РУКОВОДСТВО ПО ЭКСПЛУАТАЦИИ

компонента Архивирование (код: ARCH)

продукта Platform V Archiving (код: ARC)

Содержание

Руководство по системному администрированию 5
Термины и сокращения 5
Сценарии администрирования 9
Настройка инициализирующей выгрузки данных 9
Оптимизация параметров тракта 13
Масштабирование сервиса 15
Отказоустойчивость 17
События системного журнала 20
Типовые рекомендуемые параметры логирования 21
Расширенные параметры логирования 21
Отладочное логирование 21
Полный перечень модулей и общих категорий Archiving 22
Устранение избыточной информации в логах 23
События мониторинга 24
Базовые метрики (все модули и потоки) 24
Метрики offline ТКД и DRP 25
Init в любом модуле (универсальные модули Init или модули конкретных источников) 26
Часто встречающиеся проблемы и пути их устранения 28
Работа с инициализирующей выгрузкой 28
Работа с ТКД 34
Ошибки обработки журналов со стороны ЦС 35
Руководство прикладного разработчика 38
Термины и сокращения 38

Системные требования 45
Подключение и конфигурирование 46
Требования для интеграции системы-источника в рамках целевой DevOps схемы 46
Общее описание DevOps-этапов для интеграции системы-источника (Сборка и ИФТ) 46
Описание DevOps-этапов для фазы тестирования 47
Состав дистрибутива конфигурации системы-источника 48
B-Pipeline. Сборка конфигурации (конфигурационного архива) для Archiving 54
Подключение JSL сборки конфигурации Archiving 57
R-Pipeline. ИФТ. Валидация дистрибутива системы-источника 60
Подключение JSL валидации дистрибутива конфигурации Archiving 61
R-Pipeline. ИФТ. Фаза конфигурирования экземпляра Archiving 64
R-Pipeline. ИФТ. Smoke-Regress тесты 66
Работа с клиентской библиотекой Archiving 67
API для offline ТКД с построением семпла на стороне системы-источника 72
API для online ТКД 73
API системы-источника для offline ТКД 74
API offline ТКД 86
Артефакты API 91
Maven артефакты DTO, MMT API интерфейсов и вспомогательных библиотек 92
Протокол offline ТКД 92
Транспортный контейнер 94
Сбор метрик с использованием возможностей библиотеки data-transport-api 95
Миграция на текущую версию 103
Быстрый старт 104

Стадия сборки дистрибутива 104
Стадия развертывания дистрибутива 104
Сборка конфигурации (конфигурационного архива) для Archiving 105
Подключение JSL сборки конфигурации Archiving 106
Валидация дистрибутива системы-источника 107
Фаза конфигурирования экземпляра Archiving 107
Smoke-Regress тесты 108
Настройка DPM (Platform V DevOps Pipeline Management) 108
API для offline ТКД с построением семпла на стороне системы-источника 109
API для online ТКД 110
Часто встречающиеся проблемы и пути их устранения 111

Руководство по системному администрированию

Термины и сокращения

Термин/сокращение	Расшифровка	Определение
Archiving	-	Platform V Archiving (ARC)
Init	-	Инициализирующая выгрузка данных
ТКД	Технический контроль данных	Механизм, использующийся для проверки качества загруженных данных
ТС	Технологический сервис	Сервис Platform V, выполняющий выделенный набор задач, общих для системы-источника
ЦС	Целевая система	Клиентский по отношению Archiving сервис, поставляющий свои объекты в векторах изменений для их репликации
MMTP	Межмодульный транспорт	Межмодульный транспорт (MMTP) в составе Продукта Platform V Synapse Enterprise Integration (SEI)
DRP	Disaster Recovery Plan	Восстановление данных
HBase	-	СУБД класса NoSQL с открытым исходным кодом, проект экосистемы Hadoop
Hadoop	-	Свободно распространяемый набор утилит, библиотек и фреймворков для разработки и выполнения распределённых программ
DTO	Data Transfer Object	Один из шаблонов проектирования, используется для передачи данных между подсистемами приложения
Stack trace	-	Последовательность вызовов и состояние окружения в некоторой точке программы

Термин/сокращение	Расшифровка	Определение
УЗ	Учетная запись	Совокупность данных о пользователе, необходимая для его опознавания (аутентификации) и предоставления доступа к данным и настройкам
ТУЗ	Технологическая учетная запись	Неперсонифицированная, обезличенная учетная запись, от имени которой осуществляется доступ к определенным информационным ресурсам, например, запускаются те или иные приложения и сервисы
Платформа	-	Многокомпонентное решение, включающее набор сервисов, предназначенных для создания бизнес-приложений. Набор продуктов Platform V, правообладателем которых является АО «СберТех». Перечень таких продуктов обозначен в документации на конкретный Продукт.
Контур	-	Набор программных и технических средств для обеспечения определенной функциональности и выделенный в отдельную группу по определенному признаку. Например, выделяют основной и резервный контуры системы: основной — для выполнения заявленных функциональных возможностей, резервный — с дублирующими основной функциями, но использующийся при недоступности основного контура

Термин/сокращение	Расшифровка	Определение
Stand-In, SI	-	Контур с резервной базой данных. Обычно полная или почти полная копия основного контура платформы, позволяющая за минимальное время переключиться на другой контур, сохраняя работоспособность системы
Платформенный Stand-In	-	Семафор, относящийся к платформенному StandIn
Семафоры	-	Фиксированный набор состояний, в которых может пребывать система при переходе в функциональный StandIn
АРМ	Автоматизированное рабочее место	Консоль управления приложением
UI	User Interface	Интерфейс пользователя
Retry	-	Повторная попытка чтения данных из источника
Writer	-	Процесс, пишущий данные куда-либо (например, в базу Kafka)
Агрегат	-	Объединенный объект или сообщение, содержащий сведения из корневого объекта и его дочерних объектов
Зона ММТ	-	Применяется для логического разделения пространств. ММТ внутри своей логики разделяет целевое решение на зоны для маршрутизации потоков информации и балансировки нагрузки. Работа одного кластера Kafka с несколькими зонами целевого решения позволяет эффективно использовать аппаратные ресурсы серверов Kafka.

Термин/сокращение	**Расшифровка**	**Определение**
Тракт	-	Весь конвейер прохождения данных от источника до точки постоянного хранения целевой системы
Пачка	-	Пакет данных
Поток	-	Потоковая выгрузка данных (или потоковая обработка векторов изменений)
data-fabric-replication	-	Библиотека, поставляемая в составе Platform V Persistence, имеющая в своем составе динамически подключаемые библиотеки Guava и KRYO
KRYO	-	Библиотека EsotericSoftware/kryo**, преобразующая** Java**-объекты в собственный бинарный формат.**
Guava	-	Набор open-source библиотек для Java, содержащих новые типы объектов, неизменяемые коллекции, графы и утилиты для параллельной обработки данных, ввода-вывода, хеширования, кеширования, примитивов, строк

Сценарии администрирования

Platform V Archiving (ARC) не имеет собственного пользовательского интерфейса, администрирование осуществляется через:

- настройки окружения сервиса;
- параметры сервиса, расположенные в системах единой конфигурации;
- параметры сервиса, расположенные в UI смежных сервисов, которые применяются для использования функций Platform V Archiving.

Настройка инициализирующей выгрузки данных

Init нужен для первоначальной загрузки данных из системы-источника в ЦС.

Режимы работы загрузчика в ЦС

Основные режимы работы загрузчика ЦС

Основные режимы работы загрузчика ЦС:

- Default (используется для обратной совместимости)— при такой конфигурации online ТКД не работает. Если пришел объект, версия которого ниже, чем в Hbase, объект игнорируется (не сохраняется в Hbase, не возникает инцидент. Верно для любого режима), происходит переход к сверке следующего объекта. Сверка версий происходит в версии writer с online ТКД.
- Online ТКД (без полноты) — версии объекта анализируются и, в зависимости от сценария, объект либо сохраняется в Hbase в случае успеха, либо в специальный топик ТКД на стороне Archiving отправляется запрос полной версии объекта. В случае аварийного сценария прием данных контейнером, в котором возник данный сценарий, приостанавливается до момента получения ответа на аварийный запрос из Archiving.
- Retry (waitcommit) (с полнотой) — writer не запрашивает ТКД синхронно, а вместо этого помечает сообщение в технологической таблице как «поломанное» и ожидающее retry, откатывает изменения по нему, отправляет сообщение в топик retry и продолжает работать. Если через 15 минут не будет получен правильный вектор, загрузчик прекращает работу.

Примечание:

Platform V DataSpace работает только в режиме Retry, остальные источники могут использовать и другие режимы:

1. Platform V DataSpace writer сверяет версию дочернего объекта с текущей версией агрегата.

2. Platform V DataSpace writer всегда обновляет версию агрегата, даже если самого корневого объекта не было в контейнере.

3. Platform V DataSpace writer не запрашивает ТКД синхронно, а вместо этого помечает сообщение как «поломанное», откатывает изменения по нему, отправляет сообщение в топик retry и продолжает работать.

Параметры работы загрузчика ЦС:

• Strict/ Non-Strict — используется только в ТКД и только для источников, которые не хранят версию. Strict означает, что приходящий вектор изменений должен иметь версию «+1» к текущей. Strict неприменимо к источникам, использующим любые немонотонно растущие последовательности в качестве версии.

• Полнота (waitcommit) может быть включена для любого writer, это обычная отправка коммитов. Требуется согласованное включение как для ЦС, так и для источника, и возможно не для всех источников.

• Online ТКД используется вне зависимости от того, какой формат версии используется источником. Пропуск версии важен только для Дельты (Векторов изменений).

Типы векторов изменений:

• Дельта-сообщение — изменение некоторых полей в источнике. В определенном поле указан список измененных полей.

• Снепшоты (snapshots) — полное состояние объекта. Для них список значений полей (значение поля) будет `NULL`.

ЦС по этому полю распознает тип вектора.

Поле `тип операции` может быть нулевым (`NULL`). В этом случае, если тип операции `NULL`, ЦС трактует операцию как INSERT. Это работает в случае полного состояния объекта - все поля со всеми ограничениями, требуемые для операции INSERT.

Типы источников, с которыми работает Archiving:

• Источники, которые отправляют как Дельта-сообщения, так и снепшоты, Для источников, отправляющих Дельта-сообщения, может использоваться только режим Strict.

• Источники, которые всегда отправляют снепшоты. Такие источники не передают список полей для обновления. В этом случае может использоваться режим Non-Strict для загрузчика.

Снепшот — это полный элемент. В этом случае для любой системы в любом режиме нет проблемы с пропуском версий или снепшотом с версией ниже текущей: ошибка в Archiving не отправляется. Если версия ниже текущей, сообщение об ошибке записывается в лог ЦС, и сообщение игнорируется.

Переиспользование ID означает ошибки на источнике.

В случае включенной полноты пустые транспортные контейнеры не отбрасываются. В них отправляются служебные пакеты. Загрузчик также должен работать в режиме включенной полноты. Загрузчики в случае не включенной полноты отбрасывают пустые контейнеры. В этом случае в ЦС ничего не отправляется, вектор изменений отображается желтым (Warning).

Реакция ЦС в зависимости от приходящего события и состояния объекта в ЦС

Вектор изменений

Комментарии по таблице:

- `Insert` — операция добавления объекта в ЦС;
- `Update` — операция обновления объекта в ЦС;
- `Delete` — операция удаления объекта из ЦС;
- пересечение столбцов — выполняемая операция:
  - `Apply` — применение операции;
  - `Ignore` — игнорирование операции;
  - `Error` — генерирование ошибки выполнения операции.

	Insert				**Update**				**Delete**			
Отношение номера версии текущего и пришедшего вектора	<	=	+1	>+1	<	=	+1	>+1	<	=	+1	>+1
Объект, по которому пришло событие, не существует в ЦС	Apply				Error				Error			
Объект, по которому пришло событие, существует в ЦС	Ignore	Ignore	Apply	Apply	Ignore	Ignore	Apply	Error	Ignore	Apply	Apply	Error
Объект, по которому пришло событие, удален из ЦС	Ignore	Ignore	Apply	Apply	Ignore	Ignore	Error	Error	Ignore	Ignore	Error	Error

Снапшот

Комментарии по таблице:

- `Insert` — операция добавления объекта в ЦС;
- `Update` — операция обновления объекта в ЦС;
- `Delete` — операция удаления объекта из ЦС;
- пересечение столбцов — выполняемая операция:
  - `Apply` — применение операции;
  - `Ignore` — игнорирование операции;
  - `Error` — генерирование ошибки выполнения операции.

	Insert				**Update**				**Delete**			
Отношение номера версии текущего и пришедшего вектора	<	=	+1	>+1	<	=	+1	>+1	<	=	+1	>+1
Объект, по которому пришло событие, не существует в ЦС	Apply				?				Error			
Объект, по которому пришло событие, существует в ЦС	Ignore	Ignore	Apply	Apply	Ignore	Ignore	Apply	Apply	Ignore	Apply	Apply	Error
Объект, по которому пришло событие, удален из ЦС	Ignore	Ignore	Apply	Apply	Ignore	Ignore	Error	Error	Ignore	Ignore	Error	Error

Оптимизация параметров тракта

Настройка производительности возможна по каждому из направлений работы Archiving (Init, Поток, ТКД). Все три направления будут рассмотрены по отдельности.

Инициализирующая выгрузка (Init)

Важные параметры конфигурации:

• **Размер пакета** — количество партиций, который обрабатывает Worker за одну итерацию. Не рекомендуется увеличивать, так как вместе с этим увеличивается время, которое требуется обработчику для отправки статистики о количестве обработанных партиций. Партиции обрабатываются последовательно, при этом размер отдельного объекта влияния не оказывает.

- **Количество обработчиков** — в среднем сервер Archiving нормально работает с четырьмя потоками. Исходя из этого, количество обработчиков можно установить равным 4 * [количество стендов]. Параметр можно увеличивать, но кратно количеству серверов, чтобы нагрузка распределялась равномерно, так как обработчики распределяются по стендам равномерно. Рабочий диапазон значений параметра: минимум (*2 узлов), максимум (*12 узлов). Без дополнительной настройки установить выше 10 потоков на узел невозможно.
- **rawBatchSize** — количество дата-контейнеров, которое Archiving упаковывает в один необработаный контейнер. Чем больше — тем выше скорость. Например, если параметр равен 10 (по умолчанию), тогда из 3000 дата контейнеров, полученных в партиции, в топик необработанных данных будет отправлено 300 необработанных контейнеров. Далее каждый контейнер необработанных данных вычитывается одним из КТС, откуда объекты извлекаются и десериализуются, фильтруются по белому списку и отправляются в ЦС. Ограничение параметра — только максимальный размер сообщения в топике Kafka (в текущей версии 16 МБ). Следовательно, [максимальный размер rawBatchSize] = 16 МБ / [максимальный вес вашего DataContainer]. Имеет смысл увеличивать до 100…500, особенно для мелких объектов.
- **partitionsThreshold** — параметр позволяет регулировать максимальное время ожидания Archiving пакетов данных по партиции источника. Значение по умолчанию — 5 минут, однако из-за размера партиции возможная неполная ее передач за указанное время. Размер партиции можно увеличить на стороне источника, параллельно увеличив время ожидания партиции на Archiving.

