ИТ-ПРОФИКС

Описание функциональных характеристик программного обеспечения «МАРС. Система маршрутизации и протоколирования»

Екатеринбург, 2023

Оглавление

Термины и определения

Термин	Определение
ИС	Информационная система
Платформа	МАРС. Система маршрутизации и протоколирования
JMS	стандарт программного обеспечения для рассылки сообщений, позволяющий приложениям, выполненным на платформе Java EE создавать, посылать, получать и читать сообщения
JSON	(англ. *JavaScript Object Notation*, обычно) — текстовый формат обмена данными, основанный на JavaScript
REST	(от англ. Representational State Transfer - передача состояния представления) архитектурный стиль взаимодействия компонентов распределенного приложения в сети
URL	Адрес (ссылка), указывающий точное местоположение веб-ресурса
XML	eXtensible Markup Language — расширяемый язык разметки — рекомендованный Консорциумом всемирной сети (W3C) язык разметки, представляющий собой свод общих синтаксических правил. XML — текстовый формат, предназначенный для хранения структурированных данных, для обмена информацией между программами.
Бизнес-процесс (БП)	Процесс, описывающий прохождение сообщения и его преобразований внутри Платформы от отправителя до конечного получателя
ActiveMQ	Брокер сообщений с открытым исходным кодом, написанный на Java, вместе с полноценным клиентом Java Message Servise (JMS).
Kafka	Распределенный программный брокер сообщений с открытым исходным кодом, написанный на языках Java и Scala, позволяет обрабатывать потоковые данные в реальном времени с высокой скоростью и низкой задержкой

Назначение системы

МАРС. Система маршрутизации и протоколирования (далее по тексту Платформа) – интеграционная Платформа, позволяющая осуществлять маршрутизацию сообщений любого формата от Инициатора обмена к Потребителю вне зависимости от их содержимого.

Сообщения транслируются из одной очереди брокера ActiveMQ (AMQ) в другую очередь брокера AMQ (в том числе очередь другого брокера) на основе мета-информации, передаваемой в заголовках сообщений и данных настроек маршрутов прохождения («Бизнес-процессов»). Так же Платформа имеет возможность взаимодействовать с топиками брокера Apache Kafka, транслировать сообщения в топик и читать сообщения из топиков, с последующей их маршрутизацией в очереди брокера ActiveMQ.

Доставка сообщений от Инициатора к Получателю информации осуществляется в асинхронном режиме.

Маршруты прохождения сообщений между компонентами интегрируемых систем могут быть многоуровневыми, разветвляться, включать в себя запросы на получение информации от интегрируемых систем и получение ответов. Правила маршрутизации сообщений настраиваются в таблице базы данных и объединяются в единый Бизнес-процесс с уникальным идентификатором.

В процессе передачи формат сообщений может быть трансформирован через компонент Atlasmap из формата JSON в формат XML и обратно, исходя из правил, которые настраиваются заранее в таблице базы данных Платформы и соответствуют Бизнес-процессу, определенному по составу мета-данных заголовков маршрутизируемого сообщения.

Скорость прохождения сообщений по маршрутам регулируется отдельным компонентом Платформы THROTTLE.

Дополнительно Платформа позволяет генерировать JWT-токены для потребителей – пары access и refresh токенов.

Платформа включает в себя следующие программные компоненты:

- **bpRouter** – сервис идентификации Бизнес-Процессов (БП);
- **ATLASMAP-COMMON** – сервис трансформации форматов сообщений JSON в XML и наоброт;
- **AUTH-COMMON** – сервис генерации JWT-токенов;
- **BROKER-SWITH** – сервис передачи сообщений между брокерами;
- **Store** – сервис-планировщик отправки сообщений;
- **THROTTLE** – сервис-планировщик траффика сообщений;
- **STOCK.FILTER** – сервис дополнительной обработки маршрутизации (фильтрация сообщений);
- **Common-Routers** – осуществляет передачу сообщения в указанную очередь брокера;
- **BRIDGE** – сервис передачи сообщений между брокерами.

Функциональные возможности

Основной функциональной возможностью Платформы является доставка сообщения от одной интегрируемой Информационной системы в другую в асинхронном режиме через очереди брокера ActiveMQ.

Дополнительный функционал Платформы:

- Трансформация формата сообщения из XML в JSON и обратно согласно ранее настроенного маппинга полей;
- Доставка сообщения в очередь другого брокера AMQ;
- Доставка сообщения в топик брокера Apache Kafka, чтение сообщения из топика брокера Apache Kafka;
- Генерация и доставка JWT-токенов;
- Планирование времени доставки сообщение;
- Управление траффиком;
- Фильтрация сообщений согласно установленным правилам;

Для демонстрации работы Платформы настроен тестовый маршрут доставки сообщения от Информационной системы №1 (условное обозначение - IS_1) в Информационную систему №2 (условное обозначение -IS_2). С этой целью в состав дистрибутива Платформы включены программные сущности - Адаптеры информационных систем ADP.IS_1 и ADP.IS_2, которые не являются компонентами Платформы и предназначены исключительно для того, чтобы продемонстрировать работоспособность Платформы.

Проверку работоспособности системы можно провести с использованием преднастроенных примеров через web-интерфейс, подробности прохождения сообщений по маршрутам находятся в логах системы. Порядок обращения к логам Платформы приведен в «Инструкции по установке экземпляра ПО «МАРС. Система маршрутизации и протоколирования».

Примеры для проверки каждого из компонента приведены в соответствующих главах настоящего документа.

Компоненты системы.

Настройки маршрутизации (описание «Бизнес-процесса»)

Маршрут следования сообщения может быть сложным и многоуровневым, разветвляться и содержать несколько этапов.

Настройки маршрута хранятся в таблице базы данных Платформы Business_processes. Описание структуры таблицы приведено в Приложении №1.

На уровне записей в таблице business_processes маршрут разделяется на этапы (Trans_Hop), каждому из этапов соответствует одна запись в таблице и одно логическое действие – переместить сообщение из текущего места в определенную очередь. Объединяет этапы маршрута в целое уникальный идентификатор бизнес-процесса BP_ID.

В описании маршрута могут быть даны указания на проведение дополнительной обработки сообщения перед отправкой по указанному адресу.

Например, в маршруте может быть заложено указание осуществить трансформацию формата сообщения из XML в JSON – в параметре Key1 указано имя файла трансформации.

Так же в маршрут могут быть заложены логические условия, согласно которым сообщение отправляется на тот или иной этап маршрута (ветвление).

Например, при условии содержания в заголовке HRoute_Hop_Key1 = 2 сообщение будет переведено на этап 2. Если в заголовке HRoute_Hop_Key1 будет 0 или ничего не указано, то сообщение будет отправлено на следующий по порядку этап.

Параметры таблицы маршрутизации:

Поле	**Описание**
BP_ID	Уникальный идентификатор Бизнес-процесса. Перечень Бизнес-процессов настраивается в справочнике bp_descr (см. Приложение 1)
SYS_ID	Идентификатор программной сущности (микросервиса/адаптера), которая делает запрос на маршрутизацию сообщения. Допустимые значения настраиваются в справочнике SYS_DESCR (см. Приложение 1)
ROUTE_HOP	Номер этапа прохождения маршрута. Используется для корректного позиционирования сервиса bpRouter в настроечной таблице при прохождении многоуровневых и разветвляющихся маршрутов.
ORGANIZATON	Идентификатор организации, для которой настраивается маршрут
JMS_TYPE	Значение соответствует заголовку сообщения, которое должно быть маршрутизировано по настраиваемому маршруту
SENDER	Идентификатор информационной системы, инициирующей отправку сообщения в начале маршрутизации
RECEIVER	Идентификатор информационной системы, в которую должно быть доставлено сообщение после прохождения всего маршрута
WORK_FLAG	Индикатор состояния этапа маршрута – 1 этап работает, 0 – этап не работает.
DESCR	Текстовое описание этапа маршрута
KEY1	Произвольное значение для дополнительной обработки сообщения
KEY2	Произвольное значение для дополнительной обработки сообщения
KEY3	Произвольное значение для дополнительной обработки сообщения
TRANS_HOP	Номер этапа прохождения внутри маршрута. Описание этапов настраивается в таблице TRANS_CODE_DESCR (см. Приложение 1)
ROUTE_TO	Наименование очереди брокера, в которую должно быть маршрутизировано сообщение на данном этапе
FAULT_ROUTE_TO	Наименование очереди, в которую будет отправлено сообщение в том случае, если возникнут ошибки в результате идентификации бизнес-процесса или невозможности корректной маршрутизации
FAULT_HOP	Номер этапа для отправки некорректного сообщения

VERSION	Версия маршрута
WRITE_BODY	Записывать тело сообщения при формировании логов прохождения сообщения. 1 – записывать, 0 – не записывать
PRIORITY	Приоритет обработки этапа маршрута
ROUTE_HOPE_KEY	Предназначен для ветвления маршрута сообщения. При наличии в таблице нескольких одинаковых строк описания этапа и заголовка обрабатываемого сообщения HRoute_Hope_key, сервис bp_router выберет строку описания соответствующую значению заголовка в таблице
PERSISTEN	Свойство гарантированной доставки сообщения

Пример настройки Бизнес-процесса, в котором сообщение с JMSType TEST.MESSAGE.RQ в формате JSON из адаптера Информационной системы 1 (ADP.IS_1), который обслуживается брокером AMQ BROKER1, должно быть доставлено в Адаптер информационной системы 2 (ADP.IS_2), который обслуживается брокером AMQ BROKER2, или топик kafka test.topic.rq Ветвление маршрута осуществляется на уровне адаптера ADP.IS_1 за счет параметров, передаваемых в дополнительном заголовке сообщения. При доставке в адаптер ADP.IS_2 сообщение должно быть трансформировано из формата JSON в XML.

Информационная система №2 получив сообщение формирует ответ TEST.MESSAGE.RS, который отправляет или через адаптер ADP.IS_2 и очереди AMQ, либо выкладывает в топик kafka test.topic.rs. При получении ответа из ADP.IS_2 он должен быть трансформирован из формата XML в формат JSON. Обмен через топики кафка осуществляется в формате JSON.

BP_ID	SYS_ID	ROUTE_HOP	ORGANIZATON	JMS_TYPE	SENDER	RECEIVER	WORK_FLAG	DESCR	KEY1
ROUTE.IS_1.IS_2	ADP.IS_1	0	TEST	TEST.MESSAGE.RQ	IS_1	IS_2	1	Запрос	Map1.json
ROUTE.IS_1.IS_2	SRV.ATLAMAP-COMMON	1	TEST	TEST.MESSAGE.RQ	IS_1	IS_2	1	Запрос	
ROUTE.IS_1.IS_2	SRV.BROKER-SWITH	2	TEST	TEST.MESSAGE.RQ:FROM_BROKER:BROKER1	IS_1	IS_2	1	Запрос	
ROUTE.IS_1.IS_2	ADP.IS_2	3	TEST		IS_2	IS_1	1	Ответ	
ROUTE.IS_1.IS_2	SRV.BROKER-SWITH	4	TEST		IS_2	IS_1	1	Ответ	
ROUTE.IS_1.IS_2	SRV.ATLAMAP-COMMON	5	TEST	TEST.MESSAGE.RS	IS_2	IS_1	1	Ответ	Map1.json
ROUTE.IS_1.IS_2	ADP.IS_1	0	TEST	TEST.MESSAGE.RQ	IS_1	IS_2	1	Запрос	
ROUTE.IS_1.IS_2	SRV.BRIDGE	1	TEST	TEST.MESSAGE.RQ	IS_1	IS_2	1	Запрос	
ROUTE.IS_1.IS_2	SRV.BRIDGE	2	TEST	TEST.MESSAGE.RS	IS_2	IS_1	1	Ответ	

