROLE_CUSTOMER"
    ],
    "ext_sub": "8faebec2-7441-4402-abbb-2dd572fff7d6",
    "cn": "+79988877036",  // scope cn
    "token_type": "Bearer",
    "prn": "sso_____8faebec2-7441-4402-abbb-2dd572fff7d6",
    "client_id": "selfservice_e_m2m",
    "executionId": "d664a2a4-1dd0-4701-8c0f-000ed6dde0fb",
    "aud": [
      "selfservice_e_m2m"
    ],
    "scope": [
      "telephoneNumber",
      "contactEmail",
```

```
      "user_name",
      "displayName",
      "amr",
      "roles",
      "fullName",
      "cn",
      "givenname",
      "organizations",
      "middleName",
      "sn",
      "authType",
      "contactPhone"
    ],
    "auth_time": 1766584664,
    "organizations": [], // scope organizations
    "realm": "employee",
    "authType": "login_password",  // scope authType
    "contactPhone": "+79988877036",  // scope contactPhone
    "expires_in": 3566,
    "authLevel": [
      "0"
    ],
    "jti": "1fc43b1d-bd42-427d-bdc5-2d87305fa7d8"
  },
  "expirationTime": 1766784360587,
  "privateJwtToken": "1fc43b1d-bd42-427d-bdc5-2d87305fa7d8",
  "jwtToken": "1fc43b1d-bd42-427d-bdc5-2d87305fa7d8",
  "anonymous": false
}
```


Комментариями указаны области доступа (scope) необходимые для получения данного атрибута. Конкретное имя скоупа и его значение могут быть настроены в sso-server/ Другие данные устанавливаемые в токен через scope так же будут доступны в модели.

Базовые атрибуты запроса доступные через объект exchange

Для вычисления в SpEl можно также использовать различные атрибуты запроса. Например:

- `exchange.request.queryParams` — список параметров запроса,
- `exchange.request.headers` — список заголовков запроса,
- `exchange.request.cookies` — список кук.

Примеры использования:

Проверка роли:

```
exchange.attributes['com.rooxteam.gateway.principal'].properties['roles'].contains('ROLE_CUSTOMER')
```

Проверка прохождения второго фактора или аутентификации через Kerberos:

```
exchange.attributes['com.rooxteam.gateway.principal'].properties['amr'].contains('sms') || exchange.attributes['com.rooxteam.gateway.principal'].properties['amr'].contains('wia')
```

Проверка что в параметре userId передан идентификатор принципала из токена доступа:

```
exchange.request.queryParams['userId'][0] ==
exchange.attributes['com.rooxteam.gateway.principal'].properties['sub']
```

Также, используя функцию шлюза, при которой предикаты `Host` и `Path` позволяют извлекать параметры из URL запроса, можно формировать следующие конфигурации:

```
  - id: adminpanel-api-users
    uri: http://127.0.0.1:8998
    method: get
    predicates:
      - Host={tenant}.uidm.ru
      - Path=/uidm-adminpanel-api/users/{userId}
    filters:
      - name: OAuth2Security
        args:
          aud: audience
      - name: SpelDecision
        args:
          expression: >
exchange.attributes['org.springframework.cloud.gateway.support.ServerWebExchangeUtils.uriT
emplateVariables']['userId']
            ==
            exchange.attributes['com.rooxteam.gateway.principal'].properties['sub']
      - SetRequestHeaderWithSpel=x-tenant,
@{exchange.attributes['org.springframework.cloud.gateway.support.ServerWebExchangeUtils.ur
iTemplateVariables']['tenant']}
```

Параметры, заданные выражениями в предикатах вида `{имя параметра}`, сохраняются в атрибут `org.springframework.cloud.gateway.support.ServerWebExchangeUtils.uriTemplateVariables` и далее могут использоваться в SpEl-фильтрах. В примере один из них используется для принятия решения о предоставлении доступа, а второй для установки в заголовок запроса.

Использование дополнительных объектов приложения

В SpEl-выражениях можно использовать объекты, доступные в контексте приложения компонента. Можно обращаться к свойствам объектов или вызывать их методы. Технически такие объекты представляют из себя JavaBeans.

Для получения дополнительной информации, в том числе о других доступных объектах, обратитесь в техническую поддержку RooX.

Объект конфигурации `rooxConfig`

Предназначен для чтения параметров, заданных в конфигурации приложения (`.properties`-файлы и другие поддерживаемые источники).

Пример SpEL-выражения:

```
@rooxConfig.getString("com.rooxteam.gw.tenant", "default")
```

Пример использования в фильтре `SetRequestHeaderWithSpel`