Партиционирование топиков

Рекомендуемые параметры:

- количество партиций в топике данных = числу узлов (`pprbod-source-provider-v4@kafka.topics.data.partitions`);
- количество партиций в топике Init = числу узлов (`pprbod-source-provider-v4@kafka.topics.init.partitions`)
- количество партиций в топике необработанных данных _raw = число узлов x*7 (`pprbod-source-provider-v4@kafka.topics.raw.partitions`);

- количество партиций в системных топиках _eventbus = числу узлов (`pprbod-source-provider-v4@kafka.topics.eventbus.partitions`);
- фактор репликации на все топики не более 2 (рекомендованное значение - 1, в этом случае производительность будет максимальной) (`pprbod-source-provider-v4@kafka.topics.*.replication=1`).

Поток (обработка векторов изменений от источника в ЦС)

Производительность настраивается партицированием топиков. Настройки те же, что и для Init. Другие параметры на производительность не влияют.

ТКД и DRP

Топик ответов на запросы offline ТКД и топик ответов восстановления данных (DRP) также может быть партицирован. Партиционирование используется при больших репликах данных. Партиционирование топиков требует согласования с ЦС.

Партиционирование топиков

Рекомендуемые параметры:

- количество партиций топика offline ТКД = количество узлов Archiving в решении (`pprbod-source-provider-v4@kafka.topics.offdq.partitions`);
- количествово партиций топика ответов offline ТКД = количество узлов Archiving в решении (`pprbod-source-provider-v4@kafka.topics.offdqresponse.partitions`);
- количество партиций топика ТКД = количество узлов Archiving в решении (`pprbod-source-provider-v4@kafka.topics.dq.partitions`);
- количество партиций топика ответов ТКД = количество узлов Archiving в решении (`pprbod-source-provider-v4@kafka.topics.dqresponse.partitions`);
- количество партиций топика ошибок = 1 (всегда) (`pprbod-source-provider-v4@kafka.topics.error.partitions`);
- количество партиций в системных топиках _eventbus = числу узлов (`pprbod-source-provider-v4@kafka.topics.eventbus.partitions`).

Масштабирование сервиса

Archiving является горизонтально масштабируемым сервисом, то есть для увеличения или уменьшения пропускной способности можно изменять число серверов, на которых установлены компоненты сервиса. При администрировании Archiving может быть

реализован сценарий, при выполнении которого необходимо увеличить или уменьшить количество узлов, на которых установлен сервис.

В архитектуре Archiving предусмотрена возможность работы сервиса с различным количеством узлов для увеличения или уменьшения предельного объема обрабатываемых данных в единицу времени в зависимости от потребностей. Минимальное допустимое количество узлов - 1. При первоначальной настройке сервиса (подробнее см. Руководство по установке) производится конфигурация определенных параметров Archiving в зависимости от того, сколько узлов предполагается использовать. В случае изменения этого количества, необходимо изменение значений тех же параметров.

Добавление нового узла

Добавление нового узла выполняется для каждого нового узла (первый узел формируется в процессе первичной установки).

Для ввода в решение нового узла Archiving:

4. Скопируйте сертификаты Kafka, выпущенные в рамках пункта «Создание клиентских сертификатов, выдача прав» Руководства по установке, на новый узел. Адрес каталога, где будут размещены сертификаты, должен совпадать с адресами каталогов на уже присутствующих узлах Archiving.

5. Скопируйте сертификаты Platform V DataGrid (IGN), выпущенные в рамках пункта «Выпуск сертификата Archiving для Platform V DataGrid» Руководства по установке. Адрес каталога, где будут размещены сертификаты, должен совпадать с адресами каталогов на уже присутствующих узлах Archiving.

6. Установите компоненты согласно параграфу «Установка модулей на WildFly» Руководства по установке.

Обновление конфигурации решения

Изменение максимального количества подключений к БД Archiving

БД Archiving имеет фиксированный пул подключений. Каждый новый узел требует дополнительного места для подключения.

7. Рассчитайте требуемый размер пула по формуле:

```
[количество серверов Archiving в решении] * jdbc.maxPoolSize + 10
```

, где значение `jdbc.maxPoolSize` — параметр единой системы конфигурации в артефакте pprbod-source-provider-v4 (по умолчанию 5). Число 10 добавляется в случае необходимости дополнительных подключений.

8. Рассчитанный размер пула установите в соответствии с новым количеством серверов Archiving.

Отказоустойчивость

Режимы работы и принцип переключения

Archiving обеспечивает бесшовную работу систем-источников при переходе между контурами Normal (основной) и Stand-In (резерв). Archiving использует шину Kafka на общем для Normal и SI контуров кластере. Режим работы Archiving после интеграции с семафорами исключает одновременную работу потребителей данных на основном и Stand-In-контурах.

Технически работа контура в Stand-In или Normal по своей логике ничем не отличается. Принципиально важно следующее: в момент времени активным может быть только один контур — SI или Normal.

Режимы работы Archiving после интеграции с семафорами:

- Контур активен, работает в стационарном режиме обрабатывает все векторы и контейнеры: Поток, Init, online/offline ТКД и DRP.
- Контур неактивен, векторы и контейнеры не обрабатываются.

Переключение режимов контуров:

Переход контура *из активного состояния в неактивное* (переход с Normal на SI или с SI на Normal) занимает около 40 секунд. При данном переходе последние вычитанные сообщения дообрабатываются и происходит остановка обработки данных.

Особенности работы в зависимости от типа взаимодействия Archiving и системы источника:

- Init (первоначальная выгрузка данных) — запущенные по ныне активному контуру завершатся ошибкой.
- Поток (обработка векторов изменений) — все взятые в обработку векторы будут обработаны. Происходит остановка обработки Потока в Archiving. Векторы, которые не

успели обработаться и остались на стороне Archiving и векторы, которые могли скопиться в системе обработки векторов, при активации другого контура будут обработаны уже им.

• Online ТКД — все взятые в обработку сообщения будут обработаны, далее происходит остановка обработки online ТКД.

• Offline ТКД и DRP — все взятые в обработку контейнеры с ключами будут обработаны. По ключам, по которым обработка прошла с ошибкой или которые еще не были обработаны, будет отправлена информация в топик ошибок с соответствующим статусом. Далее происходит остановка обработки offline ТКД и DRP, происходит закрытие сессии.

Переход контура из *неактивного состояния в активное* (переход с Normal на SI или с SI на Normal) занимает порядка 40 секунд.

При таком переходе происходит активация обработки данных.

Данные (векторы и контейнеры), которые не были обработаны предыдущим контуром дообрабатываются ныне активным с момента обработки последнего сообщения предыдущим контуром.

Типы взаимодействия Archiving и системы источника:

• Init (первоначальная выгрузка данных) — если завершились ошибкой, нужно перезапустить. Если ЦС поддерживает отправку с заданной партиции можно запустить Init с номером партиции, который нужно продолжить Init.

• Поток (обработка векторов изменений) — векторы, которые не успели обработаться предыдущим контуром, дообрабатываются. Происходит подписка на чтение журналов , идет обработка всех скопившихся журналов.

• Online ТКД — обработка ключей с момента обработки последнего сообщения предыдущим контуром если такие имеются.

• offline ТКД и DRP — обработка контейнеров с ключами, которые не обработал предыдущий контур.

Варианты администрирования

При работающем потоке

Включен online-Поток на контуре Normal. Если контур переходит в SI, то все накопленные векторы обработаются автоматически в контуре SI, никакие действия предпринимать не нужно.

При нахождении в SI и переходе в Normal, логика аналогична: все накопленные векторы при смене активности обработаются автоматически в Normal контуре.

При работающем Init

При запущенном процессе Init на Normal. Если контур переходит в SI, Init успешно успел завершиться, перезапуск не требуется.

Если Init завершается с ошибкой и требуется перезапуск Init, выполните повторный запуск с заданной партиции.

При работающем online ТКД

При запущенном процессе online ТКД — аналогично при работающем потоке (все накопленные сообщения обработает контур SI).

При работающем offline ТКД или DRP

Идет процесс offline ТКД/DRP. Контур переходит в SI.

Если по финальному отчету offline ТКД нет явных ошибок и пропусков по ключам, то перезапуск процесса не требуется. В противном случае если есть ошибки и/или пропуски, выполните Повторный запуск.

По процессу DRP: ЦС не формирует финальный отчет, поэтому нужно выяснить, был ли в это время переход в SI и если да и требуется перезапуск процесса DRP, выполните Повторный запуск.

Повторный запуск: по истечении таймаута (5-10 минут) и при необходимости ТКД/DRP, можно запустить находясь в Stand-In если работу в платформенном Stand-In для ТКД/DRP поддержит сам источник.

- При переходе контура с Normal на SI или с SI на Normal, процесс offline ТКД или DRP запускать не рекомендуется.
- При смене активности контуров в момент offline ТКД/DRP, данные процессы завершаются. По всем ключам, не взятым в работу, будут ошибки и пропуски на том контуре, на котором был запущен процесс.

События системного журнала

Логирование работы Archiving осуществляется средствами компонента Журналирование (LOGA) Platform V Monitor (OPM). Модули Archiving записывают действия в стандартный системный файл sysout с использованием стандартных Java-методов журналирования. Для записи используется библиотека log4j.

Для подключения компонента Журналирование (LOGA) необходимо добавить maven зависимость в pom.xml файл:

```
<dependency>
  <groupId>sbp.logger</groupId>
  <artifactId>logger-api</artifactId>
</dependency>
```

Существуют следующие основные типы (уровни) событий системного журнала:

- **Error** — ошибки, возникающие в процессе работы сервиса. По умолчанию выводятся в компонент «Журналирование OPM Platform V Monitor»;
- **Warn** — предупреждения, возникающие в процессе работы сервиса. По умолчанию выводятся в компонент «Журналирование Platform V Monitor»;
- **Info** — информационные сообщения, возникающие в процессе работы сервиса. По умолчанию выводятся в компонент «Журналирование Platform V Monitor»;
- **Debug** — отладочные сообщения (детали взаимодействия с внешними системами), возникающие в процессе работы сервиса. Не выводятся по умолчанию в компонент «Журналирование Platform V Monitor»;
- **Trace** — трассировочные сообщения (ход выполнения сложных участков сервиса — информация для анализа ошибок), возникающие в процессе работы сервиса. Не выводятся по умолчанию в компонент «Журналирование Platform V Monitor».

Конфигурация позволяет для каждого класса гибко настроить уровень ошибок и предупреждений.

Журналы можно направилять в:

- компонент Журналирование (LOGA);
- на сервер(ы) в консольные журналы;
- на сервер(ы) в роллинг файлы по модулям.

Типовые рекомендуемые параметры логирования

Отправляемые в компонент «Журналирование Platform V Monitor» логи приложения зависят от текущей конфигурации.

Для корректной работы журналирования необходимо настроить параметры при помощи log4j по таблице ниже.

Категории и уровни логирования, обеспечивающие оптимальный вывод.

Категория	Уровень	Описание
com.sbt.pprbod	WARN	Логирование Archiving
com.sbt.pprbod.offlinedq	INFO	Логирование offline ТКД
com.sbt.pprbod.bgp	INFO	Логирование Init
com.sbt.pprbod.common	INFO	Логирование общей библиотеки
com.sbt.pprbod.journal	INFO	Логирование Потока
org.apache.kafka.common.utils.Log	ERROR	Отключение избыточной информации Kafka
transport-ott-logging	OFF	Отключение избыточной информации OTT transport-ott-logging

Расширенные параметры логирования

Задаются при необходимости.

Категория	Уровень	Описание
com.sbt.pprbod.journal.aladdin	DEBUG	Логирование принятых журналов Потока

Отладочное логирование

Применяется при необходимости локализации проблемы. Включение отладочной категории приводит к большому количеству логов — поэтому нужно выполнять осознанно и отключать после локализации проблемы.

Категория	Уровень	Описание
com.sbt.pprbod	DEBUG	Расширенное логирование Archiving. Включается на стороне Archiving
com.sbt.core.transport	DEBUG	Расширенное логирование ММТ. Включается на стороне Archiving и на стороне системы-потребителя
pprbod_shaded.org.apache.kafka	DEBUG	Расширенное логирование Kafka. Включается на стороне Archiving

Категория	Уровень	Описание
com.sbt.bgp	DEBUG	Расширенное логирование Init. Включается на стороне Archiving
com.sbt.pprbod.exchange.rpc **com.sbt.pprbod.transport_kafka**	**DEBUG**	**Расширенное логирование клиентской библиотеки для Platform V DataSpace. В логируемую информацию входит информация о вызовах (полученных и отправленных) с распечаткой requestId. Включается на стороне системы-потребителя на стеке k8s или OpenShift (опционально)**

Полный перечень модулей и общих категорий Archiving

Используется для включения полной отладки по всем модулям на полигоне.

Модуль	Категория
aladdin	com.sbt.pprbod.journal.aladdin
arch-journal-common	com.sbt.pprbod.common
data-transport-api	com.sbt.pprbod.data
data-transport-bgp-client	com.sbt.pprbod.data
data-transport-metamodel	com.sbt.pprbod.serialization
pprbod-applied-journal-handler	com.sbt.pprbod.journal.transport.service.utils
pprbod-data-quality-processor	com.sbt.pprbod.quality
pprbod-descriptor-generator-plugin	com.sbt.pprbod.metamodel.generator
pprbod-eventbus-lib	com.sbt.pprbod.eventbus
pprbod-grid-lib-v4	com.sbt.pprbod.offdq
pprbod-handler-lib	com.sbt.pprbod.journal.transport.api.service.universal
pprbod-kafka-lib	com.sbt.pprbod.kafka
pprbod-offline-dq-collector-v4	com.sbt.pprbod.offlinedq.collector
pprbod-offline-dq-processor-v4	com.sbt.pprbod.offlinedq
pprbod-source-api	com.sbt.pprbod.dtoprovider
pprbod-source-handlers-lib	com.sbt.pprbod.dtoprovider
pprbod-source-plugin-api	com.sbt.pprbod.listeners
pprbod-source-provider-lib	com.sbt.pprbod.dtoprovider
pprbod-sources-artifacts (cdm-source-artifact consents-source-artifact gbk-source-artifact mapper-source-artifact)	com.sbt.pprbod.artifacts
pprbod-stream-processor	com.sbt.pprbod.raw
pprbod-transport-kafka-lib	com.sbt.pprbod
schema-generator-from-metamodel	com.sbt.pprbod.generator

Устранение избыточной информации в логах

На каждом контуре нужно индивидуально проверить, нет ли глобально включенных категорий, которые распространяются на все узлы и все модули и добавлять избыточную информацию в логи.