KEY2	KEY3	TRANS_HOP	ROUTE_TO	FAULT_ROUTE_TO	FAULT_HOP	VERSION	WRITE_BODY	PRIORITY	ROUTE_HOPE_KEY	PERSISTEN
JSON		0	SRV.ATLASMAP.COMMON.IN	FAULT.QUEUE	0	1	1	1	0	1
		1	SRV.BROKER.SWITCH.IN	FAULT.QUEUE	0	1	1	1	0	1
		2	BROKER2:ADP.IS_2.IN	BROKER2:FAULT.QUEUE	0	1	1	1	0	1
		3	SRV.BROKER.SWITCH.IN	FAULT.QUEUE	0	1	1	1	0	1
		4	BROKER1:SRV.ATLASMAP.COMMON.IN	BROKER1:FAULT.QUEUE	0	1	1	1		
XML		5	ADP.IS_1.IN	FAULT.QUEUE	0	1	1	1	0	1
		0	SRV.BRIDGE.IN	FAULT.QUEUE	0	1	1	1	1	1
		1	Kafka:test.topic.rq	FAULT.QUEUE	0	1	1	1	1	1
		2	ADP.IS_1.IN	FAULT.QUEUE	0	1	1	1	1	1

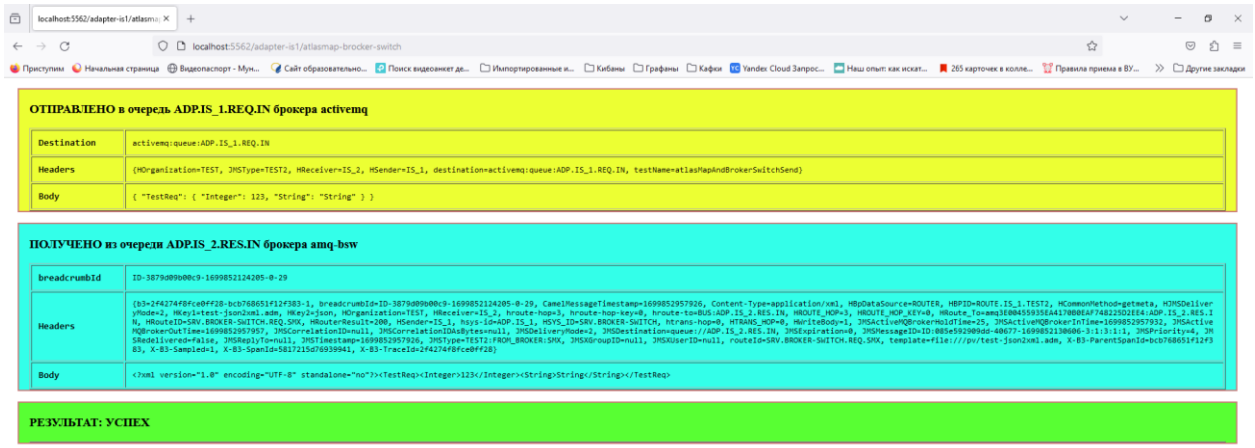
Примеры:

Пример 1. Доставка сообщения по основному маршруту.

Демонстрация примера по доставке сообщения от адаптера системы ADP.SYS_1 в адаптер ADP.IS_2:

Открыть в web-браузере страницу http://localhost:5562/adapter-is1/atlasmap-brocker-switch

В результате выполнения сценария сообщение будет доставлено из одной системы в другую. Подробная информация о прохождении по маршруту будет добавлена в логи Платформы.



ОТПРАВЛЕНО в очередь ADP.IS_1.REQ.IN брокера activemq

Destination	activemq:queue:ADP.IS_1.REQ.IN
Headers	{HOrganization=TEST, JMSType=TEST2, HReceiver=IS_2, HSender=IS_1, destination=activemq:queue:ADP.IS_1.REQ.IN, testName=atlasMapAndBrokerSwitchSend}
Body	{ "TestReq": { "Integer": 123, "String": "String" } }

ПОЛУЧЕНО из очереди ADP.IS_2.RES.IN брокера amq-bsw

breadcrumbId	ID-3879d09b00c9-1699852124205-0-29
Headers	{b3=2f4274f8fce0ff28-bcb768651f12f383-1, breadcrumbId=ID-3879d09b00c9-1699852124205-0-29, CamelMessageTimestamp=1699852957926, Content-Type=application/xml, HBpDataSource=ROUTER, HBPID=ROUTE.IS_1.TEST2, HCommonMethod=getmeta, HJMSDeliveryMode=2, HKey1=test-json2xml.adm, HKey2=json, HOrganization=TEST, HReceiver=IS_2, hroute-hop=3, hroute-hop-key=0, hroute-to=BUS:ADP.IS_2.RES.IN, HROUTE_HOP=3, HROUTE_HOP_KEY=0, HRoute_To=amq3E00455935E4417080EAF748225D2EE4:ADP.IS_2.RES.IN, HRouteID=SRV.BROKER-SWITCH.REQ.SMX, HRouterResult=200, HSender=IS_1, hsys-id=ADP.IS_1, HSYS_ID=SRV.BROKER-SWITCH, htrans-hop=0, HTRANS_HOP=0, HWriteBody=1, JMSActiveMQBrokerMoldTime=25, JMSActiveMQBrokerInTime=1699852957932, JMSActiveMQBrokerOutTime=1699852957957, JMSCorrelationID=null, JMSCorrelationIDAsBytes=null, JMSDeliveryMode=2, JMSDestination=queue://ADP.IS_2.RES.IN, JMSExpiration=0, JMSMessageID:ID:885e592909dd-40077-1699852130606-3:1:3:1:1, JMSPriority=4, JMSRedelivered=false, JMSReplyTo=null, JMSTimestamp=1699852957926, JMSType=TEST2:FROM_BROKER:SMX, JMSXGroupID=null, JMSXUserID=null, routeId=SRV.BROKER-SWITCH.REQ.SMX, template=file:///pv/test-json2xml.adm, X-B3-ParentSpanId=bcb768651f12f383, X-B3-Sampled=1, X-B3-SpanId=5817215d76939941, X-B3-TraceId=2f4274f8fce0ff28}
Body	<?xml version="1.0" encoding="UTF-8" standalone="no"?><TestReq><Integer>123</Integer><String>String</String></TestReq>

РЕЗУЛЬТАТ: УСПЕХ

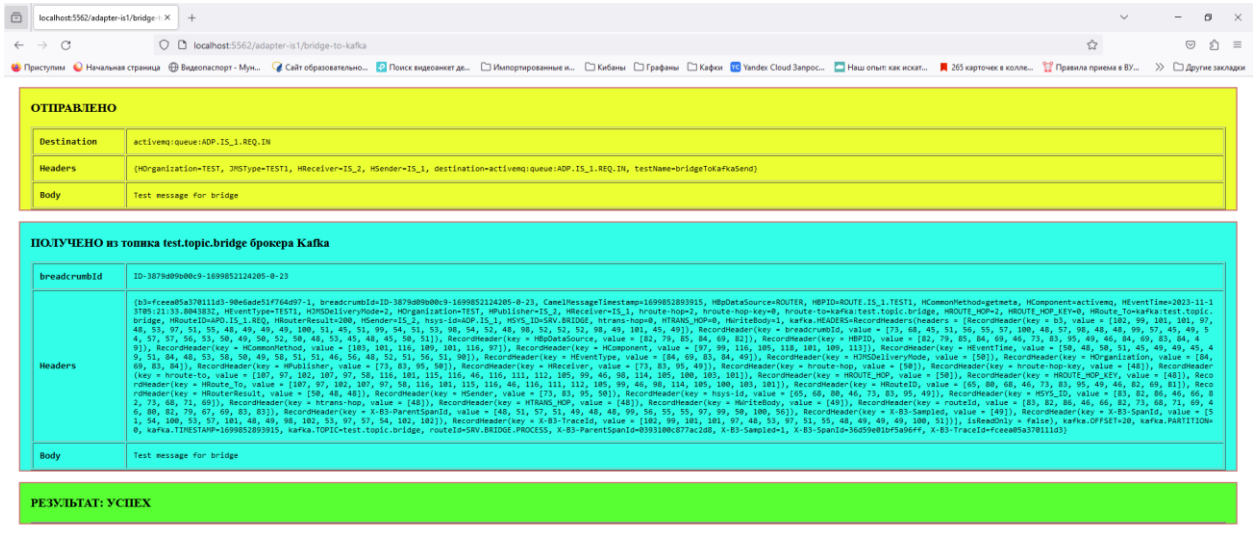
Пример 2. Доставка сообщения по альтернативному маршруту.

Альтернативный маршрут представляет собой доставку сообщения из системы IS_1 в топик кафка системы IS_2.

Демонстрация примера по доставке сообщения от адаптера системы ADP.SYS_1 в адаптер ADP.IS_2:

Открыть в web-браузере страницу http://localhost:5562/adapter-is1/bridge-to-kafka

В результате выполнения сценария сообщение будет доставлено из одной системы в другую. Подробная информация о прохождении по маршруту будет добавлена в логи Платформы.



ОТПРАВЛЕНО

Destination	activemq:queue:ADP.IS_1.REQ.IN
Headers	{HOrganization=TEST, JMSType=TEST1, HReceiver=IS_2, HSender=IS_1, destination=activemq:queue:ADP.IS_1.REQ.IN, testName=bridgeToKafkaSend}
Body	Test message for bridge

ПОЛУЧЕНО из топика test.topic.bridge брокера Kafka

breadcrumbId	ID-3879d09b00c9-1699852124205-0-23
Headers	{b3=fceea05a370111d3-90e6ade514764d97-1, breadcrumbId=ID-3879d09b00c9-1699852124205-0-23, CamelMessageTimestamp=1699852893915, HBpDataSource=ROUTER, HBPID=ROUTE.IS_1.TEST1, HCommonMethod=getmeta, HComponent=activemq, HEventTime=2023-11-13T05:21:33.804383Z, HEventType=TEST1, HJMSDeliveryMode=2, HOrganization=TEST, HPublisher=IS_2, HReceiver=IS_1, hroute-hop=2, hroute-hop-key=0, hroute-to=kafka:test.topic.bridge, HROUTE_HOP=2, HROUTE_HOP_KEY=0, HRoute_To=kafka:test.topic.bridge, HRouteID=APD.IS_1.REQ, HRouterResult=200, HSender=IS_2, hsys-id=ADP.IS_1, HSYS_ID=SRV.BRIDGE, htrans-hop=0, HTRANS_HOP=0, HWriteBody=1, kafka.HEADERS=RecordHeaders(headers = [RecordHeader(key = b3, value = [102, 99, 101, 101, 97, 48, 53, 97, 51, 55, 48, 49, 49, 49, 100, 51, 45, 51, 99, 54, 51, 53, 98, 54, 52, 48, 98, 52, 52, 52, 98, 49, 101, 45, 49]), RecordHeader(key = breadcrumbId, value = [73, 68, 45, 51, 56, 55, 57, 100, 48, 57, 98, 48, 48, 99, 57, 45, 49, 54, 57, 57, 56, 53, 50, 49, 50, 52, 50, 48, 53, 45, 48, 45, 50, 51]), RecordHeader(key = HBpDataSource, value = [82, 79, 85, 84, 69, 82]), RecordHeader(key = HBPID, value = [82, 79, 85, 84, 69, 46, 73, 83, 95, 49, 46, 84, 69, 83, 84, 49]), RecordHeader(key = HCommonMethod, value = [103, 101, 116, 109, 101, 116, 97]), RecordHeader(key = HComponent, value = [97, 99, 116, 105, 118, 101, 109, 113]), RecordHeader(key = HEventTime, value = [50, 48, 50, 51, 45, 49, 49, 45, 49, 51, 84, 48, 53, 58, 50, 49, 58, 51, 51, 46, 56, 48, 52, 51, 56, 51, 90]), RecordHeader(key = HEventType, value = [84, 69, 83, 84, 49]), RecordHeader(key = HJMSDeliveryMode, value = [50]), RecordHeader(key = HOrganization, value = [84, 69, 83, 84]), RecordHeader(key = HPublisher, value = [73, 83, 95, 50]), RecordHeader(key = HReceiver, value = [73, 83, 95, 49]), RecordHeader(key = hroute-hop, value = [50]), RecordHeader(key = hroute-hop-key, value = [48]), RecordHeader(key = hroute-to, value = [107, 97, 102, 107, 97, 58, 116, 101, 115, 116, 46, 116, 111, 112, 105, 99, 46, 98, 114, 105, 100, 103, 101]), RecordHeader(key = HROUTE_HOP, value = [50]), RecordHeader(key = HROUTE_HOP_KEY, value = [48]), RecordHeader(key = HRoute_To, value = [107, 97, 102, 107, 97, 58, 116, 101, 115, 116, 46, 116, 111, 112, 105, 99, 46, 98, 114, 105, 100, 103, 101]), RecordHeader(key = HRouteID, value = [65, 80, 68, 46, 73, 83, 95, 49, 46, 82, 69, 81]), RecordHeader(key = HRouterResult, value = [50, 48, 48]), RecordHeader(key = HSender, value = [73, 83, 95, 50]), RecordHeader(key = hsys-id, value = [65, 68, 80, 46, 73, 83, 95, 49]), RecordHeader(key = HSYS_ID, value = [83, 82, 86, 46, 66, 82, 73, 68, 71, 69]), RecordHeader(key = htrans-hop, value = [48]), RecordHeader(key = HTRANS_HOP, value = [48]), RecordHeader(key = HWriteBody, value = [49]), RecordHeader(key = routeId, value = [83, 82, 86, 46, 66, 82, 73, 68, 71, 69, 46, 80, 82, 79, 67, 69, 83, 83]), RecordHeader(key = X-B3-ParentSpanId, value = [48, 51, 57, 51, 49, 48, 48, 99, 56, 55, 57, 99, 50, 100, 56]), RecordHeader(key = X-B3-Sampled, value = [49]), RecordHeader(key = X-B3-SpanId, value = [51, 54, 100, 53, 57, 101, 48, 49, 98, 102, 53, 97, 57, 54, 102, 102]), RecordHeader(key = X-B3-TraceId, value = [102, 99, 101, 101, 97, 48, 53, 97, 51, 55, 48, 49, 49, 49, 100, 51])), isReadOnly = false), kafka.OFFSET=20, kafka.PARTITION=0, kafka.TIMESTAMP=1699852893915, kafka.TOPIC=test.topic.bridge, routeId=SRV.BRIDGE.PROCESS, X-B3-ParentSpanId=0393100c877ac2d8, X-B3-Sampled=1, X-B3-SpanId=36d59e01bf5a96ff, X-B3-TraceId=fceea05a370111d3}
Body	Test message for bridge