```
filters:
  - SetRequestHeaderWithSpel=x-principal, tenant-
@{@rooxConfig.getString("com.rooxteam.gw.x-principal-prefix", "roox")}-
@{exchange.attributes['org.springframework.cloud.gateway.support.ServerWebExchangeUtils.ur
iTemplateVariables']['tenant']}
```

Эксплуатация

Конфигурация

Настройка шлюза доступа состоит из двух частей

- **Параметры компонента и фильтров**. Содержит различные параметры как компонента в целом, так и отдельных фильтров. Данные параметры задаются стандартным для RooX UIDM способами настройки.
- **Настройка маршрутов**. Содержит перечень маршрутов с указанием предикатов и фильтров.

В следующих разделах указаны особенности настройки маршрутов и ряда функциональных возможностей шлюза. Подробное описание настроек фильтров приведено в разделе Функциональность и настройки фильтров и предикатов.

Настройки маршрутов

Данные настройки задаются в формате yaml-файла, формат и структура которого унаследованы от Spring Cloud Gateway.

Пример настроек:

```
spring:
  cloud:
    gateway:
      routes:
        - id: add_request_header_route
          uri: https://example.org
          predicates:
          - Path=/red/{segment}
          filters:
          - AddRequestHeader=X-Request-Red, Blue-{segment}
```

По умолчанию компонент `api-gateway-wtl` читает файл `routes/routes.yaml` из директории, в которой установлен сервис. Путь к файлу может быть переопределен через конфигурационный ключ `spring.config.additional-location`.

Загрузка маршрутов из нескольких конфигурационных файлов

Использование единого файла для всех маршрутов может быть неудобно при эксплуатации при большом количестве независимых друг от друга маршрутов, например, определяющих доступ к нескольким разным

приложениям. Поэтому в RooX UIDM реализована дополнительная возможность: использование конфигурационного ключа `com.rooxteam.gateway.routes`, в котором перечисляется список URL к дополнительным файлам с маршрутами. Это позволяет разбить множество маршрутов на насколько групп и использовать отдельный файл для каждой группы.

ВАЖНО

Основной файл с маршрутами (`routes/routes.yaml`) должен существовать в любом случае. Шлюз будет читать его для получения общих настроек, например настройки SSL. Маршруты из основного файла будут объединены с маршрутами из дополнительных файлов.

ВАЖНО

В файлах, указанных в ключе `com.rooxteam.gateway.routes` необходимо указывать маршруты, начиная от секции `routes`.

Например, приведенный выше пример преобразуется к виду:

```
routes:
  - id: add_request_header_route
    uri: https://example.org
    predicates:
    - Path=/red/{segment}
    filters:
    - AddRequestHeader=X-Request-Red, Blue-{segment}
```

AUD-based конфигурация AAL

Шлюз управления доступом предназначен для управления доступа к различным ресурсам через валидацию токена и проверку политик в RooX UIDM. Операции по валидации токена и проверки политик реализованы в библиотеке `AAL` (Authentication/Authorization Library), которая входит в состав RooX UIDM SDK.

AAL предназначена для выполнения запросов на валидацию токенов и политик от имени некоторого приложения. Необходимые для работы библиотеки параметры приложения задаются стандартными настройками RooX UIDM. Так, идентификатор приложения задается в настройке `com.rooxteam.aal.auth.client`.

Шлюз управления доступом же в свою очередь предназначен для защиты **любых** приложений, при этом предполагается использование одного экземпляра гейтвея для нескольких приложений. Каждое приложение может иметь свой идентификатор client_id в системе RooX UIDM (sso-server). Для реализации этой функции в api-gateway-wtl конфигурация библиотеки AAL расширена и позволяет задавать разные параметры библиотеки в зависимости от параметра `aud` (от audience).

Принцип действия следующий:

Для объекта (фильтр, предикат, др.), которому требуется использование возможностей AAL для определенного приложения (clientId), в параметрах маршрута добавляется аргумент `aud`. А в стандартных настройках RooX UIDM задается необходимый набор настроек AAL, в ключ которых добавлено соответствующее значение `aud` следующим образом:

```
com.rooxteam.aal.{config-path}
        V V V
com.rooxteam.aal.{aud}.{config-path}
```

При этом, если значение aud-based конфига не установлено, то используется значение конфига без `aud`, таким образом можно задавать общие настройки AAL и переопределять значения для отдельных значений `aud`.

Конфигурация TokenCookieManager

Фильтр OAuth2Security умеет выполнять обновление токена доступа через OAuth2 Refresh Token Flow (RFC 6749). Полученные в результате токены будут установлены как куки в ответе на запрос. Для выполнения этой операции используется функциональность `TokenCookieManager`.

Конфигурация `TokenCookieManager` является clientId-based, то есть зависит от clientId, который фильтр получает из параметра `com.rooxteam.aal.{aud}.auth.client`. Обычно (но не обязательно) его значение совпадает со значением `aud`.

`TokenCookieManager` позволяет управлять настройками для нескольких сущностей, которые должны сохраняться в куки. В настройках сущность задается в ключе параметра. Плейсхолдер `{entity}` может принимать следующие значения:

- `access` — для куки с токеном доступа (OAuth2 access_token),
- `refresh` — для куки с токеном обновления (OAuth2 refresh_token),
- `pkce` — для куки с PKCE verifier.

```
# Имя куки для сущности {entity}
# по-умолчанию: {entity}=access -> `at`
#               {entity}=refresh -> `reft`
#               {entity}=pkce -> `pcv`
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.name

# Является ли кука сессионной
# по-умолчанию: {entity}=access -> false
#               {entity}=refresh -> false
#               {entity}=pkce -> true
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.is_session

# Срок жизни куки в секундах (выставляется только если is_session=false для куки)
# по-умолчанию: 600
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.age.max

# Значение атрибута Domain для выставляемых кук
# по-умолчанию: значение отсутствует
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.domain

# Значение атрибута Path для выставляемых кук
# по-умолчанию: значение отсутствует
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.path

# Добавлять или нет атрибут HttpOnly для выставляемых кук
# по-умолчанию: true
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.httponly

# Добавлять или нет атрибут Secured для выставляемых кук
# по-умолчанию: true
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.secured

# Значение атрибута SameSite для выставляемых кук
# по-умолчанию: Lax
com.rooxteam.gw.oauth2.{entity}.cookie.{clientId}.samesite
```

Политика поиска настроек:

Если в конфиге параметр для указанных значений `{entity}` и `{clientId}` отсутствует, то сначала проверяется конфиг только с указанным `{entity}`, без сегмента `{clientId}`, и только при его отсутствии принимается значение по-умолчанию. Исключение из правила — параметры, задающие имя куки и сессионность, для которых значения по умолчанию определены уже на уровне `{entity}`.

Настройка SSL и TLS

Шлюз управления доступом может выполнять терминирование SSL/TLS для входящих соединений и использовать SSL/TLS для исходящих соединений.

Реализация HTTPS предоставляется функционалом Spring Framework и Spring Cloud Gateway.

Конфигурация выполняется в файле с маршрутами

Для входящих соединений нужно задать параметры кейстора и алиас ключа:

```
server:
  ssl:
    enabled: true
    key-alias: scg
    key-store-password: scg1234
    key-store: classpath:scg-keystore.p12
    key-store-type: PKCS12
```

Для исходящих соединений необходимо перечислить доверенные сертификаты:

```
spring:
  cloud:
    gateway:
      httpclient:
        ssl:
                #useInsecureTrustManager: true -- доверяем всем сертификатам - не для прода
          trustedX509Certificates:
          - cert1.pem
          - cert2.pem
          # параметры TLS-хэндшейка
          handshake-timeout-millis: 10000
          close-notify-flush-timeout-millis: 3000
          close-notify-read-timeout-millis: 0
```

Для тестовых окружений или на этапе отладки взаимодействия с защищаемыми приложениями допускается использование настройку `useInsecureTrustManager: true`, которая отключает валидацию предъвляемых приложениями сертиифатов.

Аудит

Шлюз управления доступом может записывать события аудита о предоставлении доступа к ресурсу, через асинхронный механизм.

ВАЖНО

Режим прямой записи событий аудита в СУБД в шлюзе доступа не поддержан.

События

`gw.access_control.protected_resource.grant.success`

> доступ успешно предоставлен

`gw.access_control.protected_resource.grant.fail`

> отказано в предоставлении доступа

Конфигурация записи аудита

Функционал является отключаемым, по умолчанию события записываются.

```
# позволяет отключить запись событий аудита гейтвеем
# по-умолчанию - `true` - аудит пишется
com.rooxteam.gw.audit.enabled=false
```

Способ записи событий аудита может быть задан следующей настройкой:

```
# Указывает реализации бинов для записи событий аудита:
#  - amqpOperationAuditDao - асихнронный аудит через очередь
#  - loggingOperationAuditDao (по-умолчанию и если список пуст) - логирующий аудит (пишет в лог)
com.rooxteam.gateway.configuration.auditWriterImpl=amqpOperationAuditDao,loggingOperationAuditDao
```

Остальные необходимые настройки описаны в документе Потоковая обработка событий аудита.