Ошибки авторизации zookeeper ММТ

```
16.05.2021 21:29:58,322 INFO \[stdout\] (transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-SendThread(kishkar547.ca.sbrf.ru:2181)) pprbod-offline-dq-collector-v4:\[transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-SendThread(kishkar547.ca.sbrf.ru:2181)\] 16.05.2021 21:29:58 INFO org.apache.zookeeper.ClientCnxn\$SendThread:876 - Socket connection established to kishkar547.ca.sbrf.ru/10.124.114.11:2181, initiating session

16.05.2021 21:29:58,323 INFO \[stdout\] (transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-EventThread) pprbod-offline-dq-collector-v4:\[transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-EventThread\] 16.05.2021 21:29:58 ERROR org.apache.curator.ConnectionState:288 - Authentication failed

16.05.2021 21:29:58,325 INFO \[stdout\] (transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-SendThread(kishkar547.ca.sbrf.ru:2181)) pprbod-offline-dq-collector-v4:\[transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-SendThread(kishkar547.ca.sbrf.ru:2181)\] 16.05.2021 21:29:58 INFO org.apache.zookeeper.ClientCnxn\$SendThread:1299 - Session establishment complete on server kishkar547.ca.sbrf.ru/10.124.114.11:2181, sessionid = 0x100245effed1fc8, negotiated timeout = 30000

16.05.2021 21:29:58,325 INFO \[stdout\] (transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-EventThread) pprbod-offline-dq-collector-v4:\[transport-\[21H26M40S9179\]-\[pprbod-offline-dq-collector-v4\]-EventThread\] 16.05.2021 21:29:58 INFO org.apache.curator.framework.state.ConnectionStateManager:228 - State change: CONNECTED
```

Чтобы отключить эти сообщения, добавьте параметр `-Dzookeeper.sasl.client=false` в параметры jvm.

События мониторинга

Archiving реализует метрики, которые отражают общие показатели работы сервиса и отдельных модулей Archiving в контексте определенного источника. Для передачи метрик конечной системе используется Kafka — метрики публикуются в специальном топике.

Все метрики всегда формируются в разрезе источника данных и, в зависимости от метрики, добавляются дополнительные разрезы, которые делятся на следующие типы:

- source - источник (мнемоника источника, эквивалентна наименованию);
- channel - модуль Archiving, соответствующий потоку данных (Init, online ТКД, offline ТКД, DRP);
- type - тип данных (класс Java);
- loadingId - уникальный идентификатор сеанса загрузки;
- ErrorClass - класс по классификатору ошибок.

Базовые метрики (все модули и потоки)

ID	Тип	Описание	Разрезы
arch-journal.async_process_object.count	counter	Количество переданных объектов за 15 с	channel, source, type
arch-journal.async_process_object.duration	stopwatch	Длительность отправки в ЦС	channel, source
arch-journal.async_process_object.frequency	counter	Transactions per second - количество контейнеров в секунду	channel, source
arch-journal.errors.count	counter	Количество ошибок репликации за 15 с по Init и Потоку	channel, source
arch-journal.timestamp_of_last_sent_journal.timestamp	gauge	Время последнего отправленного сообщения	channel, source
arch-journal.timestamp_of_last_sent_journal.commit_timestamp	gauge	Время последнего получения коммита от ЦС	channel, source

ID	Тип	Описание	Разрезы
arch-journal.duration_of_response_kafka.timestamp	stopwatch	Длительность ответа Kafka - показывает наличие (большое значение) или отсутствие (малое значение) проблем с Kafka	channel, source
arch-journal.error_sending.count	counter	Количество ошибочных коммитов от ЦС при получении журналов	channel, source

Метрики offline ТКД и DRP

Метрики формируют модули ТКД и DRP: pprbod-data-quality-processor, pprbod-offline-dq-collector-v4

ID	Тип	Описание	Разрезы
arch-journal.dq.one_request_key_count	gauge	Количество ключей в одном запросе ЦС	channel, source, type
arch-journal.dq.objects_requested	counter	Общее количество запрошенных объектов	channel, source, type
arch-journal.dq.objects_received	counter	Общее количество полученных объектов от источника	channel, source, type
arch-journal.dq.objects_sent	counter	Общее количество отправленных в ЦС объектов	channel, source, type
arch-journal.dq.execute_time_sync	stopwatch	Время выполнения запроса в ММТ и источнике (синхронный sender)	channel, source, type
arch-journal.dq.execute_time_async	stopwatch	Время выполнения запроса в ММТ и источнике (асинхронный sender)	channel, source, type
arch-journal.dq.full_processing_time	stopwatch	Время между получением запроса ЦС и отправкой ответа (контейнера) в ЦС, включая время на стороне источника	channel, source, type
arch-journal.dq.tsa_processing_time	stopwatch	Время между получением ответа источника и отправкой ответа (контейнера) в ЦС	channel, source, type
arch-journal.dq.kafka_time	stopwatch	Время записи одного data container в Kafka	channel, source, type

ID	Тип	Описание	Разрезы
arch-journal.dq.retry_sent_count	counter	Количество повторных попыток отправки запроса в сторону источника	channel, source, type
arch-journal.dq.error_catch_count	counter	Количество ошибок	channel, source, type, ErrorClass
arch-journal.dq.ts_error_count	counter	Количество ошибок `ts_error` транспортного слоя	channel, source, type
arch-journal.dq.object_not_found_count	counter	Количество ошибок `object_not_found`	channel, source, type

Примечание:

Метрики **arch-journal.dq.objects_received** и **arch-journal.async_process_object.count**, означающие количество запрошенных объектов от источника и количество отправленных контейнеров ЦС в разрезе ТКД, могут отличаться. Основной причиной является получение пустого списка объектов после фильтрации по исходным ключам (запрошенным ЦС) и неотправка в ЦС пустого контейнера. Также возможно получение пустого списка объектов после фильтрации по белому списку с аналогичным действием. Другая возможная причина - закрытие сессии ТКД по таймауту до обработки всех данных, полученных от источника, что может случиться только в результате сбоя или плохой работоспособности трактов.

Init в любом модуле (универсальные модули Init или модули конкретных источников)

ID	Тип	Описание	Разрезы
arch-journal.init.estimate_time	stopwatch	Время подготовки пачек	source, type, loadingId
arch-journal.init.batch_count	gauge	Количество подготовленных пачек при загрузке	source, type, loadingId
arch-journal.init.batch_error_count	counter	Число пачек с ошибками (при обработке которых была ошибка)	source, type, loadingId
arch-journal.init.items_count	counter	Количество успешно полученных элементов (сущностей)	source, type, loadingId

ID	Тип	Описание	Разрезы
arch-journal.init.error_count	counter	Количество ошибок обработки в ходе Init	source, type, loadingId, errorClass
arch-journal.init.batch_load_time	stopwatch	Время получения одной пачки	source, type, loadingId
arch-journal.init.data_container_accept_time	stopwatch	Время обработки одной пачки в Archiving	source, type, loadingId

Часто встречающиеся проблемы и пути их устранения

Использование функционала Archiving можно разделить на 3 категории: Init, Поток, ТКД. Поэтому типовые проблемы далее будут разделены на перечисленные группы.

Работа с инициализирующей выгрузкой

9. Init завершил работу в статусе `ERROR.No api com.sbt.pprbod.data.transport.init.InitDataSampleApi`.

Для систем-источников, использующих Platform V DataSpace (APT), данная ошибка может означать следующее:

– В зоне MMT MMT_ZONE отключен или отсутствует модуль Platform V DataSpace. Обратитесь в поддержку Platform V DataSpace.

– У модуля Platform V DataSpace отключено APIInitDataSampleApi, которое используется для работы с Archiving или установлен модуль версии, где оно не реализовано. Обратитесь в поддержку Platform V DataSpace и уточните наличие и работоспособность InitDataSampleApi в соответствующей MMT зоне.

– В параметрах Init указаны неверные параметры zone или module. Проверьте корректность значений параметров. Значение параметров: module - идентификатор модуля Platform V DataSpace (dataspace-cr-migration-service-module), zone - MMT зона, где расположен модуль системы-источника (на одну зону MMT приходится один модуль Platform V DataSpace.

Для систем-источников на Platform V Persistence (HBR) или DTO класса данная ошибка может означать следующее:

– Отсутствует связь с модулем системы-источника, где реализовано API для взаимодействия с Archiving (com.sbt.pprbod.data.transport.init.InitDataSampleApi). Модуль отключен или отсутствует на полигоне, либо на нем не реализовано InitDataSampleApi. Уточните статус модуля системы-источника на контуре, где запускается Init.

– В параметрах Init указаны неверные параметры zone или module. Проверьте корректность значений параметров. Значение параметров: module - идентификатор модуля системы-источника, zone - MMT зона, где расположен модуль системы-источника.

10. Init закончил работу в статусе `ERROR.Source '\...\' not found`.

Для всех систем-источников данная ошибка означает, что конфигурация для источника с мнемоникой (идентификатором), указанной в параметре `source` Init, отсутствует в базе данных Archiving. Возможные варианты:

– Значение параметра `source` в конфигурации Init не совпадает с мнемоникой (идентификатором) источника в конфигурации Archiving (указывается в параметре `name` в файле конфигурации `sourceDescription.yaml` при формировании дистрибутива. Проверьте их соответствие.

– Конфигурация источника не загружена в базу данных Archiving. Проверьте, что успешно завершена фаза конфигурации источника в БД Archiving в процессе DevOps.

11. После запуска Init ЦС получил больше/меньше записей, чем в БД.

Если Init был завершен успешно, но количество объектов, указанных в строке состояния, отличается от количества в БД - удостоверьтесь, что сверка происходит с нужным SI контуром.

12. Init системы-источника на Platform V Persistence закончил работу в статусе `ERROR.java.lang.NoClassDefFoundError: com/sbt/pprb/integration/hibernate/changes/transform/Normalizer`.

Пример текста ошибки:

```
Caused by: java.lang.NoClassDefFoundError:
com/sbt/pprb/integration/hibernate/changes/transform/Normalizer
        at
com.sbt.pprb.integration.datafabric.hibernate.ChangeVectorBuilder.toChangesVecto
r(ChangeVectorBuilder.java:43)
        at
com.sbt.pprb.integration.datafabric.hibernate.ChangeVectorBuilder.toChangesVecto
r(ChangeVectorBuilder.java:35)
        at
com.sbt.pprb.integration.datafabric.hibernate.HibernateDataProvider.readPartitio
n(HibernateDataProvider.java:150)
        at
com.sbt.pprb.integration.datafabric.InitDataSampleFunctions.loadBatchAsync(InitD
ataSampleFunctions.java:41)
```

```
        at
com.sbt.pprbod.data.utils.InitDataSampleService.loadBatchAsync(InitDataSampleSer
vice.java:139)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
a:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at
com.sbt.core.transport.commons.utils.CBLUtils.changeClassLoaderAndInvoke(CBLUtil
s.java:21)
        at
com.sbt.core.transport.server.TransportServiceExporter.invoke(TransportServiceEx
porter.java:377)
        at
com.sbt.core.transport.server.TransportServiceExporter.invokeApiMethod(Transport
ServiceExporter.java:201)
        at
com.sbt.core.transport.server.TransportServiceExporter.onMessage(TransportServic
eExporter.java:141)
        at
com.sbt.core.transport.rpc.impl.RpcExecutorImpl.handleMessageWithSourceMessageCh
annel(RpcExecutorImpl.java:464)
        at
com.sbt.core.transport.rpc.impl.RpcExecutorImpl.handleWithTrace(RpcExecutorImpl.
java:416)
        at
com.sbt.core.transport.rpc.impl.RpcExecutorImpl.handleMessage(RpcExecutorImpl.ja
va:384)
        at
com.sbt.core.transport.rpc.impl.RpcExecutorImpl.handle(RpcExecutorImpl.java:279)
        at sun.reflect.GeneratedMethodAccessor3932.invoke(Unknown Source)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
a:43)
```

```
        at java.lang.reflect.Method.invoke(Method.java:498)
        at
com.google.common.eventbus.Subscriber.invokeSubscriberMethod(Subscriber.java:87)
        at com.google.common.eventbus.Subscriber$1.run(Subscriber.java:72)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
        at
org.springframework.remoting.support.RemoteInvocationUtils.fillInClientStackTrac
eIfPossible(RemoteInvocationUtils.java:45)
        at
com.sbt.core.transport.utils.InvokeMethodUtils.createResultException(InvokeMetho
dUtils.java:59)
        at
com.sbt.core.transport.utils.InvokeMethodUtils.getResultObject(InvokeMethodUtils
.java:40)
        ... 15 common frames omitted
```

Данное исключение возникает у систем-источников, использующих Platform V Persistence, при использовании некоторых старых версий библиотек orm-tools и data-fabric-replication.

Установите версии библиотек: orm-tools и data-fabric-replication — любая актуальная версия в соответствии с платформенными зависимостями.

13. Задача закончила работу, но при этом выгрузка данных не происходит.

Это означает, что на шаге запроса количества партиций от Archiving к системе-источнику, система-источник вернула `0` или данные не были запрошены. В такой ситуации рекомендуется:

1. Проверить параметры Init:
   - убедитесь, что в параметрах Init указаны типы данных, объекты по которым присутствуют в БД источника (если таблицы пустые, то Init отработал правильно);

- проверьте имена параметров `type_X`. Они должны идти по порядку от `type_0` до `type_N-1`, где `N` - количество типов данных в Init. Если это не так - исправить.

2. Проверить, что при Init система-источник на запрос количества данных (`getBatchCount`) возращает количество данных больше 0. Если это не так, вернитесь к шагу 1.