РЕЗУЛЬТАТ: УСПЕХ

bpRouter – сервис идентификации бизнес-процессов.

Сервис позволяет однозначно идентифицировать «Бизнес-процесс» (БП), по которому должно быть маршрутизировано сообщение из одной очереди брокера AMQ в другую очередь, на основе данных стандартизированных заголовков, передаваемых в сообщении.

Вся настроечная информация сервиса хранится в таблице BUSINESS_PROCESES. При запуске сервис считывает данные из таблицы и записывает их во внутреннюю map. Так же в сервисе реализован функционал автоматического обновления настроечной информации. Для этого в установленные промежутки времени сервисом выполняется поиск в таблице в БД. Найденные записи, у которых временная метка обновления позже чем у аналогичных записей во внутренней Map, обновляются актуальными значениями. Так же сервисом контролируется добавление и удаление записей из БД.

Для корректной работы сервиса в конфигурационном файле настраиваются следующие переменные, характеризующие окружение (properties):

Переменная	Значение
MARS.BP_ROUTER.MAIN_QUEUE	очередь для маршрутизируемых сообщений. Значение по умолчанию MARS.BP_ROUTER.
MARS.BP_ROUTER.RULES_QUEUE	очередь для хранения сообщения с настроечной информацией в формате xml. Значение по умолчанию MARS.BP_ROUTER.RULES.
MARS.BP_ROUTER.RULES_TABLE	наименование настроечной таблицы в БД
MARS.BP_ROUTER.RELOAD_QUEUE	очередь для инициирования процесса обновления настроечной информации из БД. Значение по умолчанию MARS.BP_ROUTER.RELOAD.
MARS.BP_ROUTER.STOCK_QUEUE	очередь для сообщений, для которых правила маршрутизации не найдены. Значение по умолчанию MARS.BP_ROUTER.STOCK.TRANSIT.
MARS.BP_ROUTER.HARD_MODE	режим работы сервиса. Если установлено значение «true» и для сообщения не найдена настроечная информация, то значение заголовка "HOrganization" копируется в значение заголовка "HOrganization", а заголовок "HOrganization" удаляется (сообщение уйдет в STOCK). Значение по умолчанию «false».
MARS.BP_ROUTER.JETTY.ADDRESS	адрес REST интерфейса сервиса
MARS.BP_ROUTER.JETTY.PORT	порт REST интерфейса
MARS.BP_ROUTER.HEARTBEAT.INTERVAL	интервал обновления информации о доступности сервиса

Сервис предоставляет смежным системам REST-интерфейс, реализующий следующие методы:

- Метод **GetBP** - позволяет по мета-информации, полученной из заголовков сообщения, идентифицировать БП (HBPID).

- Метод **GetMeta** - позволяет по идентификатору БП (HBPID) получить мета-информацию, необходимую для дальнейшей маршрутизации сообщения, а так же дополнительные настройки для обработки этого сообщения.

- Метод **getProperties -** возвращает json-структуру, соответствующую классу **ModuleProperties** CommonRoutes (с версии 3.4.2) в которой находятся свойства модуля из таблицы **SYS_PROPERTIES**.

При вызове методов используются стандартные заголовки (набор обязательных заголовков варьируется, исходя из требований вызываемого метода):

- **JMSType** – наименование маршрутизируемого сообщения
- **HSender** – идентификатор системы-отправителя сообщения
- **HReceiver** – идентификатор системы –получателя сообщения
- **HOrganization** – идентификатор организации, для которой производится маршрутизация
- **HMessageID –** уникальный **UID** операции
- **HCorrelationId –** используется для сопоставления сообщения –запроса и сообщения-ответа в асинхронном обмене, равно значению HMessageID сообщения-запроса
- **HSYS_ID** – идентификатор программной сущности, которая должна отправить сообщение на данном этапе
- **HBPID** – идентификатор бизнес-процесса
- **HROUTE_HOP** – идентификатор этапа прохождения маршрута
- **HROUTE_HOP_KEY** – указатель выбора записи в таблице маршрутизации для этапа при ветвлении маршрута

Вызываемые методы возвращают в ответ заголовки (для каждого метода определен свой набор заголовков):

- **HBPID** – идентификатор бизнес-процесса
- **JMSType** – наименование маршрутизируемого сообщения
- **HSender** - идентификатор системы-отправителя сообщения
- **HReceiver** – идентификатор системы –получателя сообщения
- **HRoute_To** – наименование очереди, в которую должно быть отправлено сообщение
- **HKey1** – дополнительная информация для обработки сообщения
- **HKey2** - дополнительная информация для обработки сообщения
- **HKey3** - дополнительная информация для обработки сообщения
- **HTrans_Hop** – номер этапа маршрута
- **WRITE_BODY** – производится ли логирование тела сообщения на данном этапе маршрута

Метод GetBP

Метод **GetBp** используется на начальном этапе маршрута прохождения сообщения для получения идентификатора бизнес-процесса, по которому будет осуществляться процедура маршрутизации.

Обязательные заголовки, которые используются при вызове метода:

JMSType, HSender, HReceiver, HOrganization, HSYS_ID

Опционально - HMessageId, HCorrelationId

bpRouter ищет в настроечной таблице маршрутизации запись, в которой значение полученных заголовков совпадает со значением одноименных полей, имеет значение WORK-FLAG = 1 и имеет минимальный номер этапа TRANS_HOP

В случае успешного поиска bbRouter возвращает данные настроек бизнес-процесса в заголовках:

HBPID, HRoute_To, HKey1, HKey2, HKey3, HTrans_Hop, WRITE_BODY.

В случае если Бизнес-процесс не идентифицирован, то сообщение отправляется в очередь MARS.STOCK или иную очередь, с соответствии с правилами STOCK -фильтра.

Метод GetMeta

Метод **GetMeta** позволяет идентификатору бизнес-процесса (заголовок "HBPID") получить мета-информацию, необходимую для интегрируемой системы, а также, дополнительную настроечную информацию, которая может быть использована при обработке конкретного сообщения "внутри" интеграционных сервисов.

Обязательные заголовки, которые используются при вызове метода:

HBPID, HROUTE_HOP, HOrganization, HSYS_ID

Опционально: HROUTE_HOP_KEY, HMessageId, HCorrelationId

bpRouter ищет в настроечной таблице маршрутизации запись, в которой значение полученных заголовков совпадает со значением одноименных полей и имеет значение WORK-FLAG = 1. Если в таблице маршрутизации есть несколько записей, попадающих под критерии, то выбирается первая из найденных.

В случае успешного поиска bbRouter возвращает данные настроек бизнес-процесса в заголовках:

JMSType, HSender, HReceiver, HRoute_To, HKey1, HKey2, HKey3, HTrans_Hop, WRITE_BODY.

В случае если Бизнес-процесс не идентифицирован, то сообщение отправляется в очередь MARS.STOCK или иную очередь, с соответствии с правилами STOCK -фильтра.

Метод getProperties

Метод **getProperties** возвращает json-структуру, соответствующую классу **ModuleProperties** CommonRoutes, в которой находятся свойства модуля из таблицы **SYS_PROPERTIES**. Вызывающий модуль аутентифицируется по заголовкам **hsys-id** и **Authorization** (Bearer TOKEH).

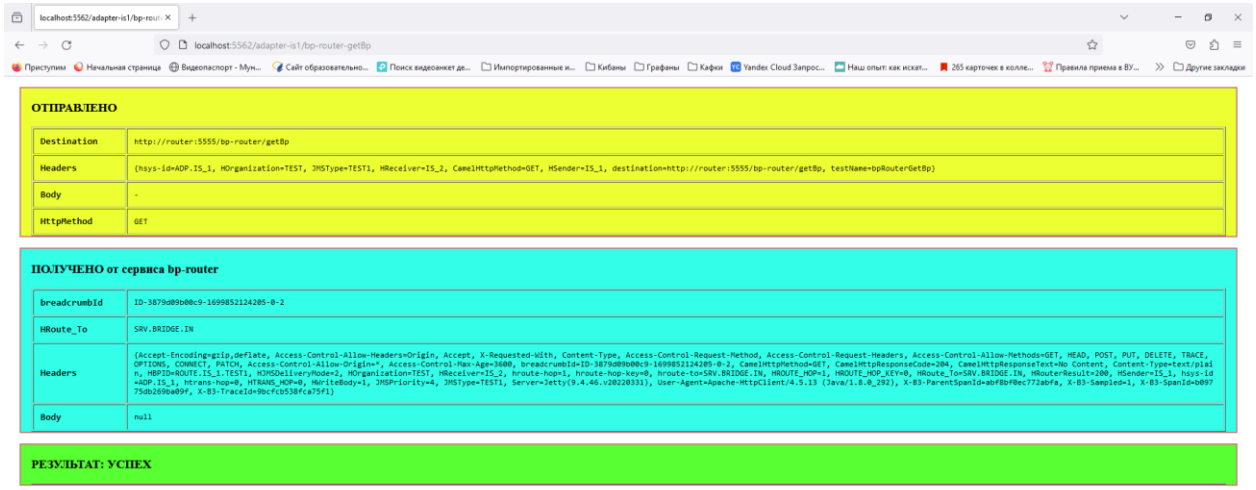
Для работы механизма аутентификации необходимо в таблицу **SYS_PROPERTIES** для каждого модуля-потребителя добавить строку с настройками

Поле	Значение
SYS_ID	bp-router
PROPERTY_NAME	AUTH_TOKEN
PROPERTY_VALUE	<TOKEH>
KEY1	<ID_МОДУЛЯ>

Примеры

Пример 1. Проверка метода getBp

Для проверки работоспособности метода нужно открыть в web-браузере страницу http://localhost:5562/adapter-is1/bp-router-getBp. В результате запроса будут возвращены значения устанавливаемых во время выполнения запроса заголовков:



ОТПРАВЛЕНО

Destination	http://router:5555/bp-router/getBp
Headers	{hsys-id=AOP.IS_1, HOrganization=TEST, JMSType=TEST1, HReceiver=IS_2, CamelHttpMethod=GET, HSender=IS_1, destination=http://router:5555/bp-router/getBp, testName=bpRouterGetBp}
Body	-
HttpMethod	GET

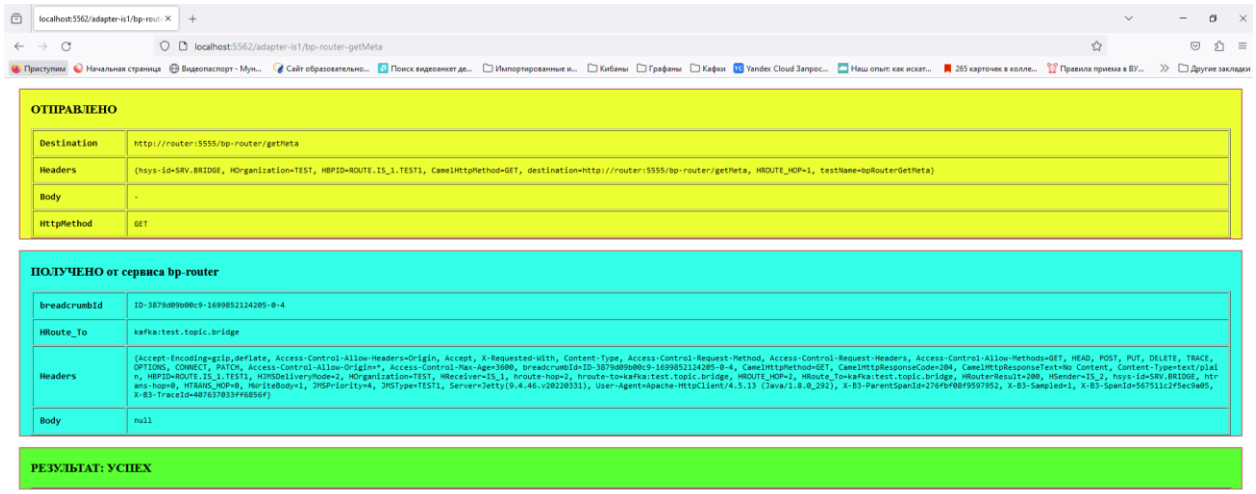
ПОЛУЧЕНО от сервиса bp-router

breadcrumbId	ID-3879d09b00c9-1699852124205-0-2
HRoute_To	SRV.BRIDGE.IN
Headers	{Accept-Encoding=gzip,deflate, Access-Control-Allow-Headers=Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Methods=GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH, Access-Control-Allow-Origin=*, Access-Control-Max-Age=3600, breadcrumbId=ID-3879d09b00c9-1699852124205-0-2, CamelHttpMethod=GET, CamelHttpResponseCode=204, CamelHttpResponseText=No Content, Content-Type=text/plain, HBPID=ROUTE.IS_1.TEST1, HJMSDeliveryMode=2, HOrganization=TEST, HReceiver=IS_2, hroute-hop=1, hroute-hop-key=0, hroute-to=SRV.BRIDGE.IN, HROUTE_HOP=1, HROUTE_HOP_KEY=0, HRoute_To=SRV.BRIDGE.IN, HRouterResult=200, HSender=IS_1, hsys-id=AOP.IS_1, htrans-hop=0, HTRANS_HOP=0, HWriteBody=1, JMSPriority=4, JMSType=TEST1, Server=Jetty(9.4.46.v20220331), User-Agent=Apache-HttpClient/4.5.13 (Java/1.8.0_292), X-B3-ParentSpanId=abf8bf0ec772abfa, X-B3-Sampled=1, X-B3-SpanId=b09775db269ba09f, X-B3-TraceId=9bcfcb538fca75f1}
Body	null

РЕЗУЛЬТАТ: УСПЕХ

Пример 2: Проверка метода getMeta

Для проверки работоспособности метода нужно открыть в web-браузере страницу http://localhost:5562/adapter-is1/bp-router-getMeta. В результате запроса будут возвращены значения устанавливаемых во время выполнения запроса заголовков:



ОТПРАВЛЕНО

Destination	http://router:5555/bp-router/getMeta
Headers	{hsys-id=SRV.BRIDGE, HOrganization=TEST, HBPID=ROUTE.IS_1.TEST1, CamelHttpMethod=GET, destination=http://router:5555/bp-router/getMeta, HROUTE_HOP=1, testName=bpRouterGetMeta}
Body	-
HttpMethod	GET

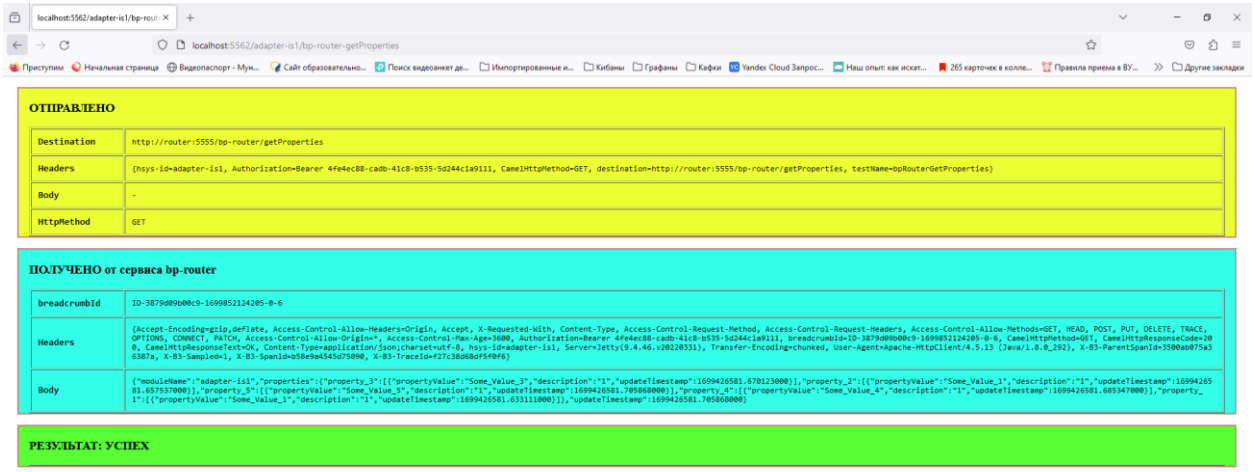
ПОЛУЧЕНО от сервиса bp-router

breadcrumbId	ID-3879d09b00c9-1699852124205-0-4
HRoute_To	kafka:test.topic.bridge
Headers	{Accept-Encoding=gzip,deflate, Access-Control-Allow-Headers=Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Methods=GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH, Access-Control-Allow-Origin=*, Access-Control-Max-Age=3600, breadcrumbId=ID-3879d09b00c9-1699852124205-0-4, CamelHttpMethod=GET, CamelHttpResponseCode=204, CamelHttpResponseText=No Content, Content-Type=text/plain, HBPID=ROUTE.IS_1.TEST1, HJMSDeliveryMode=2, HOrganization=TEST, HReceiver=IS_1, hroute-hop=2, hroute-to=kafka:test.topic.bridge, HROUTE_HOP=2, HRoute_To=kafka:test.topic.bridge, HRouterResult=200, HSender=IS_2, hsys-id=SRV.BRIDGE, htrans-hop=0, HTRANS_HOP=0, HWriteBody=1, JMSPriority=4, JMSType=TEST1, Server=Jetty(9.4.46.v20220331), User-Agent=Apache-HttpClient/4.5.13 (Java/1.8.0_292), X-B3-ParentSpanId=276fbf08f9597952, X-B3-Sampled=1, X-B3-SpanId=567511c2f5ec9a05, X-B3-TraceId=407637033ff6856f}
Body	null

РЕЗУЛЬТАТ: УСПЕХ

Пример 2: Проверка метода getProperties

Для проверки работоспособности метода нужно открыть в web-браузере страницу http://localhost:5562/adapter-is1/bp-router-getMeta. В результате запроса в теле сообщения будет передана JSON-структура properties.



ОТПРАВЛЕНО

Destination	http://router:5555/bp-router/getProperties
Headers	{hsys-id=adapter-is1, Authorization=Bearer 4fe4ec88-cadb-41c8-b535-5d244c1a9111, CamelHttpMethod=GET, destination=http://router:5555/bp-router/getProperties, testName=bpRouterGetProperties}
Body	-
HttpMethod	GET

ПОЛУЧЕНО от сервиса bp-router

breadcrumbId	ID-3879d09b00c9-1699852124205-0-6
Headers	{Accept-Encoding=gzip,deflate, Access-Control-Allow-Headers=Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Methods=GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH, Access-Control-Allow-Origin=*, Access-Control-Max-Age=3600, Authorization=Bearer 4fe4ec88-cadb-41c8-b535-5d244c1a9111, breadcrumbId=ID-3879d09b00c9-1699852124205-0-6, CamelHttpMethod=GET, CamelHttpResponseCode=200, CamelHttpResponseText=OK, Content-Type=application/json;charset=utf-8, hsys-id=adapter-is1, Server=Jetty(9.4.46.v20220331), Transfer-Encoding=chunked, User-Agent=Apache-HttpClient/4.5.13 (Java/1.8.0_292), X-B3-ParentSpanId=3500ab075a36387a, X-B3-Sampled=1, X-B3-SpanId=b58e9a4545d75090, X-B3-TraceId=f27c38d68df5f0f6}
Body	{"moduleName":"adapter-is1","properties":{"property_3":[{"propertyValue":"Some_Value_3","description":"1","updateTimestamp":1699426581.670123000}],"property_2":[{"propertyValue":"Some_Value_1","description":"1","updateTimestamp":1699426581.657537000}],"property_5":[{"propertyValue":"Some_Value_5","description":"1","updateTimestamp":1699426581.705868000}],"property_4":[{"propertyValue":"Some_Value_4","description":"1","updateTimestamp":1699426581.685347000}],"property_1":[{"propertyValue":"Some_Value_1","description":"1","updateTimestamp":1699426581.633111000}]},"updateTimestamp":1699426581.705868000}

РЕЗУЛЬТАТ: УСПЕХ

ATLASMAP-COMMON – Универсальный сервис трансформации через компонент camel-atlasmap

Сервис преобразования сообщений является транслятором, реализующим интеграционный обмен прикладных систем, использующих разные форматы данных: JSON и XML соответственно.

С помощью UI ATLASMAP создается маппинг двух файлов (соответствие полей источника полям целевого документа). Он выгружается и сохраняется как adm архив. В нём содержится json файл, который содержит необходимую информацию. Файл помещается в хранилище с уникальным названием, отражающим бизнес-процесс.

При получении запроса через очереди сервис берет имя файла трансформации из заголовка входящего сообщения **HKey1**. Ищет его на ресурсе, использует для трансформации и отдает дальше по маршруту. В случае ошибки из заголовка **HKey2** понимает, какого формата было входное сообщение (*json* или *xml*), создает соответствующий ответ с информацией об ошибке и отправляет его реквестору.

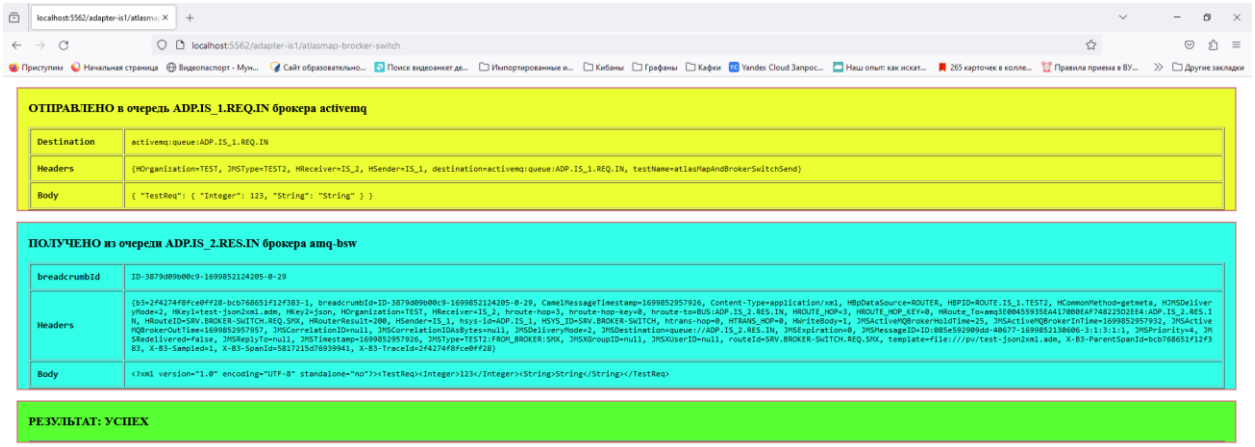
Для настройки Бизнес-процесса необходимо указывать название сервиса (SYS_ID) **SRV.ATLASMAP-COMMON**, входящую очередь **SRV.ATLASMAP-COMMON.IN**, заголовок **HKey1** равный названию файла трансформации atlasmap с расширением *.json* и заголовок **HKey2** с типом входящего файла (*json* или *xml*), для формирования корректного формата ответа в случае ошибки.

Примеры

Пример1. Трансформация формата JSON в XML.

Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/atlasmap-brocker-switch

В результате выполнения сценария сообщение будет доставлено из одной системы в другую, по пути формат тела сообщения трансформирован из JSON в XML. Подробная информация о прохождении по маршруту будет добавлена в логи Платформы.



AUTH-COMMON – сервис генерации JWT-токенов

Сервис аутентификации реализует генерацию, обновление и выдачу JWT-токенов.

Взаимодействие с сервисом строится по REST API. Сервис реализует методы, обеспечивающие следующие возможности:

- **Auth request** получение пары (access token, refresh token) по идентификационному токену;
- **Refresh request** обновление пары (access token, refresh token), по выданному ранее refresh токену.
- **Public key /auth-common/key** получение публичного ключа

Access_token имеет срок жизни от 1 до 10 минут. Подписывается с применением ассиметричного алгоритма шифрования RS256. В тело JWT токена могут быть добавлены пользовательские характеристики (Customer Claims), предварительно указанные в первичном идентификационном токене.

Содержимое access_token:

- заголовок { "alg": "RS256", "typ": "JWT"}
- *полезная нагрузка*: {
  "exp": <время прекращения действия>,
  "authID": <id токена, по которому клиент идентифицировался>,
  "services_access": [<Сервисы, к которым разрешен доступ>]
- - *подпись* с применением только ассиметричного алгоритма шифрования RS256

Refresh token имеет срок жизни от 1 месяца до 1 года. Refresh token генерируется с помощью uuid4.

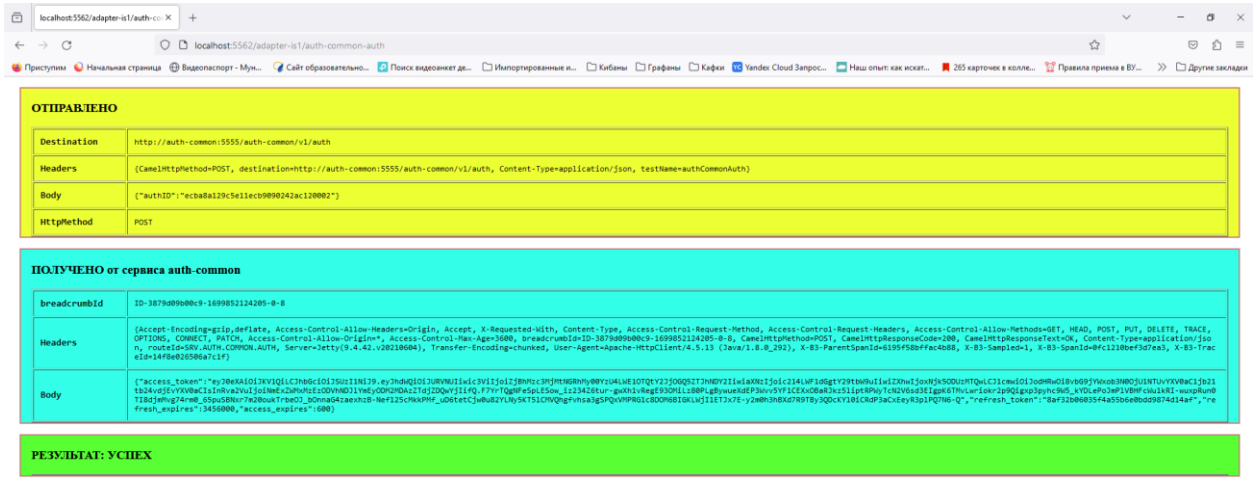
Refresh токен может быть выпущен на основании ранее выданного одноразового refresh токена, либо на основании ранее выданного идентификационного токена.

Первичный идентификационный токен хранится в базе сервиса, генерируется вручную, добавлением в таблицу БД ident_token. Описание таблицы БД смотрите **в Приложении № 1**

Примеры

Пример: Проверка метода auth
Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/auth-common-auth.

В результате запроса в теле сообщения будут представлены access_token и refresh_token. Подробная информация о прохождении сообщения размещена в логах Платформы.



BROKER-SWITH – сервис передачи сообщений между брокерами

Сервис предназначен для передачи сообщений между разными брокерами.

Подключение брокера осуществляется путем указания в конфигурационном файле имени брокера и соответствующих настроек подключения.

Настройка передачи осуществляется в таблице маршрутизации, для этого добавляются 2 дополнительных этапа прохождения.

- В первом этапе описывается процесс доставки сообщения в очередь SRV.BROKER.SWITH.IN в переменной ROUTE_TO, все остальные значения описываются исходя из данных основного маршрута.
- Во втором этапе описывается обработка сообщения сервисом BROKER-SWITH – в поле SYS_ID указывается идентификатор сервиса SRV.BROKER-SWITH, в поле ROUTE_TO указывается очередь, в которую должно быть помещено сообщение с указанием префикса **BROKER_NAME:** соответствующего имени брокера, в который предусматривается доставка сообщения.

Дополнительно реализована возможность возвращения ответа на запрос тому же брокеру, от которого запрос поступил. Для этого в таблице маршрутизации, на этапе передачи сообщения-запроса, при заполнении JMSType добавляется суффикс с именем брокера-источника

:FROM_BROKER:{BROKER_NAME},

где **FROM_BROKER:** - маркер, указывающий сервису на наличие в данном заголовке имени брокера, **{BROKER_NAME}** - имя брокера-источника, из которого пришло сообщение.

Для корректной работы сервиса необходимо, чтобы на отрезке маршрута, от момента добавления данного суффикса до момента его использования (на этапах, описывающих прохождение ответа), заголовок JMSType не перезаписался при

вызовах метода getMeta сервиса bp-router, то есть поле JMS_TYPE в таблице маршрутизации на этих этапах должно быть пустым.

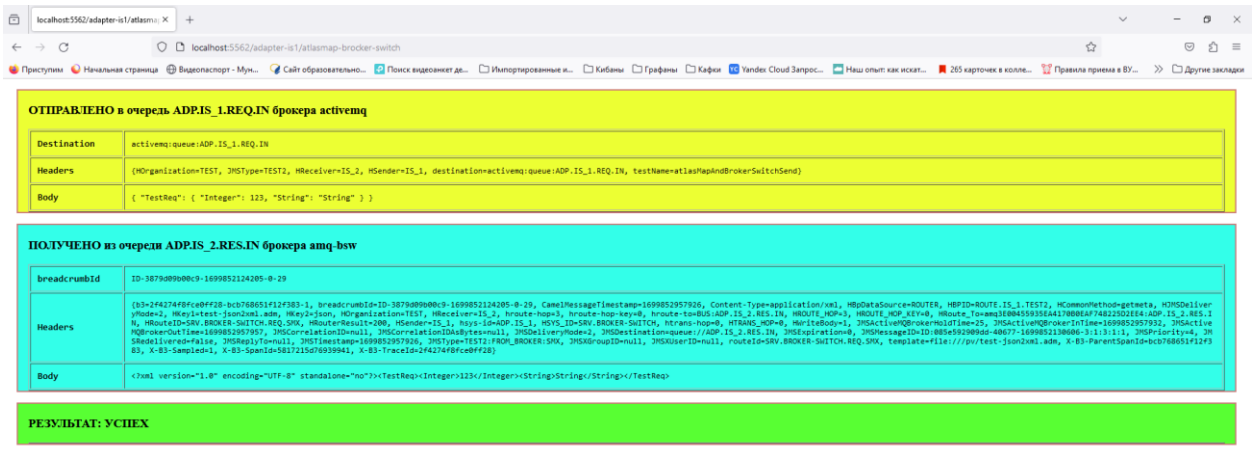
Сервис также предоставляет функционал заполнения заголовков HCorrelationMsgID и HMessageID (обычно требуется на маршруте ответа). Значение HMessageID записывается в HCorrelationMsgID, а для самого HMessageID генерируется новое значение. Для включения данного функционала необходимо в таблице маршрутизации на этапе ответа в строке сервиса в поле KEY3 указать значение CHANGE_HEADERS.

Пример

Пример1. Доставка сообщения от брокера 1 (activemq) – адаптера системы IS_1 в брокер 2 (amq_bsw), который обслуживает адаптер системы IS_2

Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/atlasmap-brocker-switch

В результате выполнения сценария сообщение будет доставлено из одной системы в другую, от одного брокера AMQ в другой брокер AMQ. Подробная информация о прохождении по маршруту будет добавлена в логи Платформы.



Store – сервис-планировщик отправки сообщений

Данный сервис выполняет функции планировщика и может использоваться в качестве:

- промежуточного звена при отправке сообщений в тех случаях, когда для успешного выполнения взаимодействия может потребоваться несколько циклов отправки одного сообщения;
- кеша сообщений, когда полученные один раз данные могут потребоваться при обработке несколько раз в дальнейшем.

Взаимодействие с сервисом строится по REST API. Сервис поддерживает аутентификацию по токенам доступа.

В REST API доступен один ендпоит **/store/message.** Сохранение сообщения осуществляется методом **POST**, а получение сохраненной информации по сообщению - методом **GET**. Аутентификационная информация передается в заголовке **Authorization** в формате

Bearer <токен доступа>

где **<токен доступа>** - уникальная последовательность символов, передаваемая системой-отправителем при сохранении сообщения. Получить информацию по сохраненному сообщению можно только передав в заголовке **Authorization** точно такой же токен доступа, который был указан при его сохранении. В противном случае сервис вернет ошибку **404**. Ограничений на длину токена доступа нет.

Если при сохранении сообщения была указана очередь для отправки, то по окончании срока хранения сервис перед удалением из базы отправляет копию хранящегося сообщения в указанную очередь.

В качестве хранилища данных сервис использует Redis.

Для сохранения сообщения необходимо отправить **POST** запрос на ендпоит **/store/message**. По умолчанию сохраняется только тело сообщения-запроса. В случае необходимости сохранения заголовков можно в заголовке **headerPatterns** передать список regexp через запятую с масками имен заголовков для сохранения.

Параметры POST запроса:

Параметр	Описание	Обязательный
sys_id	Идентификатор системы-отправителя	да
Reqid	id запроса	да
Waitbefore	время (количество миллисекунд с начала Эпохи), до которого хранить сообщение	да
Authorization	токен авторизации в формате **Bearer <токен доступа>**. Если не указан, считается пустым.	нет
headerPatterns	regexp`ы для имен сохраняемых заголовков с разделителем «,». Если заголовок не передан, то никакие заголовки исходного сообщения не сохраняются.	нет
Queuename	имя очереди для отправки сообщения по окончании срока хранения (waitbefore). Если не указано, то сообщение хранится как в кэше и при наступлении времени waitbefore просто удаляется.	нет
lock_mode	если в заголовке передано значение "**fast**", то сохранение сообщения осуществляется в потоконебезопасном режиме без блокировок. Не рекомендуется использовать этот режим при многопоточных циклах сохранения/чтения сообщения с одним и тем же reqid (например, для обогащения сообщения). Однако, в большинстве случаев этот режим вполне безопасен и ускоряет работу.	нет

Кроме того, могут передаваться любые заголовки, имена которых соответствуют паттернам, переданным в заголовке headerPatterns.