14. Init завершил работу с ошибкой

```
Normalizer.\<init\>(Ljavax/persistence/EntityManagerFactory;)V
```

Текст исключения:

```
Caused by: com.sbt.pprbod.data.transport.exception.BatchLoadException: An error occurred during call 'loadBatchAsync(ru.sbt.bas.bpsr.domain.controlsheet.ControlSheet,0,d766bce2-05a0-4f13-9af9-bcf7b9ad69a7)' and result waiting
    at com.sbt.pprbod.data.sample.bgp.InitApplicationTask.loadBatch(InitApplicationTask.java:64)
    at com.sbt.pprbod.data.AbstractInitApplicationTask.process(AbstractInitApplicationTask.java:152)
    ... 11 more
Caused by: com.sbt.core.amqp.exceptions.TransportException: java.lang.NoSuchMethodError: com.sbt.pprb.integration.hibernate.changes.transform.Normalizer.<init>(Ljavax/persistence/EntityManagerFactory;)V
    at com.sbt.core.transport.utils.InvokeMethodUtils.getResultObject(InvokeMethodUtils.java:44)
    at com.sbt.core.transport.rpc.impl.AbstractRpcCall.recreateResult(AbstractRpcCall.java:411)
    at com.sbt.core.transport.rpc.impl.AsyncReply.createAsyncResult(AsyncReply.java:158)
    at com.sbt.core.transport.rpc.impl.AsyncReply.deliverResult(AsyncReply.java:114)
```

```
    at
com.sbt.core.transport.rpc.impl.RpcInvokerImpl.replyResult(RpcInvokerImpl.java:4
16)
    at
com.sbt.core.transport.rpc.impl.RpcInvokerImpl.processReplyMessage(RpcInvokerImp
l.java:345)
    at
com.sbt.core.transport.rpc.impl.RpcInvokerImpl.processReplyMessageWithTrace(RpcI
nvokerImpl.java:306)
    at
com.sbt.core.transport.rpc.impl.RpcInvokerImpl.handle(RpcInvokerImpl.java:278)
    at sun.reflect.GeneratedMethodAccessor927.invoke(Unknown Source)
    at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
a:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at
com.google.common.eventbus.Subscriber.invokeSubscriberMethod(Subscriber.java:87)
    at com.google.common.eventbus.Subscriber$1.run(Subscriber.java:72)
    ... 3 more
Caused by: java.lang.NoSuchMethodError:
com.sbt.pprb.integration.hibernate.changes.transform.Normalizer.<init>(Ljavax/pe
rsistence/EntityManagerFactory;)V
    at
com.sbt.pprb.integration.datafabric.hibernate.ChangeVectorBuilder.toChangesVecto
r(ChangeVectorBuilder.java:43)
    at
com.sbt.pprb.integration.datafabric.hibernate.ChangeVectorBuilder.toChangesVecto
r(ChangeVectorBuilder.java:35)
    at
com.sbt.pprb.integration.datafabric.hibernate.HibernateDataProvider.readPartitio
n(HibernateDataProvider.java:150)
    at
com.sbt.pprb.integration.datafabric.InitDataSampleFunctions.loadBatchAsync(InitD
ataSampleFunctions.java:41)
    at
com.sbt.pprbod.data.utils.InitDataSampleService.loadBatchAsync(InitDataSampleSer
```

```
vice.java:139)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    ...
```

При возникновении данной ошибки проверьте версии библиотек Platform V Persistence, используемых в модуле фабрики: Platform V Persistence, data-fabric-replication – любая актуальная версия в соответствии с платформенными зависимостями.

15. Init завершился с ошибкой `Not all objects has been replicated`.

При Init или offline ТКД может возникнуть ошибка, приведенная ниже.

Текст исключения:

```
Not all objects has been replicated! Last result:
ContainerSendingResult{objectsSent=0, objectsWarn=0, objectsError=7}. See
adapter logs for details at
com.sbt.pprbod.data.AbstractInitApplicationTask.process(AbstractInitApplicationT
ask.java:189) ... 11 more
```

Наиболее частая причина возникновения данной ошибки - несоответствие модели данных в системе-источнике и конфигурации Archiving.

Проверьте актуальность метамодели в базе данных конфигураций источников Archiving.

Работа с ТКД

16. Данные не попадают в топики

Пример текста ошибки:

```
In zone Z_APPEAL_1 no any instance of module customer-cases-bps:
com.sbt.core.amqp.exceptions.TransportNoRouteException: No api
[com.sbt.pprbod.data.transport.tkd.QualityDataTransportApi] services available
for route: [Z_APPEAL_1]-[*]-[customer-cases-bps] (zone-node-module).. List of
services violations: ZONE NODE MODULE PING ENABLED FILTER VIOLATIONS default
wf_str-vat-app1406 ucp-fd-sync true [Service is not supported interaction by
rest.] default wf_tkli-pprb3464 gbk-main true [Service is not supported
interaction by rest.] default wf_str-vat-app1407 ucp-fd-sync true [Service is
not supported interaction by rest.] default wf_tkli-pprb1750 ucp-consents-fd-
sync true [] default wf_tkli-pprb3468 gbk-policy-mngr true [] default wf_tklia-
pprb02003 gbk-integration true [] default wf_tkli-pprb3468 gbk-main true
```

```
[Service is not supported interaction by rest.] default wf_tklia-pprb02002 gbk-integration true []
```

Этапы анализа проблемы:

17. Убедитесь, что модуль источника запущен.

18. Проверьте, что зона модуля Kafka/MMT системы-источника совпадает с реально используемой зоной системы-источника по артефакту транспорт Kafka.

Ошибки обработки журналов со стороны ЦС

Ошибки, вызванные невозможностью применить data-container на стороне ЦС и возвращаемые в ответе коммита ЦС в топик коммитов.

Общий формат — строка, первая часть (до вертикальной черты) — код ошибки, вторая — информационные параметры.

Название	Код	Формат описания	Описание ошибки
Ошибка нарушения последовательности операций	`ERROR_WRONG_OPERATION_SEQUENCE`	`ERROR_WRONG_OPERATION_SEQUENCE |type[%s], key[%s], operation[%s], version[%s], hversion[%s], timestamp[%s]`	Ошибка нарушения последовательности получения (перепутан порядок операций удаление-вставка, удаление-изменение и так далее)

Название	Код	Формат описания	Описание ошибки
Неверная версия	ERROR_WRONG_VERSION	ERROR_WRONG_VERSION|type[%s], key[%s], operation[%s], version[%s], hversion[%s], timestamp[%s]	Ошибка возникает, если передана неверная версия (пропуск версии, понижение версии)
Неверный формат	ERROR_CONTAINER_DATA	ERROR_CONTAINER_DATA|type[%s], key[%s], operation[%s], version[%s], hversion[%s], timestamp[%s], description[%s]	Любое отклонение от утвержденного протоколом формата контейнера, включая ошибку разбора, несоответствие схеме и так далее

Нотация параметров:

- `type` — передаваемый Archiving тип;
- `key` — передаваемый в контейнере ключ;
- `operation` — тип операции из контейнера;
- `version` — версия из контейнера;
- `hversion` — версия HBase;
- `timestamp` — время возникновения ошибки;
- `description` — описание ошибки (только для ситуации неверного формата). В случае исключения параметр записывается исключение без stack trace.

Примечание:

Параметры могут отсутствовать или быть недоступны в точке возникновения ошибки - в таком случае они заполняются значением `[ ]`.

Параметры могут иметь значение `null` — в таком случае они заполняются как `["null"]`.

Пример разбора ошибки

19. Получено сообщение:

```
ERROR_CONTAINER_DATA|type[com.sbt.bm.ucp.consents.model.dictionary.UcpConsentOperatorType], key[UcpConsentOperatorType_432.1653997258660], operation[null], version[0], hversion[], timestamp[2022-05-31 14:41:31.141], description[Part of the data is absent.].
```

20. Согласно мнемонике, это ошибка со стороны ЦС (см. таблицу выше) - неверный формат. Далее необходимо проанализировать сообщение по полям (названия полей см. в таблице выше).

21. Итоговое сообщение об ошибке: `description[Part of the data is absent.]`.

Руководство прикладного разработчика

Термины и сокращения

Термин/сокращение	Расшифровка	Определение
БД	База данных	Массив данных, хранящихся в соответствии со схемой данных, манипулирование которыми осуществляют в соответствии с правилами средств моделирования данных
Archiving	-	Platform V Archiving (ARC)
DataSpace	-	Platform V DataSpace (APT)
Init	-	Инициализирующая (первоначальная) выгрузка данных
Kafka	-	Распределённый программный брокер сообщений с открытым исходным кодом
URL	Uniform Resource Locator	Стандарт унифицированных адресов и записи ссылок на объекты
УЗ	Учетная Запись	Совокупность данных о пользователе, необходимая для аутентификации и контроля предоставления доступов
ТУЗ	Технологическая Учетная Запись	Разновидность учетной записи, применяемая только в случаях, когда пользователем выступает автоматизированная система
Avro	-	Ориентированная на строки платформа удаленного вызова процедур и сериализации данных, разработанная в рамках проекта Apache Hadoop

Термин/сокращение	**Расшифровка**	**Определение**
DTO	Data Transfer Object	Один из шаблонов проектирования, используется для передачи данных между подсистемами приложения
ЦС	Целевая система	Система, в которую система-источник отправляет изменения с помощью Platform V Archiving
Семафоры	-	Индикаторы состояния, обеспечивающие синхронизацию работы процессов, определяющие их режим работы
СП	Сервер приложений	Программная платформа для эффективного исполнения процедур, на которых построены приложения
СП Wildfly	-	Сервер приложений Wildfly
Система-источник	-	Система, которая отправляет в целевую систему данные с помощью векторов изменений
Smoke-тесты	-	Минимальный набор тестов на явные ошибки
Regress-тесты	-	Тестирование уже протестированной программы, проводящееся после ее модификации
Stand-In	-	Контур с резервной базой данных и/или сервисами. Обычно полная или почти полная копия основного контура платформы, позволяющая за минимальное время переключиться на другой контур, сохраняя работоспособность системы

Термин/сокращение	**Расшифровка**	**Определение**
StandBy	-	Дублирующая система (БД, сервер, сервис и т.д.), включающаяся в работу в случае неполадок на основной (Primary) системе, переключение идет автоматически
IAM	Identity and Access Management	Система централизации и автоматизации управления учетными записями пользователей и правами доступа к информационным системам предприятия
Response data	-	Ответное сообщение с данными на запрос ТКД
ТКД	Технический контроль данных	Механизм, используемый для проверки качества загруженных в целевую систему данных
Pipeline	-	Поток, конвейер
Поток	-	Потоковая выгрузка данных (или потоковая обработка векторов изменений)
Дескриптор модели	-	Файловый артефакт, содержащий техническое описание классов источника с уровнем детализации, достаточным для построения в среде выполнения экземпляров классов определенной версии модели источника, пригодных для выполнения десериализации и последующей репликации объектов этих классов
B-Pipeline	-	Фазы Build Pipeline - стадии сборки дистрибутива источника
R-Pipeline	-	Фазы Release Pipeline - стадии релизного DPM конвейера источника

Термин/сокращение	Расшифровка	Определение
DPM	DevOps Pipeline Management	Продукт Platform V DevOps Pipeline Management (DPM)
ММТ	Межмодульный транспорт	Компонент Межмодульный транспорт (ММТР) Platform V Synapse Enterprise Integration (SEI)
API	Application Programming Interface	Описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой
ТЯ	Технологическое ядро	Набор различных функций, связанных между собой в процесс создания продукта
ЛМ	Логическая модель	Взаимосвязь понятий предметной области и ограничения на данные, которые она налагает
ИФТ	Интеграционное и Функциональное Тестирование	Одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе
ORM	Object-Relational Mapping	Метод, связывающий БД с концепциями объектно-ориентированных языков программирования
DevOps	Development/Operations	Методология автоматизации технологических процессов сборки, настройки и развёртывания программного обеспечения

Термин/сокращение	**Расшифровка**	**Определение**
QG	Quality Gates	Набор флагов, свидетельствующих об успешном или неуспешном прохождении дистрибутивом источника определенной стадии релизного конвейера
QGM	Quality Gates Manager	Менеджер управления Quality Gates
Shard, Sharding	-	Стратегия горизонтального масштабирования кластера, при которой части одной базы данных размещаются в разных зонах
gRPC	Remote Procedure Calls	Система удалённого вызова процедур (RPC) с открытым исходным кодом
Job	-	Шаблон проектирования, который содержит все необходимые данные для параллельного выполнения задач
Консюмер	-	Потребитель данных, который работает с событиями (считывает и использует)
Семплирование, Семпл	-	Способ обработки, при котором общая информация о данных собирается на основе некоторой части этих данных (семплов)
Пилот	-	Ограниченное развёртывание процесса
HttpBridge	-	Компонент Шлюз запросов UI (HTTP-bridge) (HTBR) Platform V API Management (API)
Jenkins	-	Средство автоматизации с открытым исходным кодом, которое используется для автоматизированного создания, тестирования и развертывания программного обеспечения

Термин/сокращение	Расшифровка	Определение
CSRF	Cross-Site Request Forgery	Межсайтовая подделка запроса
JSON	JavaScript Object Notation	Текстовый формат для представления структурированных данных, основанный на синтаксисе JavaScript
Вектор изменений	-	Вектор Изменения (ВИ) представляет собой фрагмент данных, описывающий изменения, сделанные транзакцией в текущей БД, и предназначенный для репликации в смежные системы
SHA256	Secure Hash Algorithm	Алгоритм хеширования
Зона ММТ	-	Логическое разделение пространств. ММТ внутри своей логики делится на зоны, для маршрутизации потоков информации и балансировки нагрузки
Метамодель	-	Обект, описывающий структуру и принципы действия другой модели
Apache Maven		Фреймворк автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (Project Object Model)
HTTP	HyperText Transfer Protocol	Протокол прикладного уровня передачи данных
Nexus (Public)	-	Платформа, обеспречавающиая процессы управления исходными данными и создания целевых артефактов
ift, psi, prom зоны	-	Обозначание окружения, относящегося к одной из стадий разработки решения с использованием Platform V Archiving

Термин/сокращение	Расшифровка	Определение
Перечень типов	-	Список типов, по которым выполняется процедура сверки системы-источника и реплики
Дата загрузки	-	Бизнес-дата, по которой производится сверка
Семпл идентификаторов	-	Перечень идентификаторов, сформированный системой-источником по одному конкретному запрошенному типу на запрошенную дату загрузки (loading_date). В семпл идентификаторов включаться должны системой-источником только объекты, существующие на эту дату
HBase	-	СУБД класса NoSQL с открытым исходным кодом, проект экосистемы Hadoop
Семпл системы-источника	-	Перечень объектов, сформированный системой-источником по семплу идентификаторов
KRYO	-	Библиотека EsotericSoftware/kryo, преобразующая Java-объекты в собственный бинарный формат.

Системные требования

Системные требования отражены в документе Руководство по установке, раздел «Системные требования».

Подключение и конфигурирование

Все сообщения протокола, использующие криптографический хеш, подразумевают использование алгоритма `sha256`. Если не оговорено иное - хеш-сумма рассчитывается от всего контейнера полностью.

Требования для интеграции системы-источника в рамках целевой DevOps схемы

- Система-источник должна использовать **один из перечисленных ниже вариантов** представления собственной модели данных:
    1. Java DTO классы.
    2. Работа с данными с использованием ORM библиотеки Platform V Persistence (HBR).
    3. Работа с данными посредством использования Platform V DataSpace.
- Система-источник должна иметь интеграцию с метамоделью, а именно:
    1. Иметь опубликованную ЛМ системы в метамодели.
    2. Вести разметку белых списков в разрезе типов данных и аттрибутов в метамодели.
- Система-источник должна иметь настроенный сборочный конвейер с возможностью добавлять в него внешние фазы в виде сторонних задач.
- Система-источник должна иметь настроенный релизный DPM конвейер с возможностью добавлять в него внешние этапы в виде требуемых задач.

Общее описание DevOps-этапов для интеграции системы-источника (Сборка и ИФТ)

Этапы разделяются на фазы, выполняемые как на стадии сборки дистрибутива, так и на стадии развертывания дистрибутива системы-источника через релизный DPM конвейер.

Стадия сборки дистрибутива включает в себя следующие этапы:

1. **Для систем-источников, использующих Java DTO классы** — подключение в проект плагина Archiving для создания дескриптора модели подключения Apache Maven плагина в проект системы-источника. Для таких систем-источников предусмотрена

возможность разметки физической модели системы-источника аннотациями для унификации работы Archiving и устранения необходимости кодирования данной разметки в коде Archiving.

2. **Для систем-источников, использующих иное представление модели данных, кроме DTO** — подключение плагина не требуется.

3. Подключение фазы B-Pipeline (сборка конфигурации (конфигурационного архива) для Archiving) для сбора от систем-источников и из метамодели и включения в свой собственный дистрибутив параметров и артефактов для Archiving — согласно требованиям к составу дистрибутива.

Стадия развертывания дистрибутива включает в себя этапы, обеспечивающие проверку дистрибутива, конфигурирование Archiving для системы-источника и выполнение smoke-regress тестов системы-источника. По результатам каждой фазы проставляются соответствующие Quality Gate.

Этапы:

1. R-Pipeline. ИФТ. Валидация дистрибутива системы-источника;
2. R-Pipeline. ИФТ. Фаза конфигурирования экземпляра Archiving;
3. R-Pipeline. ИФТ. Smoke-Regress тесты;
4. R-Pipeline. ИФТ. Настройка DPM.

Примечание

Прохождение этапов синхронизируется по QG-флагу.

Описание DevOps-этапов для фазы тестирования

На фазе тестирования выполняется только задача конфигурирования Archiving на целевом решении и выставляются соответствующие флаги DPM.

Со стороны Archiving — это фаза конфигурирования системы-источника (tsa_configure).

По итогу успешного выполнения фазы проставляется флаг **pprbod.uat = ready**, свидетельствующий о готовности Archiving на сервере тестирования к работе. Выполняется подготовка и публикация плагина, а также выставляется флаг **pprbod.uat = valid**, свидетельствующий о готовности системы-источника к проведению тестов на сервере.

Обязательным предусловием конфигурирования Archiving на целевом решении является наличие флагов pprbod.uat установленного в значение ok и pprbod.uat установленного в значение valid.

Со стороны Archiving — это фаза выполнения конфигурирования системы-источника (tsa_configure).

Состав дистрибутива конфигурации системы-источника

Дистрибутив системы-источника Archiving поставляется в виде zip-архива.

Для Archiving дистрибутив системы-источника включает артефакт с конфигурацией системы-источника. Этот файл содержит артефакты конфигурации, необходимые для корректной работы Archiving с версией модели данных системы-источника, соответствующей текущему выпускаемому дистрибутиву.

Структурно артефакты конфигурирования размещаются в папке `./other/model`. В нее помещаются:

- Архив, содержащий файлы конфигурации — это артефакт с конфигурацией системы-источника, передаваемой в Archiving.
- Avro—схемы модели системы-источника, сгенерированные на фазе сборки дистрибутива по актуальной версии модели данных, белому и черному спискам.

  Схем модели системы-источника две: полная (**model_schemas_full.txt**) и усеченная по черному и белому списку (**model_schemas.txt**), которая и является рабочей. Эти схемы генерируются на фазе B-Pipeline.
- Файл **sourceDescription.yml**, содержащий параметры системы-источника. Создается и предоставляется самой системой-источником в процессе сборки дистрибутива. Все параметры должны быть строкового типа, даже те, в которые входит число, например, не верно:

  dataSampleThreshold: 1000

  Верно:

  dataSampleThreshold: “1000”

  Параметрами системы-источника являются:

1. **name** - мнемонический идентификатор системы-источника в **верхнем регистре** (например, SYSTEM_NAME). Предпочтительно, чтобы мнемоника отражала в себе название системы-источника.

Внимание

Мнемоника должна быть уникальна внутри конфигурации Archiving.

Обязательным требованием является согласование мнемоники при первичной интеграции с Archiving. В процессе (после первичной интеграции) эта мнемоника более не меняется для системы-источника. Мнемоника системы-источника не может отличаться для различных контуров и решений, на которых представлен модуль системы-источника.

Примечание

Мнемоника не может содержать дефис (минус). При необходимости используйте знак нижнего подчеркивания (NAME_DATA_SOURCE). Максимальная длина — 25 символов.

Пример:

Не верно: **ABCDEF-12 ABCDEF_12**

Верно:

ABCDEF_12

2. **description** — человекочитаемое описание системы-источника.
3. **modelType** — тип модели данных, применяемых системой-источником (один из вариантов):

Тип системы-источника	**Передаваемое значение/modelType**
Системы-источники, генерирующие дескриптор модели на основе кастомных DTO-классов	CLASSES
Системы-источники на **Platform V Persistence** и **Platform V DataSpace**	MAPPER_METAMODEL

4. **whiteListMode** — режим работы черного и белого списков (один из вариантов: **ROOT_TYPES, ENABLED, DISABLED**),

где

Параметр	Описание
ENABLED	Самый жесткий уровень фильтрации. В белом списке должны присутствовать не только типы объектов, но и перечень их атрибутов
ROOT_TYPES	Средний уровень фильтрации. В белом списке должны присутствовать только типы объектов, все их атрибуты будут переданы, даже если атрибуты в списке не указаны
DISABLED	Передаются все объекты, которые приходят в адаптер

5. **verboseKey** — по умолчанию false (выключен). Включается в исключительных случаях.

Если включен (true) - то в каждой реплицируемой записи к идентификатору будет добавляться префикс именни класса.

Примеры:

1. если VerboseKey = false, ID записи будет выглядеть так: “11223344”;
2. если VerboseKey = true, ID будет выглядеть так: “com.className.11223344”.

6. **initSourceModuleId** (**DEPRECATED**) — идентификатор модуля для Init (например, moduleName-main).

Внимание

Параметр выводится из эксплуатации. Если у вас настроен DevOps с заполнением параметра, можно все оставить без изменений. Работоспособность не изменится. Новым системам-источникам заполнять параметр не нужно.

7. **dqModuleId** — идентификатор модуля, через который осуществляется ТКД и offline ТКД, например, moduleName-main.

Примечание

Для систем-источников, использующих для интеграции шину взаимодействия с системой-источником данных в качестве идентификатора модуля, используется мнемоника системы-источника, задаваемая в application.yml в параметре **pprbod.cloud.client.source.** т.е. в качестве ***dqModuleId*** нужно передавать ее.

1. **ajZones** — список зон сервиса приема векторов изменений:

4. **ift**;
5. **psi**;
6. **prom**.

Если параметры по целевому решению отсутствуют, то при установке на этом решении применяется значение базового **ajZones**, Archiving при работе будет вычитывать вектора изменений в этой зоне.

Если зона еще не существует, то нужно зарегистрировать ее в нужных контурах по шаблонам сервиса приема векторов изменений.

2. **ajDataTypes** — типы данных сервиса приема векторов изменений (перечисляются через точку с запятой). Представляет собой аналогичное множество:
   1. **ift**;
   2. **psi**;
   3. **prom**.

Если параметры по целевому решению отсутствуют, то при установке на этом решении применяется значение базового **ajDataTypes**. Archiving при работе будет вычитывать указанные типы данных в указанных зонах.

3. ***mmtZones*** - список зон ММТ (перечисляются через точку с запятой), представляет собой аналогичное множество:
   1. **ift**;
   2. **psi**;
   3. **prom**.

Примечание

Для систем-источников, использующих для интеграции шину взаимодействия с системой-источником данных в качестве идентификатора ММТ зоны нужно передавать то, что указано в параметре application.yml **pprbod.cloud.client.zone.**

Если у системы-источника в топологии существует одна sharding зона, в данном параметре и в параметре **pprbod.cloud.client.zone** указывается **default**. На данный момент Archiving не поддерживает более одной sharding зоны у системы-источника.

4. ***nexusArtifactId*** — artifactId системы-источника в Nexus. Нужен для простановки флагов QGM.

Пример: `*.setArtifact("***ci01072092_pprbod***")`.

Если параметры по целевому решению отсутствуют, то при установке на этом решении применяется значение базового параметра ***mmtZones***.

5. ***nexusGroupId*** — groupId системы-источника в Nexus. Нужен для простановки в пространстве QGM системы-источника флагов: ***pprbod_ift_ready*** и ***pprbod_ift_valid***.
6. ***offDQRequestSenderFactoryMode*** — режим работы offline-ТКД. Методика анализа расхождений offline ТКД (один из вариантов):
    1. ***SYNC_WITH_SAMPLES*** (синхронная передача данных с поддержкой сэмплов). Используют системы-источники на Platform V Persistence и системы-источники на универсальном векторе изменений или DTO классах, реализующих Синхронный API Init.
    2. ***KAFKA_WITH_SAMPLES*** (передача данных с поддержкой примеров, поверх Kafka). Используют системы-источники, взаимодействующие с Archiving через Kafka (без использования ММТ), использующие для интеграции с Archiving Platform V Persistence или Platform V Dataspace.
    3. ***SYNC*** (синхронная передача данных).
7. ***nonRootTypeHandlingMode*** — параметр, указывающий необходимость конвертации запрошенного типа в корневой (иными словами, отдает ли система-источник не только корневые, но и дочерние типы). В случае, когда система-источник отдает и рутовые, и дочерние типы, параметр устанавливается в значение **AS_IS**. Если же отдаваемые типы - только рутовые, то значение параметра устанавливается как **SEND_ROOT**.
8. ***initiator*** — эквивалентно содержимому поля initiator заголовка сервиса приема векторов изменений, то есть названию модуля, который отправляет данные в сервис приема векторов изменений.
9. **capWaitCommit — true** или **false -** необязательный параметр, ожидать ли подтверждение на доставку сообщений. По умолчанию значение **false**.
10. **deserializerName** — название класса реализации интерфейса модели плагина PprbodSourcePlugin.

Для систем-источников на Platform V Persistence **com.sbt.pprbod.artifacts.mapper.MapperPprbodSourcePlugin**.

Для систем-источников на универсальном векторе сервиса приема векторов изменений и Platform V DataSpace **com.sbt.pprbod.artifacts.mapper.MapperJsonPprbodSourcePlugin**.

Для остальных систем-источников значение параметра равно полному именни класса реализации интерфейса модели плагина PprbodSourcePlugin, например **com.sbt.pprbod.artifacts.gbk.GbkPprbodSourcePlugin**.

11. **modelProviderType** - тип провайдера модели определяется исходя из типа интеграции с Archiving:

    1. ***MAPPER*** - системы-источники, использующие Platform V Persistence или Platform V Dataspace;
    2. ***ANNOTATION*** - системы-источники, использующие собственную реализацию модели, размеченную аннотациями Archiving;
    3. ***CUSTOM*** - системы-источники, использующие собственную реализацию провайдера модели, интегрирующие её в Archiving через плагин.

Параметры для интеграции с метамоделью:

1. **componentCode** — код компонента, задается при генерации или загрузке модели для метамодели на стороне системы-источника, например, «APP712».

Параметр является обязательным, если не предоставлен идентификатор компонента.

Требования по работе с метамоделью

Логика вычисления:

Если в параметрах функции configArhive.create(**modelPath** …..) передается путь к файлу с метамоделью и в параметре **whiteListPath** передан путь к файлу белого списка («B-Pipeline. Сборка конфигурации (конфигурационного архива) для Archiving»), Archiving возьмет для обработки эти файлы. Если же в этих параметрах передано значение NULL, то по параметру **componentCode** Archiving запрашивает метамодель и белый список. Если запрос не вернет метамодель, методы Archiving не выполнятся и дальнейшая работа будет невозможна!

Если запрос успешен, то схемы AVRO будут генерироваться именно из полученной от метамодели информации.

1. **dataSampleThreshold** (опциональный) — для ТКД число объектов, которое надо запросить у системы-источника. Значение 0 выключено. При отсутствии параметра, по умолчанию берется “0”.

2. **initTopicMode** (опциональный) — обозначает то, в какой топик выгружать данные в ходе Init (топик данных или отдельный топик Init). Допустимые значения: **data, init**.

Если указано другое значение, фаза конфигурации выдаст ошибку.

Когда initTopicMode отсутствует в sourceDescription.yml передается значение по умолчанию: **data**.

B-Pipeline. Сборка конфигурации (конфигурационного архива) для Archiving

Внимание!

Сборка конфигурации предполагает дополнение сборочного архива системы-источника файлами, предназначенными для настройки Archiving.

Внутренний состав сборки никак не затрагивается, собирается командой системы-иисточника без изменения технологического процесса.

Функциональность по сборке конфигурации для Archiving реализована в виде Jenkins-библиотеки (далее по тексту — ***JSL***).

Назначение функциональности: сборка артефактов конфигурации Archiving, необходимых для конфигурирования экземпляра Archiving на целевых решениях.