Извлечение сообщения

До наступления времени, переданного в заголовке waitbefore при сохранении, сообщение может быть получено отправкой **GET** запроса на ендпоинт **/store/message/{id запроса}**.

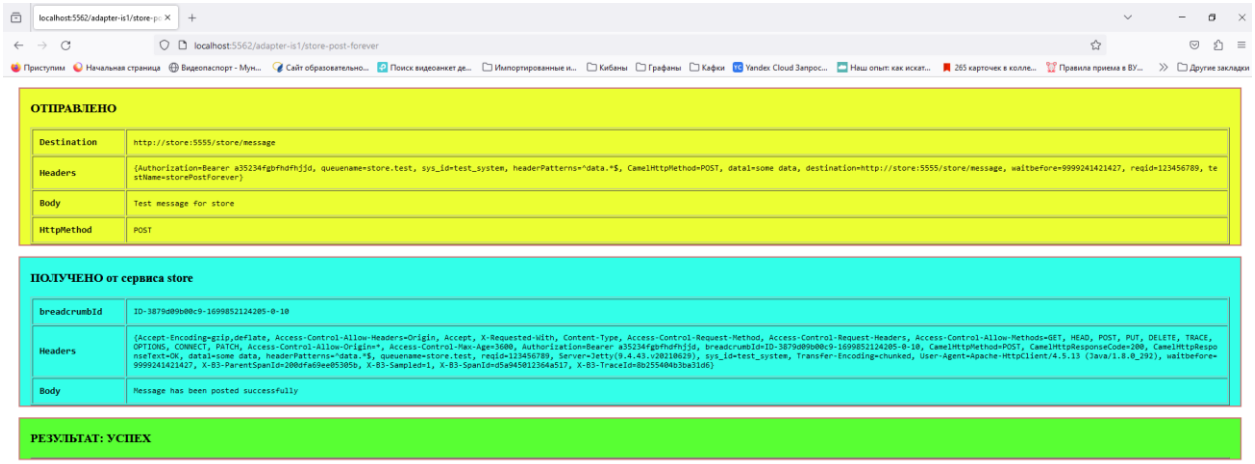
Параметры GET запроса

Параметр	Описание	Обязательный
sys_id	идентификатор системы-отправителя	да
mode	если = **remove**, то сообщение удаляется из store после извлечения, иначе - остается	нет
lock_mode	если в заголовке передано значение "**fast**", то извлечение сообщения осуществляется в потоконебезопасном режиме без блокировок. Не рекомендуется использовать этот режим при многопоточных циклах сохранения/чтения сообщения с одним и тем же reqid (например, для обогащения сообщения). Однако, в большинстве случаев этот режим вполне безопасен и ускоряет работу.	нет
Authorization	токен авторизации в формате **Bearer <токен доступа>**. Если не указан, считается пустым. Должен совпадать с токеном доступа, который был указан при его сохранении. В противном случае сервис вернет ошибку **404**.	нет

Примеры

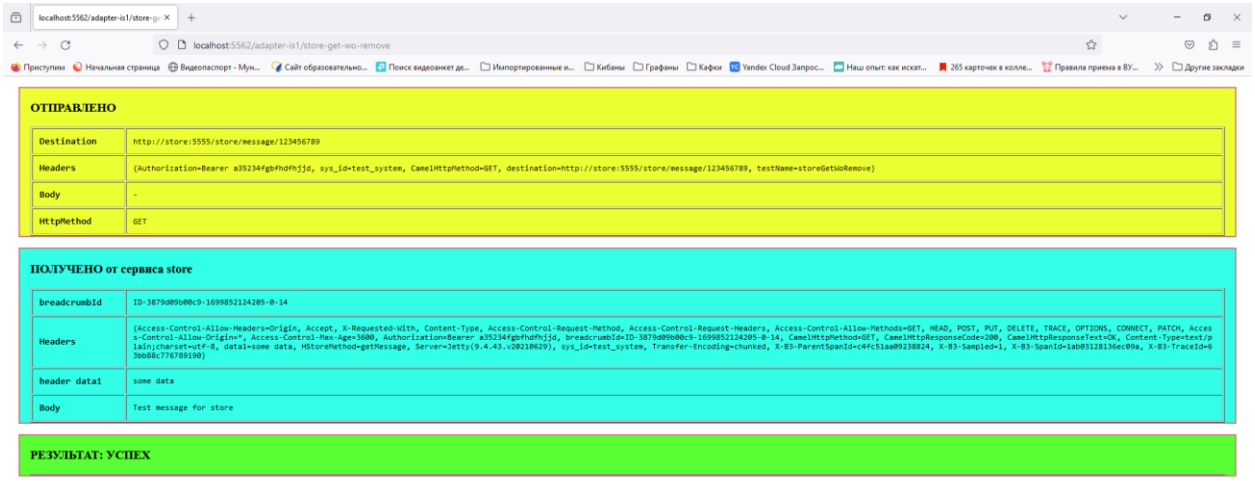
Пример: Проверка метода /store/message сохранение сообщения

Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/store-post-forever



Получение сообщения и удаление после получения

Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/store-get-wo-remove



ОТПРАВЛЕНО

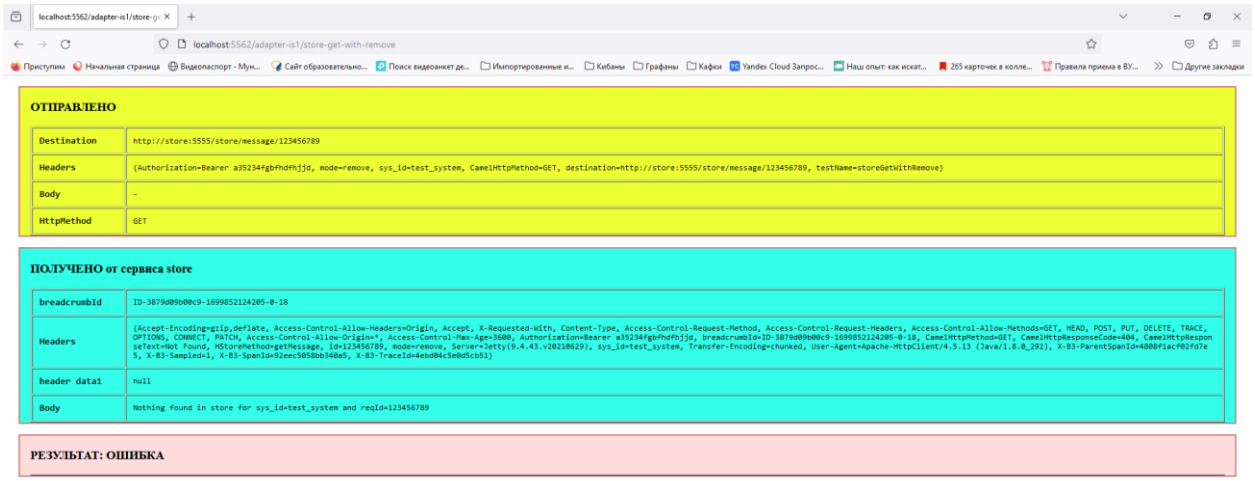
Destination	http://store:5555/store/message/123456789
Headers	{Authorization=Bearer a35234fgbfhdfhjjd, sys_id=test_system, CamelHttpMethod=GET, destination=http://store:5555/store/message/123456789, testName=storeGetWoRemove}
Body	-
HttpMethod	GET

ПОЛУЧЕНО от сервиса store

breadcrumbId	ID-3879d09b00c9-1699852124205-0-14
Headers	{Access-Control-Allow-Headers=Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Methods=GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH, Access-Control-Allow-Origin=*, Access-Control-Max-Age=3600, Authorization=Bearer a35234fgbfhdfhjjd, breadcrumbId=ID-3879d09b00c9-1699852124205-0-14, CamelHttpMethod=GET, CamelHttpResponseCode=200, CamelHttpResponseText=OK, Content-Type=text/plain;charset=utf-8, data1=some data, HStoreMethod=getMessage, Server=Jetty(9.4.43.v20210629), sys_id=test_system, Transfer-Encoding=chunked, X-B3-ParentSpanId=c4fc51aa09238824, X-B3-Sampled=1, X-B3-SpanId=1ab0312b136ec09a, X-B3-TraceId=63bb88c776789190}
header data1	some data
Body	Test message for store

РЕЗУЛЬТАТ: УСПЕХ

Проверка после удаления http://localhost:5562/adapter-is1/store-get-with-remove



ОТПРАВЛЕНО

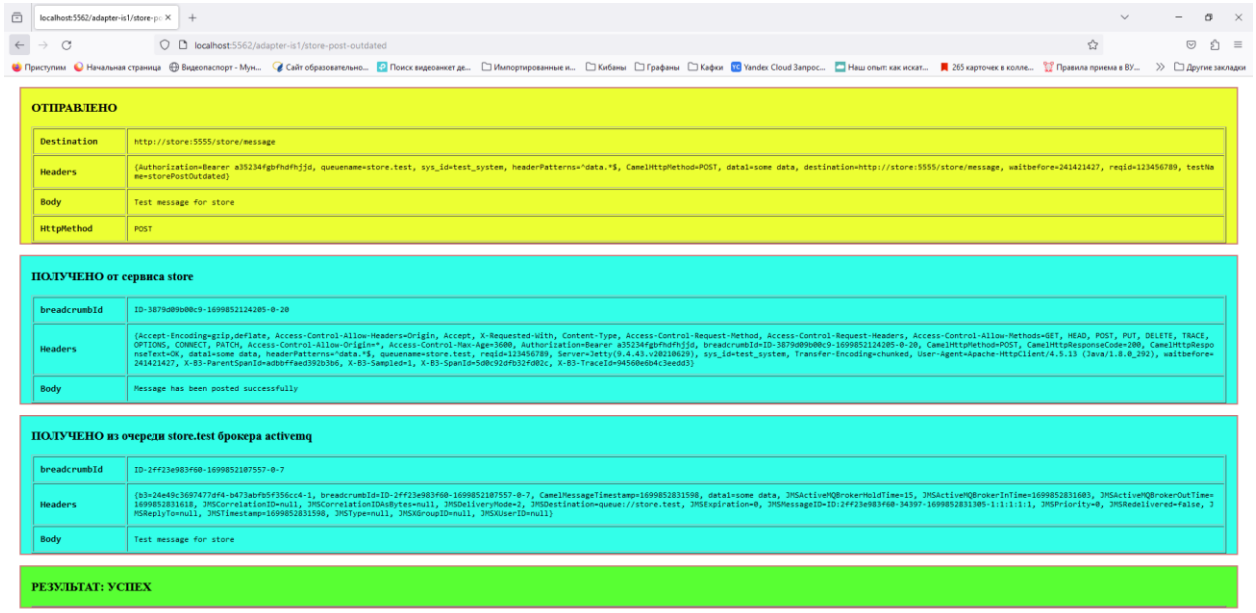
Destination	http://store:5555/store/message/123456789
Headers	{Authorization=Bearer a35234fgbfhdfhjjd, mode=remove, sys_id=test_system, CamelHttpMethod=GET, destination=http://store:5555/store/message/123456789, testName=storeGetWithRemove}
Body	-
HttpMethod	GET

ПОЛУЧЕНО от сервиса store

breadcrumbId	ID-3879d09b00c9-1699852124205-0-18
Headers	{Accept-Encoding=gzip,deflate, Access-Control-Allow-Headers=Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Methods=GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH, Access-Control-Allow-Origin=*, Access-Control-Max-Age=3600, Authorization=Bearer a35234fgbfhdfhjjd, breadcrumbId=ID-3879d09b00c9-1699852124205-0-18, CamelHttpMethod=GET, CamelHttpResponseCode=404, CamelHttpResponseText=Not Found, HStoreMethod=getMessage, id=123456789, mode=remove, Server=Jetty(9.4.43.v20210629), sys_id=test_system, Transfer-Encoding=chunked, User-Agent=Apache-HttpClient/4.5.13 (Java/1.8.0_292), X-B3-ParentSpanId=4808f1acf02fd7e5, X-B3-Sampled=1, X-B3-SpanId=92eec5058bb340a5, X-B3-TraceId=4ebd04c5e0d5cb51}
header data1	null
Body	Nothing found in store for sys_id=test_system and reqId=123456789

РЕЗУЛЬТАТ: ОШИБКА

Получение сообщения из очереди брокера http://localhost:5562/adapter-is1/store-post-outdated



THROTTLE – сервис-планировщик траффика сообщений

Сервис SRV.THROTTLE предназначен для замедления потока сообщений. Пропускает N сообщений в единицу времени. Остальные сообщения ожидают своей очереди.