Функциональность (в виде JSL) предназначена для встраивания в сборочный Pipeline системы-иисточника непосредственно после сборки артефактов самой системы-иисточника, после того, как:

- Артефакты системы-источника собраны.
- Плагином Archiving на этапе выполнения Maven-сборки создан файл дескриптора модели.

Внимание

Файл дескриптора модели генерируется плагином только для систем-источников, использующих в качестве DTO артефактов java-классы. Для систем-иисточников,

использующих Platform V Persistence или Avro, в качестве дескриптора должно передаваться XML-описание объектов Platform V Persistence, либо Avro-схема соответственно.

Функциональность сборки включает в себя:

- Работу с метамоделью, а именно:
    1. Получение из метамодели логической модели системы-источника.

ТУЗ для метамодели

Для работы методов Archiving с метамоделью и получения вашей актуальной модели и белого списка необходимо наличие ТУЗ у каждой системы-источника.

*Роль **ROLE_UDM_READER***

После получения ТУЗ добавьте ее в вашем пространстве Jenkins.Credentials с id: ***meta-prom-creds**. > **Внимание** > > Для систем-иисточников, использующих «коммунальные» подзадачи в пространстве Archiving, учетные данные уже определены и регистрировать собственную ТУЗ нет необходимости.

При вызовах функции **create()** допускается использование как *credentian id*, так и формы “**username:password**”.

Внимание

Внутренний ТУЗ в метамодели.

1. Получение из метамодели белых списков, размеченных системой-источником в design-time.
2. Сопоставление логической модели с физическим описанием данных (файлом дескриптора системы-источника).
3. Формирование файла белого списка в установленном для Archiving формате на основании разметки белого списка ЛМ из метамодели.

- Работу по подготовке (структурировании в требуемом для Archiving формате) данных, поставляемых системой-источником, и данных, полученных из ЛМ метамодели.
- Работу по генерации Avro — схем для Archiving.

Данные, поставляемые непосредственно системой-источником:

1. Файл дескриптора модели (физический). Файл может быть одного из трех вариантов:

1. Для систем-источников, использующих классические java-классы (DTO) — файл дескриптора модели в формате **JSON**, генерируемый плагином Archiving в фазе сборки . Является выходным артефактом плагина. Имя файла соответствует шаблону **.model.v..json**.
2. Для систем-источников, использующих Platform V Persistence — файл логической модели в формате **XML**. Генерируется плагином eip-metamodel-scanner-maven-plugin. Является ресурсом системы-источника.
3. Файл белого списка. Это текстовый файл, опциональный в целевом варианте, т.к. в целевом варианте белый список извлекается задачей из ЛМ метамодели. Обязательный для системы-источника, который не имеет интеграции с ЛМ метамодели. Шаблон имени файла: ***.whitelist.v.***, например, **cdm.whitelist.v.0.0.1**

Примечание

Передача ЛМ в виде файлов допустима для «старых» систем-источников в целях совместимости. Для всех вновь подключаемых систем-источников целевой вариант — выгрузка ЛМ из метамодели, выгрузку осуществит Archiving по указанному **componentCode** и списку версий моделей в параметрах вызова функции `create()`.

2. Набор параметров конфигурирования Archiving.

Archiving реализует работу с метамоделью — вариацией логики **resolved**.

Логика вычисления:

Если в параметрах функции configArchive.create(**modelPath** …..) передается путь к файлу с метамоделью и в параметре **whiteListPath** передан путь к файлу белого списка, Archiving возьмет для обработки эти файлы.

Если же в этих параметрах передан NULL, то по параметру **componentCode** Archiving запрашивает метамодель указанных в **modelVersions** версий и белый список.

Если запрос не вернет метамодель, методы Archiving не выполнятся и дальнейшая работа будет невозможна!

Если запрос успешен, то схемы AVRO будут генерироваться именно из полученной от метамодели информации.

Внимание

Для систем-источников пилота и первой волны требование не обязательно, Archiving обработает метамодель и белый список в виде файлов!

Подключение JSL сборки конфигурации Archiving

Для сборки дистрибутива системы-источника был разработан компонент библиотеки JSL.

Внимание

Для систем-источников, не интегрированных с метамоделью, необходимо, чтобы в рабочем пространстве сборочной задачи системы-источника присутствовали файлы дескриптора модели и белого списка.

Результатом работы компонента библиотеки является появление следующих файлов в рабочем пространстве сборочной Jenkins-задачи, в которой подключена библиотека:

1. архив конфигурации (**.model..zip**);
2. дескриптор версий **version.info.yml**;
3. Avro-схемы.

Эти файлы должны быть помещены в папку **/other/model** дистрибутива системы-источника.

Порядок подключения сборочной библиотеки

- Зарегистрируйте библиотеку в Jenkins (если это еще не сделано).

3. Добавьте вызов компонента библиотеки сборки архива

```
configArchive.create(sourceDescriptionPath, modelPath, whiteListPath,
artifactVersion, dataModelVersion , listVersion , LinkedHashMap
dataModelVersions, auditCredId = "SBT-SA-TSPI001A"  ,  metaCredId = "meta-prom-
creds" )
```

В данном случае Archiving будет работать с данными, запрошенными из метамодели ЛМ.

где:

1. **sourceDescriptionPath** — путь к файлу sourceDescription.yml в сборочной директории задачи, куда подключается компонент (пример: “${WORKSPACE}/tmp/sourceDescription.yml”);

2. **modelPath** — путь к файлу дескриптора модели в сборочной директории задачи, куда подключается компонент (пример: “${WORKSPACE}/tmp/ucp.model.v.0.0.1.json”). В случае интеграции системы-источника с метамоделью должно передаваться значение **null**;
3. **whiteListPath** — путь к файлу белого списка в сборочной директории задачи, куда подключается компонент (пример: “${WORKSPACE}/tmp/ucp.whitelist.v.0.0.2”). В случае интеграции системы-источника с метамоделью должно передаваться значение **null**;
4. **artifactVersion** — версия дистрибутива (Пример: 4.000.01).

Внимание

Это значение будет использоваться для выставления флага при дальнейших вызовах функций tsa_validate и tsa_configure.

5. **dataModelVersion** — устаревшая форма, версия метамодели в виде строки “X.X.X”.

Внимание

Данная форма используется только для обратной совместимости.

6. **dataModelVersions** — версия моделей в формате **LinkedHashMap**. Пример: **[“spr”:“4.8.3”,“base_model”:“0.1.2.3”]**, при этом параметр **modelVersion** нужно установить в значение **null**.
7. **whiteListVersion** — версия белого списка.
8. **auditCredId** — учетные данные для аудита событий, согласно требованиям безопасности. Учетные данные должны существовать в проектной области, из которой запускается ваша подзадача.

Если вы пользуетесь эталонными подзадачами Archiving, запускаемыми из пространства Archiving, то там сработает значение по умолчанию и параметр заполнять необязательно. Если вы используете методы Archiving в своих подзадачах в своем пространстве, то нужно определить этот параметр. Можно использовать те же учетные данные, под которыми вы обращаетесь в Nexus.

9. **metaCredId** — учетные данные для вычитывания метамодели. Учетные данные должны существовать в проектной области, из которой запускается ваша

подзадача. По умолчанию используются учетные данные “meta-prom-creds”. Они определены в проектной области Archiving, из которой запускаются “коммунальные” подзадачи. Вместо учетных данных допускается использовать форму “username:password”

Подключение компонента генерации Avro-схем

Внимание

Необходимо, чтобы в рабочем пространстве сборочной задачи системы-источника присутствовали файлы дескриптора модели и белого списка, названные согласно инструкции.

Результатом работы компонента библиотеки является появление следующих файлов в рабочем пространстве сборочной Jenkins-задачи, в которой подключена библиотека:

1. **model_schemas_full.txt** — содержит Avro-схемы классов из дескриптора модели;
2. **model_schemas.txt** - содержит только те Avro-схемы классов из дескриптора модели, которые присутствуют в белом списке. Если в архиве не будет файла белого списка, то в данном файле будет отображена ошибка, а сам файл в дальнейшем не пройдет валидацию.

В случае если в процессе генерации схем возникнет ошибка, ее стектрейс будет записан в выходные файлы, а сборка завершится неудачей.

Эти файлы должны быть помещены системой-источником в папку **/other/model** дистрибутива системы-источника.

Порядок подключения библиотеки генерации Avro-схем

- Добавить вызов компонента генерации схем для архива:

```
AvroSchemas.generate(credId, sourceName, modelFilePath, verboseKey,
whiteListFilePath,sdPath)
```

Описание параметров:

Параметр	Описание	Пример использования
credId	Идентификатор учетных данных, которые будут использоваться для скачивания файла, реализующего в себе логику генерации Avro-схем	SBT-SA-TSPI001A
sourceName	Мнемоника системы-источника	

Параметр	Описание	Пример использования
modelFilePath	Путь до файла, содержащего описание модели системы-источника. Если не в текущем каталоге, то с путем	gbk-main.model.v.0.1.json
verboseKey	Признак формирования вербального ключа (использование префикса имени класса в идентификаторе). Имеет значения true/false. Значение verboseKey присутствует в каждой Avro-схеме в словаре ReferenceProps поля doc. Рекомендуется указывать false. В случае необходимости использования true - дополнительно проконсультируйтесь с командой Archiving	false
whiteListFilePath	Примечание: Если ранее вызывался методом configArchive.create() без указания имен файлов модели и белых списков, то файлы будут созданы автоматически с шаблонными именами, например, <sourceName>.whitelist.v.<version>	cdm.whitelist.v.0.0.1
sdPath (Опциональный)	Относительный (от WORKSPACE) путь до файла SourceDescription.yml (указывается, если zip дистрибутива содержит много модулей и каждый каталог содержит свой SourceDescription.yml)	xxxx-crm-deals-journal/

Примечание

Генерация схем производится внешним кодом “schema-generator-from-metamodel-0.0.XX.jar”

R-Pipeline. ИФТ. Валидация дистрибутива системы-источника

Функциональность валидации дистрибутива системы-источника реализована в виде Jenkins-библиотеки (далее по тексту - JSL). Валидатор можно запускать как в виде этапа DPM, так и при сборке дистрибутива непосредственно в сборочных Job.

Внимание:

Валидация дистрибутива не вносит никаких изменений в сборку.

Назначение функциональности:

1. Валидация дистрибутива системы-источника в части наличия в дистрибутиве системы-источника артефакта с параметрами конфигурации Archiving.
2. Проверка корректности параметров и значений в этом артефакте.

Критерии валидности дистрибутива системы-источника:

- Наличие необходимых файлов (белого списка, дескриптора модели (либо файла/файлов ЛМ системы-источника), файла параметров системы-источника, дескриптора версий, Avro-схем).
- Валидность имеющихся файлов (форматный контроль):
    1. Файлы схемы данных (дескриптора модели Archiving) валидируются чтением, по JSON-схеме и накачкой (должны быть валидны).
    2. Файлы схемы данных (model_schemas) проверяются на наличие и валидируются на корректность содержимого попыткой прочтения.
    3. Белые списки проверяются по формату согласно спецификации.
    4. Файл конфигурации YAML проверяется на существование, корректность и наличие всех требуемых параметров (существуют и не пустые).
- Пригодность файлов для реальной работы (применяются специально написанные на Java артефакты валидации); для дескриптора модели проверяется:
    1. соответствие формата типу ЛМ в YAML-файле, и возможность объективно сгенерировать классы;
    2. соответствие сгенерированных классов имеющемуся белому списку в поставке.

Фаза валидации необходима для того, чтобы выполнять проверку дистрибутива системы-источника, прежде чем выполнять попытку конфигурирования с использованием параметров из дистрибутива, на реальном экземпляре Archiving на целевом решении.

В зависимости от результата проверки проставляются флаг QG `pprbod.ift` (статусы `ok_need_config` либо `ok_skip_config`).

В фазу настройки Archiving R-Pipeline ИФТ Фаза конфигурирования экземпляра Archiving встраивается проверка наличия флага `pprbod.ift = ok_skip_config`, и при отсутствии соответствующего флага фаза выполнена успешно не будет. Помимо этого, DPM конвейр должен быть настроен на обязательное наличие флага успешной проверки дистрибутива для перехода на дальнейшие стадии.

Подключение JSL валидации дистрибутива конфигурации Archiving

Внимание:

При работе на этапах сборки (не в DPM) необходимо, чтобы к моменту вызова метода валидации уже имелся собранный архив дистрибутива системы-источника.

Если работа идет в составе Job сборки и дистрибутив уже расположен в рабочем пространстве сборочной Job, то можно оптимизировать процесс, исключив скачивание и распаковку дистрибутива.

Подключение библиотеки валидации

Перед началом работы необходимо зарегистрировать библиотеку в Jenkins (если это еще не сделано).

Подключение библиотеки для тестовой разработки

На этапе разработки и тестирования, а также пока не отсутствуют учетные данные для обращения к системе управления доступом, можно воспользоваться отладочной веткой, не выставляющей флаги в QGM и не выполняющей соединения с системой управления доступом, а просто логирующей информацию о флагах.

При вызовах валидатора можно воспользоваться упрощенной формой, опустив часть параметров (будут использованы значения по умолчанию). Пример:

```
distributive.tsa_validate(sourceName, polygonName, distributivePath, artifactVersion, nexusArtifactId, tuz )
```

На этапе обращения к системе управления доступом и HTTP Bridge в тестовой ветке для проверок версий будет выводиться ошибка:

```
❖ ===INFO: Getting CSRF token...
...
❖ ===ERROR: =======Ошибка обращения к API HttpBridge
...
java.lang.NullPointerException: Cannot invoke method getAt() on null object
```

Если в логах Job появился данный текст - значит все, что можно получить с тестового бранча, уже получено, основные этапы валидации конфигурации пройдены.

Вызов компонента библиотеки валидации дистрибутива

Добавить вызов:

```
distributive.tsa_validate(sourceName, polygonName, distrPath, artifactVersion, nexusArtifactId, tuz, sudirUserId, String nexusGroupId = 'Nexus_PROD')
```

где:

Параметр	Описание
sourceName	Мнемонический идентификатор системы-источника
distrPath	Путь к архиву дистрибутива системы-источника. Это не URL, то есть при вызове tsa_validate вы должны обеспечить наличие файла в пределах файловой системы
artifactVersion	Версия дистрибутива системы-источника. Внимание: значение должно совпадать с тем, что использовалось при вызове configArchive.create() при сборке дистрибутива, так как это значение требуется для выставления флага
nexusArtifactId	Идентификатор артефакта системы-источника в Nexus, используется только для простановки флага pprbod в QGM в пространстве системы-источника
polygonName	Идентификатор целевого решения, на который устанавливается дистрибутив
tuz	id технологической учетной записи системы-источника для возможности выставления флагов из пространства системы-источника. Также допускается использование строки вида "username:password"
sudirUserId	id кредeнталов учетной записи системы-источника для авторизации в систему управления доступом. Также допускается использование строки вида "username:password". Данная учетная запись должна иметь права на выполнение HTTP-запросов к HttpBridge. Для этого необходимо добавить ей такие права, оформив заявку согласно инструкции
nexusGroupId	groupId системы-источника в Nexus. Нужен для простановки в пространстве QGM системы-источника флагов pprbod_ift_ready и pprbod_ift_valid. Значение по умолчанию: Nexus_PROD

Примечание:

Для параметров `tuz` и `sudirUserId` допускается возможность вводить либо учетные данные либо пару "user:password". Признаком того, что используется метод авторизации "user:password", является наличие в параметре символа ':', соответственно **в пароле не должен использоваться этот символ (':')**.

Результат работы валидатора

Итогом работы компонента валидации дистрибутива системы-источника является появление в пространстве QGM:

• Промежуточные флаги, результат валидации. Возможные значения флага `pprbod.ift`:

1. `ok_miss_Avro` (валидация архива конфигурации Archiving прошла успешно, однако отсутствуют либо невалидны Avro-схемы);

2. `err` (архив конфигурации Archiving невалиден).

• Статусы флага `pprbod.ift` успешной валидации определяют необходимость дальнейшего конфигурирования системы-источника в Archiving. Возможные значения:

1. `ok_need_config` (переконфигурирование системы-источника необходимо, поскольку изменились ключевые параметры конфигурации);

2. `ok_skip_config` (нет необходимости дальнейшего конфигурирования системы-источника, т.к. ключевые параметры конфигурации не изменились, в тело флага (body) вписана строка = номер ранее установленной версии вида "D-XX.XXX.XX").

Возможные ошибки валидации

При валидации белого списка возможны ошибки, связанные с некорректным синтаксисом.

Библиотека имеет возможность построчного анализа белого списка.

При этом в лог выводится каждая строка белого списка и результат проверки этой строки - `true` или `false`. `true` - строка валидна, `false` - строка с нарушением структуры.

R-Pipeline. ИФТ. Фаза конфигурирования экземпляра Archiving

Функциональность конфигурирования Archiving на основе дистрибутива системы-источника реализована в виде Jenkins-библиотеки (далее по тексту - JSL).

Внимание:

Системы-источники, которые не используют для интеграции с Archiving Platform V DataSpace, Platform V Persistence или универсальный сервис приема векторов изменений, и для обработки данных требуют добавления плагина десериализации данных в состав Archiving, выполняют данный шаг только после подтверждения факта выхода релиза Archiving, содержащего в своей кодовой базе плагин десериализации данных, на контур ИФТ.

Назначение функциональности: конфигурирование Archiving для работы с системой-источником выпускаемой версии. Конфигурирование выполняется на основе артефакта, включенного в дистрибутив системы-источника. Задача конфигурирования выполняется только в том случае, если выставлен флаг Quality Gate `pprbod_ok_need_config`. Наличие такого флага означает, что:

- Дистрибутив успешно прошел валидацию.
- Содержание дистрибутива отличается от текущей конфигурации системы-источника (либо текущая конфигурация отсутствует, т.е. система-источник впервые подключается к Archiving).

Конфигурирования экземпляра Archiving

Для того чтобы встроить задачу конфигурирования экземпляра Archiving, необходимо выполнить следующие действия: перед началом работы зарегистрируйте библиотеку в Jenkins (если это еще не сделано).

Подключение библиотеки для тестовой разработки

На этапе разработки и тестирования, а также пока не получены учетные данные для обращения к системе управления доступом, можно воспользоваться веткой, не выставляющей флаги в QGM, а просто логирующей информацию о флагах.

Вызов компонента библиотеки JSL конфигурирования дистрибутива осуществляется следующим образом:

```
distributive.tsa_configure(sourceName, polygonName, distrArchivePath, tuz,
sudirUserId, nexusGroupId) |
```

где:

Параметр	Описание
sourceName	Мнемонический идентификатор системы-источника
distrArchivePath	Путь к архиву дистрибутива системы-источника. Внимание: для предотвращения повторного скачивания и распаковки дистрибутива (например, если метод конфигурирования вызывается в том же pipelinee, что и метод валидации, и после валидации распакованное содержимое архива не удалялось) то в качестве значения `distrArchivePath` должна быть передана строковая константа `UNPACKED` .Если же метод конфигурирования вызывается НЕ в том же pipelinee, что метод валидации, то в качестве значения этого параметра передается путь к архиву дистрибутива системы-источника (например, "${WORKSPACE}/gbk-main.1.0.0.zip").
polygonName	Идентификатор целевого решения, на который устанавливается дистрибутив (сейчас доступны варианты `IFT_RB` и `IFT_KB`, также допускается использование имен `DENON` и `BOSTON` соответственно именам стендов.
`tuz`	id технологической учетной записи системы-источника для возможности выставления флагов из пространства системы-источника, допускается передача в виде строки "username:password".

Параметр	Описание
sudirUserId	id кредeнталов учетной записи системы-источника для авторизации в систему управления доступом.
nexusGroupId	`groupId` системы-источника в Nexus

Итогом работы компонента является появление в пространстве QGM флага, определяющего результат.

Возможные значения:

Флаг	Статус флага	Описание
`pprbod.ift`	`ok`	Конфигурирование прошло полностью успешно, т.е. в дистрибутиве присутствуют и валидны файлы, необходимые Archiving).
`pprbod.ift`	`err`	Конфигурирование Archiving закончилось неудачей.

Конфигурация системы-источника сохранена в БД Archiving для того, чтобы сервер начал обрабатывать поток данных. Сервис Archiving нужно перезапускать.

R-Pipeline. ИФТ. Smoke-Regress тесты

Назначение задачи - выполнение **Smoke-regress** автоматизированных тестов с целью проверки корректности работы системы-источника на Archiving после применения обновленной модели системы-источника и настроек. Задача тестов - проверить обратную совместимость, работоспособность репликации после обновления модели.

Данные тесты реализуются по методикам автотестов Archiving для задач regress тестирования в режиме **полной** автоматизации.

Для реализации задачи разрабатывается унифицированный тестовый pipeline, который может быть запущен на любом целевом решении ИФТ.

Запуск Smoke-regress-тестов выполняется до публикации Avro-схем.

В случае ошибки тестов схемы не публикуются, подготовка дистрибутива не инициируется.

Обязательным условием дальнейшего движения дистрибутива системы-источника далее по конвейеру - является успешное прохождение фазы smoke-regress тестирования.

Флаг может принимать следующие статусы:

Статус флага	Описание
ok_smoke_regress	Успешно пройденные тесты позволяют дистрибутиву системы-источника двигаться дальше по конвейеру и устанавливаться на следующее целевое решение. Отсутствие такого флага не препятствует установке дистрибутива системы-источника на решение ИФТ, но запрещает его дальнейшую установку на целевые сервера
err_smoke_regress	Тесты полностью или частично провалены

Работа с клиентской библиотекой Archiving

Ограничения, накладываемые на реализацию системой-источником API Archiving

1. От системы-источника API offline ТКД должен реализовывать только jlby модуль.
2. Допускается, что этот модуль может иметь несколько shard, в этом случае Archiving последовательно опрашивает все shard при offline ТКД. Список зон ММТ, в которых эти shard расположены, в этом случае предварительно конфигурируется при помощи DevOps-конвеера.

API Init (целевой)

Базовые принципы построения целевого API Init:

1. Передача данных по определенному типу.
2. Передача данных партициями, размер партиции определяется динамически (системой-источником).
3. Передача данных выполняется в объеме всей первоначальной выгрузки потоковым образом с разбиением на партиции со стороны системы-источника.
4. Для передачи партиции система-источник разбивает ее на пакеты и отправляет эти пакеты в Archiving.
5. Для передачи одной партиции используется один асинхронный со стороны Archiving и необходимое количество синхронных вызовов API загрузки пакета (части партиции) со стороны системы-источника.
6. Обработка последующих партиций определяется результатом (кодом ответа системы-источника) на запрос получения данных текущей партиции.

Ключевое - это схема импорта потоком, с фиксацией последнего полученного от системы-источника при обработке данных идентификатора объекта и инициировании запроса на следующую партицию с этого идентификатора.

Синхронный API Init V1

```
@Api
public interface InitDataTransportApi {
@ApiMethod(apiName = "initLoad", version = "0.0.1")
String initLoad(String type);
@ApiMethod(apiName = "getBatchCount", version = "0.0.1")
EstimateResult getBatchCount(String loadingId) throws BatchEstimateException;
@ApiMethod(apiName = "loadBatch", version = "0.0.1")
LoadResult loadBatch(String loadingId, int index) throws BatchLoadException;
@ApiMethod(apiName = "abort", version = "0.0.1")
void abort(String loadingId);
}
```

Описание методов InitDataTransportApi

Список методов:

1. **initLoad** - начать выгрузку по типу `type`, вернется идентификатор выгрузки;
2. **getBatchCount** - получить количество партиций для указанного типа. `loadingId` - идентификатор выгрузки, полученный в методе `initLoad`;
3. **loadBatch** - получить партицию с индексом index для идентификатора выгрузки `loadingId`. Предполагается итерирование и вызов этого метода для индексов от нуля до количества партиций, полученного в `getBatchCount`. Результат - набор записей, относящийся к получаемой партиции;
4. **abort** - уведомить систему-источник, что Archiving больше не будет запрашивать партиции и можно освободить ресурсы, занятые для подготовки данных для выгрузки с идентификатором `loadingId`.

Синхронный API Init V2

```
@Api
public interface InitDataSampleApi {
@ApiMethod(apiName = "initLoad", version = "0.0.1")
String initLoad(String type);
@ApiMethod(apiName = "getBatchCount", version = "0.0.1")
```

```
EstimateResult getBatchCount(String loadingId) throws BatchEstimateException;
@ApiMethod(apiName = "loadBatchAsync", version = "0.0.1")
void loadBatchAsync(String loadingId, int index, String requestId) throws
BatchLoadException;
@ApiMethod(apiName = "abort", version = "0.0.1")
void abort(String loadingId);
}
@Api
public interface InitDataSampleLoad {
@ApiMethod(apiName = "loadInitBatch", version = "0.0.1")
void loadInitBatch(InitBatchResult initBatchResult) throws
PartitionResultAcceptException;
}
```

Методы InitDataSampleApi (реализуется на стороне системы-источника)

Описание методов:

1. **initLoad** - начинает выгрузку по типу `type`, возвращает идентификатор выгрузки;
2. **getBatchCount** - возвращает количество партиций для выгрузки, начатой вызовом `initLoad()`;
3. **loadBatchAsync** - инициирует отправку партиции с индексом `index` для идентификатора выгрузки `loadingId`. Предполагается итерирование и вызов этого метода для индексов от нуля до количества партиций, возвращённого `getBatchCount()`. Параметр `requestId` имеет смысл глобально уникального идентификатора, значение должно коррелировать с `InitBatchResult#requestId` в последующем вызове `InitDataSampleLoad.loadInitBatch(InitBatchResult)` на стороне Archiving;
4. **abort** - освобождает все ресурсы, необходимые для отправки данных, относящихся к выгрузке с идентификатором `loadingId`. Выполнив вызов `abort()`, Archiving больше не планирует получать пакеты этой выгрузки.

Описание методов InitDataSampleLoad (реализуется на стороне Archiving)

1. **loadInitBatch** - загружает пакет, отправленный системой-источником; отправка должна быть ранее успешно инициирована вызовом `InitDataSampleApi.loadBatchAsync()` на стороне системы-источника.

Оценка

```
/**
* Результат оценки размера партиции
*/
public class EstimateResult implements Serializable {
private static final long serialVersionUID = -5906616942515441183L;
private final PartitionEstimateCode estimateCode;
private final Integer size;
}
/**
* Возможность оценки
*/
public enum PartitionEstimateCode {
/**
* Система-источник не умеет оценивать по списку ключей размер партиции
* (используется только при ТКД, в Init недопустимо возвращать)
*/
PS_DEFAULT,
/**
* Система-источник оценила размер портиции и подготовила данные для выгрузки
*/
PS_ADAPTIVE_READY,
/**
* Система-источник подготавливает данные
*/
PS_ADAPTIVE_PENDING;
}
/**
* Базовый класс для загрузки пакета данных со стороны системы-источника
*/
public abstract class PartitionResult<T extends Serializable> implements Serializable
{
private static final long serialVersionUID = -5280480427310163519L;
private final long responseId; // ИД пакета (части ответа)
private final String requestId; // ИД запроса с которым коррелирует ответ
private final Timestamp responseTimestamp; // Время формирования пакета
private final int partTotal; // Количество пакетов, на которые разделен ответ (N)
```

```
private final int partCurrent; // Номер текущего пакета (0..N-1)
private final String partHash; // хеш текущей части ответа
private final String entryType; // Тип объекта
private final String zoneId; // MMT зона источника откуда отправлены объекты
private final T data; // Фрагмент контейнера LoadResult - полезная нагрузка
private final String hash; // хеш всего образца для проверки на стороне приемника
после получения всех партиций
...
}
public class InitBatchResult extends PartitionResult<byte[]> {
private static final long serialVersionUID = -1110331909003562897L;
public InitBatchResult(long responseId, String requestId, Timestamp
responseTimestamp,
int partTotal, int partCurrent, String partHash, String entryType, String zoneId,
byte[] data, String hash) {
super(responseId, requestId, responseTimestamp, partTotal, partCurrent, partHash,
entryType, zoneId, data, hash);
}
}
```

Передача данных

```
/**
* Результат загрузки партиции
*/
public class LoadResult implements Serializable {
private static final long serialVersionUID = 7608272960976480122L;
private final ResultCode code;
private final List<DataContainer> dataContainers = new ArrayList<>();
}
/
* Контейнер данных
*/
public class DataContainer implements Serializable {
private static final long serialVersionUID = 3239401317382698849L;
private String key; //id объекта системы-источника
private String entryType; // тип объекта
private Long version; // версия модели данных объекта
```

```
private OperationType operType; // тип операции(для ТКД - create)
private List<String> updAttrs; // список обновляемых аттрибутов
private byte[] data; //сериализованый объект системы-источника
}
/
* Тип операции в случае векторов изменений
*/
public enum OperationType {
CREATE, UPDATE, DELETE;
}
```

API для offline ТКД с построением семпла на стороне системы-источника

Взаимодействие выполняется через интеграционную Kafka Archiving. Для обмена информацией используются топики:

1. _offdq - для запросов к Archiving. В топик должны помещаться сообщения типа **PprbOffdqKeyContainer** - запрос на получение полных версий объектов по списку идентификаторов.