Если необходимо контролировать поток сообщений по какому-то бизнес-процессу, можно использовать данный сервис. В маршрут для конкретного БП добавляется дополнительная строка с необходимыми параметрами (количество пропускаемых сообщений в единицу времени). В конфигурацию сервиса добавляется настройка с именем этого БП. С этого момента сообщения будут на пути проходить через дроссель.

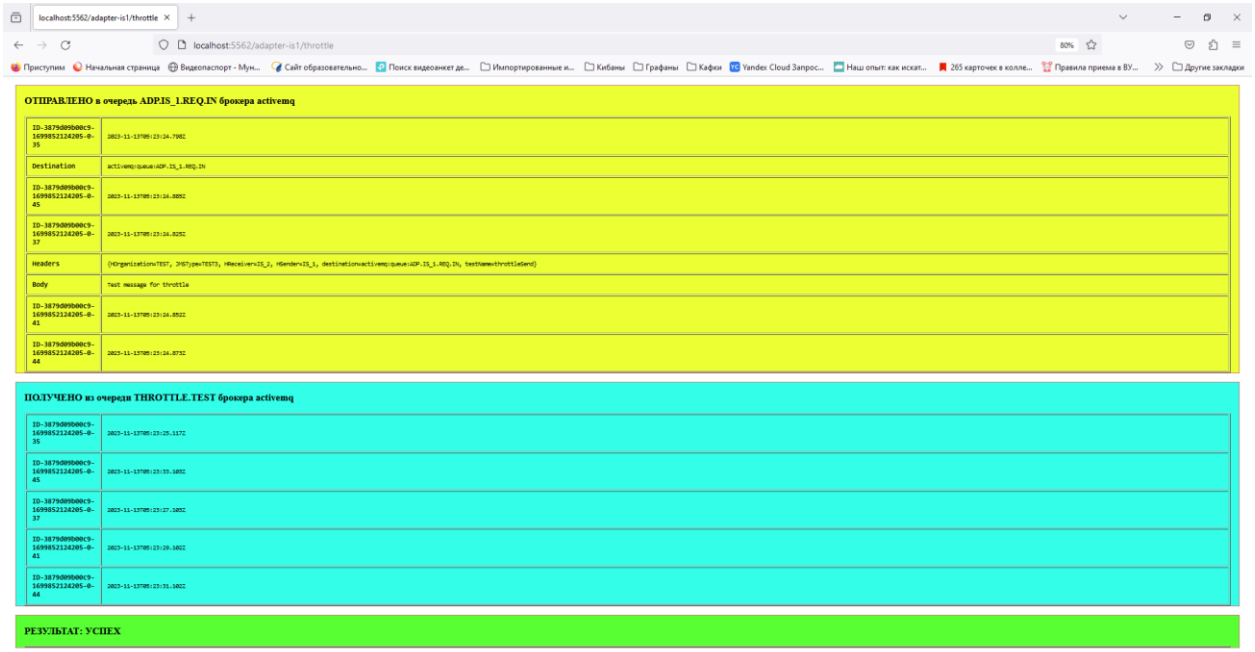
Система абонентов позволяет настроить разную пропускную способность для каждого отдельного БП. Для этого используются параметры BANKS, *.THROTTLE.AMQ.ABONENTS, *.THROTTLE.AMQ.ABONENTS, *.THROTTLE.KAFKA.ABONENTS *.THROTTLE.KAFKA.AUTOSTARTUP. Для каждого абонента создается входная очередь (или топик), которая указывается в таблице маршрутизации. Для контроля количества пропускаемых сообщений используется параметр HKey1. В нём необходимо указывать количество сообщений в секунду.

Примеры.

Пример 1. Проверка задержки сообщений – доставка задерживается на 2 сек.

Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/throttle

В течении 2 секунд сообщения будут задерживаться в очереди THROTTLE.TEST



STOCK.FILTER–сервис дополнительной обработки маршрутизации (фильтрация сообщений)

Сервис STOCK.FILTER предназначен для дополнительной обработки сообщений, направляемых в ту или иную очередь (обычно необходимо при обработке ошибок маршрутизации).

Сервис вычитывает сообщения из очередей, заданных конфигурацией, и применяет к ним определенные правила фильтрации.

В качестве основной конфигурации используются данные из таблиц базы данных сервиса - destinations, strategies и rules.

Таблица Destinations

Поле	Описание
destination_from	очередь, из которой сервис вычитывает сообщения
default_destination	очередь, в которую отправляется сообщение в случае, если оно не удовлетворяет правилам фильтрации из списка стратегий, указанного в поле strategies
strategies	список имен стратегий (через запятую) из таблицы strategies, согласно которым происходит обработка сообщений, полученных из очереди, указанной в поле destination_from
log_message	включает/выключает логирование сообщения

Таблица Strategies

Поле	Описание
strategy_name	имя стратегии
strategy_value	действие, осуществляемое при выполнении правил фильтрации, относящихся к данной стратегии. В данный момент есть 2 варианта возможных значений данного поля: • название очереди (стратегия "REDIRECT") - сообщение будет отправлено в эту очередь • константа 'DROP' (стратегия "DROP") - сообщение будет удалено В каждом из вариантов возможно указание ttl для правил фильтрации, относящихся к данной стратегии, - через символ '@' в данном поле. По истечении данного ttl (которое будет отсчитываться от значения update_timestamp в записи конкретного правила фильтрации из таблицы rules) правила фильтрации, относящиеся к данной стратегии, удаляются сервисом из таблицы rules. Запись самой стратегии в таблице strategies при этом остается.

Таблица Rules

Поле	Описание
strategy_name	имя стратегии, должно соответствовать одной из стратегий из таблицы strategies
filter_rule	правило фильтрации - условие, при выполнении которого применяется данная стратегия. Представляет собой список (через точку с запятой) регулярных выражений для заголовков и тела сообщения в формате {headerName}=regex или Body=regex. При этом, знак-разделитель '**;**' выполняет в данном случае роль логического оператора 'AND', а наличие нескольких строк, относящихся к одной стратегии, подразумевает между ними логический оператор 'OR'.

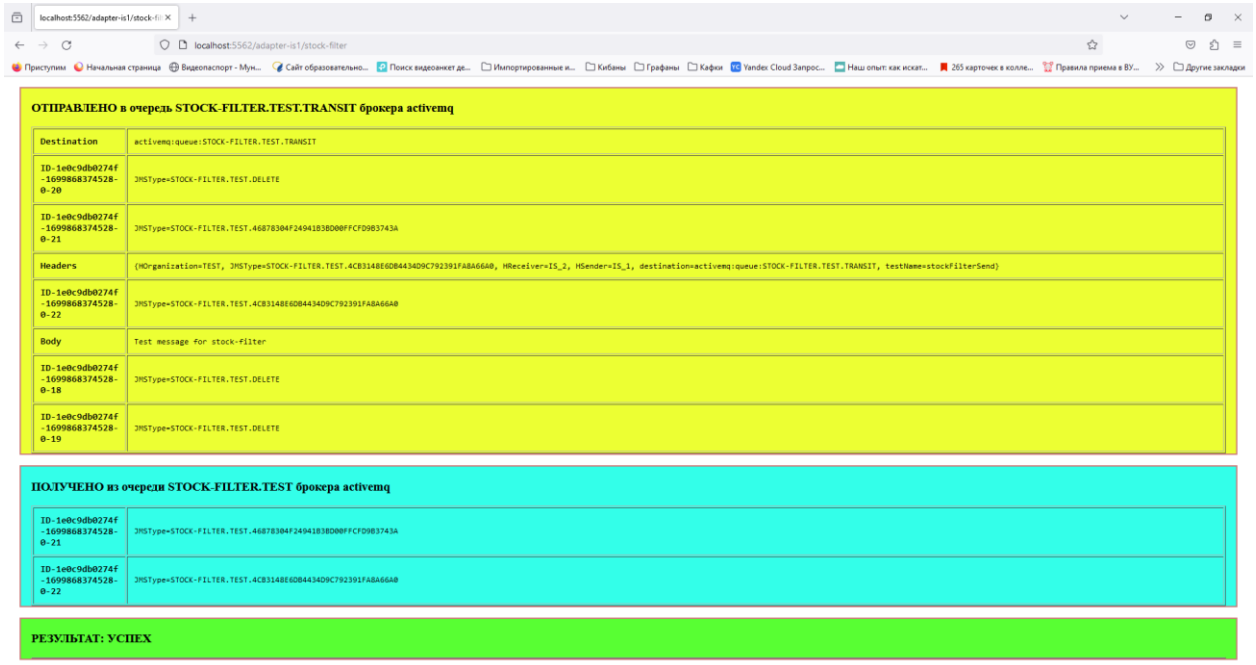
При обработке сообщений сначала применяются "REDIRECT" стратегии, затем "DROP".

Сервис с определенной периодичностью анализирует состояние своей БД и, в случае обнаружения в ней каких-либо изменений, обновляет свое состояние - подгружает новые стратегии и правила фильтрации (или изменения в них). Однако при добавлении новых очередей-источников (т.е. новых строк в таблицу destinations) необходим перезапуск сервиса.

Пример

Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/stock-filter

В указанном примере сообщение по преднастроенным правилам направлено в очередь STOK-FILTER.TEST брокера AMQ.



ОТПРАВЛЕНО в очередь STOCK-FILTER.TEST.TRANSIT брокера activemq

Destination	activemq:queue:STOCK-FILTER.TEST.TRANSIT
ID-1e0c9db0274f-1699868374528-0-20	JMSType=STOCK-FILTER.TEST.DELETE
ID-1e0c9db0274f-1699868374528-0-21	JMSType=STOCK-FILTER.TEST.46878304F24941B38D00FFCFD9B3743A
Headers	{HOrganization=TEST, JMSType=STOCK-FILTER.TEST.4CB314BE6DB4434D9C792391FA8A66A0, HReceiver=IS_2, HSender=IS_1, destination=activemq:queue:STOCK-FILTER.TEST.TRANSIT, testName=stockFilterSend}
ID-1e0c9db0274f-1699868374528-0-22	JMSType=STOCK-FILTER.TEST.4CB314BE6DB4434D9C792391FA8A66A0
Body	Test message for stock-filter
ID-1e0c9db0274f-1699868374528-0-18	JMSType=STOCK-FILTER.TEST.DELETE
ID-1e0c9db0274f-1699868374528-0-19	JMSType=STOCK-FILTER.TEST.DELETE

ПОЛУЧЕНО из очереди STOCK-FILTER.TEST брокера activemq

ID-1e0c9db0274f-1699868374528-0-21	JMSType=STOCK-FILTER.TEST.46878304F24941B38D00FFCFD9B3743A
ID-1e0c9db0274f-1699868374528-0-22	JMSType=STOCK-FILTER.TEST.4CB314BE6DB4434D9C792391FA8A66A0

РЕЗУЛЬТАТ: УСПЕХ

Common-Routers – осуществляет передачу сообщения в указанную очередь брокера.

Сервис Common-Routes реализует стандартные маршруты, сопряженные с обращением к сервисам Платформы. Смежные сервисы, посредством direct-vm, передают сообщения, для которых необходимо выполнение стандартизированных сценариев.

В сервисе реализованы следующие стандартные маршруты:

- **direct:getBp**
- **direct:getMeta**
- **direct:finish**
- **direct:fault**

При отправке финального сообщения в брокер AMQ необходимо пользоваться маршрутом **direct:finish**. В нем перед отправкой сообщения в очередь, указанную в заголовке HRoute_To, осуществляется проверка на превышение максимального размера сообщения для брокера.

В случае превышения максимального размера сообщения только логируется сообщение об ошибке. Само сообщение теряется. Эта проверка необходима для исключения зацикливания при отправки слишком больших сообщений и падения брокера.

Так же в контекст добавлен лог форматтер, автоматически форматирующий Exchange при выводе в лог. При этом в соответствии с настройками в лог выводятся breadcrumbId, заголовки (Headers), свойства (Properties), тело сообщения и его тип, а также, сообщение об ошибке и stackTrace (если обнаружено исключение). По умолчанию настроен вывод в лог только всех заголовков сообщения. ело сообщения выводится в лог в соответствии с настройкой заголовка **HWriteBody** в таблице маршрутизации.

Пример

В качестве примера использования приведен фрагмент кода адаптера ADP.IS_1.

```
from(SMXCoreMain.AMQ_COMPONENT_NAME + ":" + adpReq)
        .routeId("APD.IS_1.REQ")
        .startupOrder(startupOrder++)
        .process(LogUtils.LOG_START)
        .setHeader(HROUTE_ID, simple("${routeId}"))
        .setHeader("routeId", simple("${routeId}"))
        .setHeader(HSYS_ID_NEW, simple("${camelId}"))
        .setHeader("HSender", constant("IS_1"))
        .setHeader("HReceiver", constant("IS_2"))
        .setHeader("HOrganization", constant("TEST"))
        .choice()
          .when(simple("${in.header.test} == 'TEST1'"))
            .setHeader("JMSType", constant("TEST1"))
          .endChoice()
          .when(simple("${in.header.test} == 'TEST2'"))
            .setHeader("JMSType", constant("TEST2"))
          .endChoice()
          .when(simple("${in.header.test} == 'TEST3'"))
            .setHeader("JMSType", constant("TEST3"))
          .endChoice()
        .end()
        .to(BP_GET_BP)
        .to("direct:finish");
```

BRIDGE - Сервис передачи сообщений между брокерами ActiveMQ и Kafka

Сервис SRV.BRIDGE является промежуточным звеном при передаче сообщений между брокерами ActiveMQ и Kafka.

Сервис поддерживает два направления передачи:

1) От ActiveMq в Kafka. Сервис перенаправляет сообщение из брокера ActiveMq в топик брокера Kafka. Топик, в который должно быть отправлено сообщение и другие параметры маршрутизации настраиваются в описании Бизнес-процесса.

   Сервис слушает очередь ActiveMq указанную в конфигурации. Необходимо учитывать, что в топике, в который отправляется сообщение, необходимо дополнительно дать права пользователя сервиса Bridge (smx-bridge) на запись.

2) От Kafka в ActiveMq. Сервис подписывается на топик Kafka и имеет возможность транслировать сообщение из ленты в одну или несколько очередей ActiveMq исходя из настроек маршрутизации, указанных в описании Бизнес-процесса.

   Идентификатор Бизнес-процесса может быть назначен заранее, при помещении сообщения в топик, либо определен исходя из дополнительных параметров маршрутизации. При этом сообщение может быть задублировано и отправлено сразу по нескольким разным Бизнес процессам.

   Дополнительные параметры маршрутизации:

   Исходный тип сообщения определяется по заголовку HEventType. Если он не заполнен, то в качестве «исходного» типа принимается наименование топика Kafka.

   Значение заголовков, которые ожидаются сервисом в топике Kafka:

   - **HEventId** идентификатор сообщения. При его отсутствии сервис генерирует новый UUID. Значение транслируется в HMessageID
   - **HEventMainId** - идентификатор запроса (если данное сообщение является ответом). Значение при наличии транслируется в HCorrelationMsgID. Дополнительно заполняется заголовок HMainID (значением HCorrelationMsgID или HMessageID)
   - **HEventType** – «исходный» тип сообщения. При отсутствии исходным типом считается имя топика. Может быть переопределен в конечный тип / типы доп. настройками. Конечный тип транслируется в JMSType
   - **HPublisher** - источник сообщения. Значение транслируется в HSender. При отсутствии заполняется значением по умолчанию в зависимости от описания исходного топика в конфигурации сервиса:

     Сервис подписывается на топики, указанные в параметрах конфигурации вида:

     SENDER1.**from**=topic1
     SENDER2.**from**=topic2,topic3
     При этом активные подписки должны быть перечислены в параметре publishers:
     publishers=SENDER1,SENDER2
     При такой конфигурации для сообщения из топика topic1 будет заполнен заголовок HPublisher (только при отсутствии) значением SENDER1, а для топиков topic2 и topic3 - значением SENDER2

Необходимо учитывать, что при добавлении нового топика в конфигурацию нужно дополнительно дать права пользователю сервиса bridge (пользователь smx-bridge) на чтение из этого топика (право READ)

- **HOrganization** - ожидается переданное значение, а при его отсутствии HOrganization заполняется значением "NULL" и дополнительно переопределяется настройками SYS_PROPERTIES
- **HReceiver** - значение по умолчанию SRV.BRIDGE

«Исходный тип» транслируется в стандартный заголовок JMSType. Дополнительно для каждого JMSType можно настроить соответствующие ему заголовки HSender, HReceiver, HOrganization, которые в совокупности дают возможность определить уникальный идентификатор Бизнес-процесса в таблице маршрутизации путем вызова метода getBp сервиса bpRouter.

Дополнительные настройки хранятся в таблице базы данных SYS_Properties.

Описание настроек таблицы SYS_PROPERTIES

Поле PROPERTY_NAME	Описание	Правило заполнения имени параметра в поле	Описание заполнения других полей таблиц	
EMPTY.ИМЯ	Для обнуления бизнес-процесса (заголовка HBPID) если он по какой-то причине уже был заполнен ранее, но требуется маршрутизировать сообщение иначе	**EMPTY**. - обязательна часть ИМЯ - исходный бизнес-процесс (заголовка HBPID).Настройка специфическая, используется только при необходимости обнуления БП	PROPERTY_VALUE	флаг значения 1 - используется
			DESCRIPTION	флаг состояния: **1 - настройка включена**, иначе не будет применена
MAP.ИМЯ	Для соответствия исходного и конечных типов сообщения	**MAP.** - обязательна часть ИМЯ - Исходный тип сообщения, для которого применяется эта настройка строка для ИМЯ может отсутствовать: в этом случае считается что Конечный тип один и = Исходный тип	PROPERTY_VALUE	содержит один или несколько конечных типов, соответствующих исходному Формат: ИМЯ1,ИМЯ2
			DESCRIPTION	флаг состояния: **1 - настройка включена**, иначе не будет применена
ROUTE.ИМЯ1:ОРГАНИЗАЦИЯ	Для заполнения специфичных заголовков конечного типа	**ROUTE.** - обязательна часть ИМЯ1 - Конечный тип сообщения, для которого применяется эта настройка ОРГАНИЗАЦИЯ - значение заголовка	PROPERTY_VALUE	список заголовков в формате ИмяЗаголовка1=Значение1,ИмяЗаголовка2=Значение2 будут добавлены в сообщение / перекроют текущие

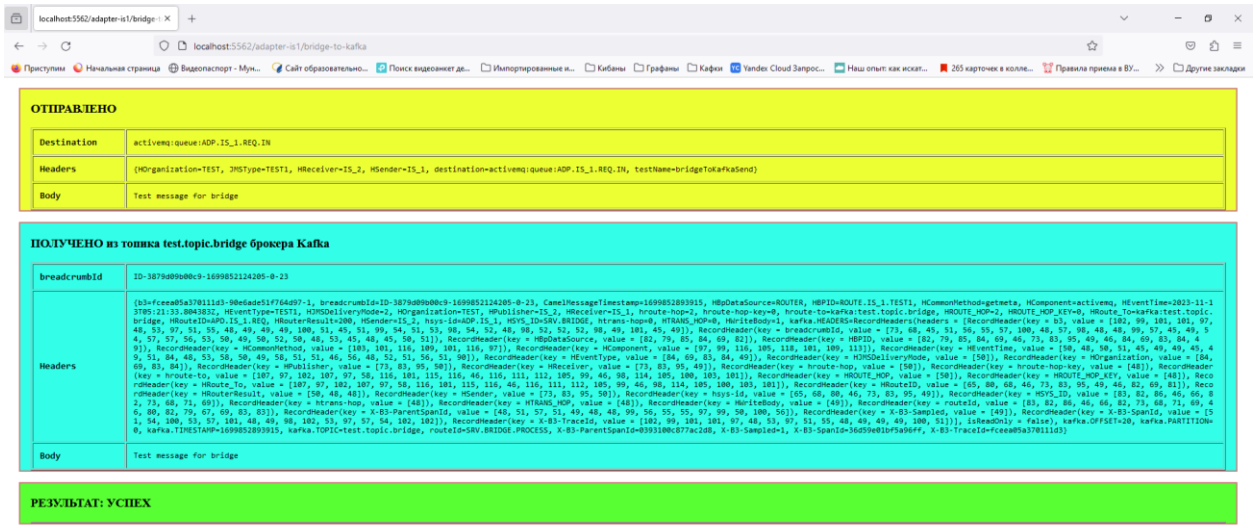
		HOrganization. а если заголовок отсутствует. то используется значение NULL, т.е: **ROUTE.**ИМЯ1:**NULL** строка для ИМЯ1 может отсутствовать: в этом случае используются значения заголовков по умолчанию		значения, если такие заголовки уже есть
			DESCRIPTION	флаг состояния: **1 - настройка включена**, иначе не будет применена

Пример

Пример 1. Доставка сообщения от брокера AMQ в топик Kafka.

Демонстрация примера представлена на странице браузера http://localhost:5562/adapter-is1/bridge-to-kafka

В указанном примере сообщение доставляется из очереди брокера AMQ в топик kafka.



Приложение 1. Описание таблиц базы данных

Таблица BP_DESCR – справочник бизнес-процессов

Поле	**Тип данных**	**Примечание**
bp_id	Varchar (50)	Идентификатор бизнес-процесса
Descr	Varchar (200)	Описание бизнес-процесса
Stream	Varchar (20)	Наименование продукта, в котором применяется БП
Owner	Varchar (100)	Владелец процесса
Developer	Varchar (100)	Разработчик
Owner_tel	Varchar (50)	Телефон владельца процесса
Developer_tel	Varchar (50)	Телефон разработчика

Таблица BUSINESS_PROCESSES – таблица параметров маршрутизации

Поле	**Тип данных**	**Примечание**
Bp_id	Varchar (100)	Идентификатор бизнес-процесса
Sys_id	Varchar (50)	Идентификатор программной сущности
Route_hop	Varchar (50)	Номер этапа, на который будет проведена маршрутизация
Organization	Varchar (50)	Идентификатор организации
Jms_type	Varchar (100)	Идентификатор сообщения
Sender	Varchar (100)	Отправитель
Receiver	Varchar (100)	Получатель
Work_flag	Bpchar (1)	Флаг работы системы
Descr	Varchar (200)	Наименование этапа
Key1	Varchar (200)	Ключ 1
Key2	Varchar (200)	Ключ 2
Key3	Varchar (200)	Ключ 3
Trans_hop	Varchar (50)	Номер этапа
Route_to	Varchar (200)	Наименование очереди, в которую будет маршрутизировано сообщение
Fault_route_to	Varchar (200)	Наименование очереди, в которую будет маршрутизировано ошибочное сообщение
Fault_hop	Varchar (50)	Номер этапа ошибочного сообщения
Version	Varchar (100)	Версия
Write_body	Varchar (20)	Логирование сообщения
Priority	Varchar (2)	Приоритет
Route_hop_key	Varchar (20)	Номер этапа, куда будет перенаправлено сообщение
Update_timestamp	timestamp (29)	Дата и время изменения
Persistent	Varchar (2)	Признак персистентности

Таблица SYS_DESCR – справочник программных сущностей

Поле	**Тип данных**	**Примечание**
Sys_id	Varchar (50)	Идентификатор программной сущности
Descr	Varchar (200)	Описание программной сущности
Owner	Varchar (100)	Владелец продукта
Developer	Varchar (100)	Разработчик
Owner_tel	Varchar (50)	Телефон владельца продукта
Developer_tel	Varchar (50)	Телефон разработчика