2. _offdq_response - топик ответов Archiving. В него со стороны Archiving должны помещаться сообщения типа **PprbTransportContainer** - универсальный контейнер, содержащий объекты системы-источника.

Типы сообщений **PprbTransportContainer** не меняются. **PprbTransportContainer** дополняется опциональной зоной - которая заполняется в случае, если передаваемый идентификатор ранее был получен от системы-источника в эту итерацию ТКД по семплу.

Формат сообщения PprbOffDqKeyContainer - дополняется идентификатором зоны

Транспортный контейнер ТКД запроса:

```
{
"name": "PprbOffDqKeyContainer",
"namespace": "com.sbt.pprbod.Avro.journal.classes",
"type": "record",
{"name": "key", "type": {"type": "array", "items": "string"} }, // Массив ключей, по которым необходимо получить полные версии объектов из системы-источника
{"name": "entry_type", "type": "string"}, // Тип объекта
{"name": "global_type", "type": "string"} // Глобальный тип. Является атрибутом
```

```
транспортного контейнера с собщением (не заполняется, нужно будет удалить)
{"name": "zone_id", "type": ["null", "string"]} // Идентификатор зоны. Заполняется в случае если идентификатор получен из семпла системы-источника и таким образом известен
}
```

Формат сообщения об ошибке PprbErrorContainer

Транспортный контейнер сообщений ошибок:

```
{
"name": "PprbErrorContainer",
"namespace": "com.sbt.pprbod.Avro.journal.classes",
"type": "record",
"fields":
[
{"name": "data_type", "type": "string" }, // "PprbData_1.0"
{"name": "message_id", "type": "string"}, // Id сообщения, при обработке которого возникло исключение
{"name": "interaction_type", "type": "string"}, // тип взаимодействия при котором случилась ошибка: STREAM - поток, DQ - Решение инцидента качества, QFFDQ - Решение запроса ТКД
{"name": "exception_code", "type": "string"}, // Код ошибки обработки контейнера (по классифиактору Archiving)
{"name": "exception_timestamp", "type": "long"}, // время когда ошибка зафиксирована
{"name": "exception_trace", "type": ["null","string"]}, // Опциональный стектрейс
{"name": "exception_message", "type": ["null","string"]}, // Опциональное инфосообщение
{"name": "id_sample", "type":{ // Массив идентификаторов либо идентифиактор, обработка которых не удалась
"type": "array",
"items": "string"
}
```

API для online ТКД

Взаимодействие выполняется через интеграционную Kafka Archiving, для обмена информацией используется топики:

1. _dq - для запросов к Archiving. В топик могут помещаться сообщения типа **PprbKeyContainer** - запрос на получение полной версии объекта по списку по типу идентификатору и зоне.

2. _dq_response - топик ответов Archiving. В него со стороны Archiving могут помещаться сообщения типа **PprbTransportContainer** - универсальный контейнер, содержащий объекты системы-источника.

Типы сообщений **PprbTransportContainer** не меняются.

Формат сообщения PprbKeyContainer (запрос online ТКД)

Транспортный контейнер ТКД запроса:

```
{
"name" : "PprbKeyContainer" ,
"namespace" : "com.sbt.pprbod.Avro.journal.classes",
"type" : "record" ,
"fields" :[
{ "name" : "key" , "type" : "string" }, // Ключ объекта, по которому зарегистрирован инцидент ТКД
{ "name" : "entry_type" , "type" : "string" }, // Тип объекта
{ "name" : "zone_id" , "type" : "string" }, // Идентификатор зоны Stand-In ZoneID. Является атрибутом транспортного контейнера с собщением. Для источников с репликацией через OGG в данном атрибуте содержится "Global name" экземпляра
]
}
}
```

API системы-источника для offline ТКД

API offline ТКД ММТ

API offline ТКД ММТ - спецификация Java API процесса offline ТКД для использования с ММТ. Является расширением и дополнением существующего **API offline ТКД**, но с поддержкой возможности дробления ответа системы-источника на партиции.

Соглашения о сериализации

Рекомендуется использование KRYO версии 4.0.2 и класс `com.sbt.pprbod.data.utils.KryoUtils`, в котором уже сделаны все необходимые настройки.

В случае ограничений, делающих невозможным использование утилит Archiving для сериализации возможно использование собственного формата сериализации путем реализации интерфейса `com.sbt.pprbod.common.api.TkdObjectDeserializer`.

Примечание:

При реализации собственного десериализатора и использовании в нем KRYO необходимо отключать регистрацию классов, т.е не использовать `com.esotericsoftware.kryo.Kryo#register(java.lang.Class)`, т.к. порядок обхода при регистрации на стороне системы-источника и на стороне Archiving не детерменирован и в случае отличия это приводит к ошибкам сериализации.

Если для отправки векторов изменений используется универсальный вектор, то формировать объекты для Init и ТКД также необходимо в формате универсального вектора, преобразованного в массив байт.

Соглашения о версионировании объектов

Система-источник ведет уникальное на своей стороне версионирование объектов. В случае изменения структуры (модели данных) объекта - должна монотонно увеличиваться и его версия. Версионирование ведется в разрезе каждого типа объекта.

Если система-источник не поддерживает ведение версионирования модели данных, в качестве версии модели данных всегда передается 0.

Внимание

Ведение версии модели данных может сказаться на возможности Archiving понять, какую схему преобразования следует использовать.

Типы данных и интерфейсы для получения семпла идентификаторов

Структуры для запроса семпла идентификаторов:

```
public abstract class PartitionResult<T extends Serializable> implements Serializable
{
private static final long serialVersionUID = -5280480427310163519L;
private final long responseId; // идентификатор партиции ответа
private final String requestId; // идентификатор запроса с которым коррелирует ответ
private final Timestamp responseTimestamp; // Время формирования части ответа
private final int partTotal; // Количество партиций на которые разделен ответ (N)
private final int partCurrent; // Номер текущей партиции (0..N-1)
```

```
private final String partHash; // хеш текущей части семпла для проверки на стороне
приемника
private final String entryType; // Тип объекта
private final String zoneId; // ММТ зона источника откуда отправлены объекты
private final T data; // Фрагмент контейнера LoadResult - полезная нагрузка
private final String hash; // хеш всего образца для проверки на стороне приемника
после получения всех партиций
...
}
public class SampleResult extends PartitionResult<ArrayList<String>> {
private static final long serialVersionUID = 3709039709945461 91L;
protected SampleResult(long responseId, String requestId, Timestamp
responseTimestamp,
int partTotal, int partCurrent, String partHash, String entryType, String zoneId,
ArrayList<String> data, String hash) {
super(responseId, requestId, responseTimestamp, partTotal, partCurrent, partHash,
entryType, zoneId, data, hash);
}
}
```

Отличается способ использования транспортного контейнера. Если в исходном API ТКД (синхронный) система-источник реализовывала на своей стороне интерфейс **public interface QualityDataTransportApi**, в текущем API применяется иной подход:

- Для получения семпла идентификаторов система-источник реализует на своей стороне API public interface QualityDataSampleApi, а именно - метод **getQualitySample** который должен на стороне системы-источника построить срез в виде списка идентификаторов и подготовить ответ, вернув соответствующий код.
- Ответ в формате потока семпла идентификаторов система-источник отправляет на API Platform V Archiving public interface QualityDataSampleLoad, а именно - метод **loadQualitySample** - потоковый, принимающий 1 и более партицию семпла идентификаторов в количестве **partTotal**.

Устанавливается таймаут, в течение которого Archiving ожидает как получение порции данных семпла после вызова getQualityDataSample, так и получение следующей партиции при ее наличии. По умолчанию таймаут составляет 10 минут.

Интерфейс и методы для формирования списка идентификаторов по типу

Интерфейс для получения списка идентификаторов по типу:

```
public interface QualityDataSampleApi {
...
@ApiMethod(apiName = "getQualitySample", version = "0.0.1")
void getQualitySample(String type, String requestId, int sampleThreshold) throws
QualityDataSampleException;
```

Интерфейс QualityDataSampleApi реализуется системой-источником.

Интерфейс и методы для получения потока идентификаторов по типу (семпла)

Интерфейс для получения списка идентификаторов по типу:

```
**public** **interface** QualityDataSampleLoad {
...
@ApiMethod(apiName = "loadQualitySample", version = "0.0.1")
void loadQualitySample(SampleResult sampleResult) throws
PartitionResultAcceptException;
...
}
```

Интерфейс QualityDataSampleLoad. Реализуется на стороне Archiving, вызывается со стороны системы-источника непосредственно. Фиксируется контракт, что в одном пакете SampleResult не может присутствовать идентификаторов, составляющих в совокупной длине более 1000 символов.

Типы данных и интерфейсы для получения результата запроса объектов (семпла системы-источника)

Структуры для оценки размера семпла:

Транспортное сообщение запроса семпла одного типа:

```
{**public** **enum** PartitionEstimateCode {
/**
* Система-источник не умеет оценивать по списку ключей размер партиции
*/
PS_DEFAULT,
/**
* Система-источник оценила размер портиции и подготовила данные для выгрузки
*/
PS_ADAPTIVE_READY,
/**
```

```
* Система-источник подготавливает данные
*/
PS_ADAPTIVE_PENDING;
}
/**
* Результат оценки размера партиции
*/
public class EstimateResult implements Serializable {
private static final long serialVersionUID = -5906616942515441183L;
private final PartitionEstimateCode estimateCode;
private final Integer size;
}
```

Структуры для отдачи полных версий объекта (транспортный контейнер):

Применяется та же структура что для первоначальной версии синхронного ТКД

API ТКД (синхронный)

Транспортное сообщение запроса семпла одного типа:

```
/**
* Результат загрузки партиции
*/
**public** **class** LoadResult **implements** Serializable {
**private** **static** **final** **long** serialVersionUID = 7608272960976480122L;
**private** **final** ResultCode code;
**private** **final** List<DataContainer> dataContainers = **new** ArrayList<>();
}
/**
* Контейнер данных
*/
**public** **class** DataContainer **implements** Serializable {
**private** **static** **final** **long** serialVersionUID = 3239401317382698849L;
**private** String key; //id объекта системы-источника
**private** String entryType; // тип объекта
**private** Long version; // версия модели данных объекта
**private** OperationType operType; // тип операции(для ТКД - create)
**private** List<String> updAttrs; // список обновляемых аттрибутов
**private** **byte**[] data; //сериализованый объект системы-источника
}
```

```
/**
* Тип операции в случае векторов изменений
*/
**public** **enum** OperationType {
CREATE, UPDATE, DELETE;
}
```

Правила заполнения:

- В DataContainer#key вносится ключ по формату обмена данными. Формат ключей для объектов - **без префикса типа (невербальный)**.
- В DataContainer#entryType вносится полный тип передаваемого объекта.
- В DataContainer#version вносится версия передаваемого объекта.
- В DataContainer#operType вносится Create (не изменяемо).
- В DataContainer#updAttrs вносится пустой список.
- В DataContainer#data вносятся сериализованные данные (согласно соглашений о сериализации, описанных выше, либо индивидуального контракта).

Объект QualityBatchResult - пакет для отправки в потоке результатов формирования пакета данных

Транспортное сообщение QualityBatchResult:

```
public class QualityBatchResult extends PartitionResult<byte[]> {
private static final long serialVersionUID = 6961184569997647971L;
public QualityBatchResult(long responseId, String requestId, Timestamp
responseTimestamp,
int partTotal, int partCurrent, String partHash, String entryType, String zoneId,
byte[] data, String hash) {
super(responseId, requestId, responseTimestamp, partTotal, partCurrent, partHash,
entryType, zoneId, data, hash);
}
}
```

Отличается способ использования транспортного контейнера. Если в исходном API ТКД система-источник реализовывала на своей стороне интерфейс **public interface QualityDataTransportApi**, в текущем API применяется иной подход:

- Для получения семпла система-источник реализует на своей стороне API public interface QualityDataSampleApi, а именно - его методы **EstimateResult** (оценки размера пакеита данных) и **getQualityBatch** (собственно запрос данных).

- После построения контейнера LoadResult он сериализуется в массив байт с использованием стандартной Java сериализации.
- Поток байт разбивается на части размером по 900 кБайт (946176 байт), и каждый такой фрагмент упаковывается в объект `QualityBatchResult`. Далее все эти `QualityBatchResult` отправляются Archiving в потоке.
- Ответ в формате потока в виде объектов QualityBatchResult система-источник отправляет на API Platform V Archiving public interface QualityDataSampleLoad, а именно метод **loadQualityBatch**. Метод потоковый, т.е. ожидает получения следующей партиции после получения предыдущей, при условии что в составе партиции получен код состояния `QualityBatchState.PROCESS`.

Устанавливается таймаут, в течение которого Archiving ожидает как получение порции данных контейнера после вызова `getQualityBatch`, так и получение следующей партиции при ее наличии: по умолчанию таймаут составляет 10 минут.

Интерфейс и методы для отправки запроса к системе-источнику по списку идентификаторов

Интерфейс для получения данных по списку идентификаторов:

```
public interface QualityDataSampleApi {
...
@ApiMethod(apiName = "getQualityBatchSize", version = "0.0.1")
EstimateResult getQualityBatchSize(String type) throws BatchEstimateException;
@ApiMethod(apiName = "getQualityBatch", version = "0.0.1")
void getQualityBatch(List<String> keys, String type, String requestId) throws
QualityBatchCreationException;
...
}
```

Интерфейс QualityDataSampleApi реализуется системой-источником.

В метод **getQualityBatch** в качестве значения ключа **key** вносится ключ по формату обмена данными. Формат ключей для объектов - **без префикса типа (невербальный)** с экранированными разделителями.

Интерфейс и методы для получения потока идентификаторов по типу (семпла)

Интерфейс для получения потока полных версий объектов:

```
public interface QualityDataSampleLoad {
...
@ApiMethod(apiName = "loadQualityBatch", version = "0.0.1")
void loadQualityBatch(QualityBatchResult qualityBatchResult) throws
PartitionResultAcceptException;
...
}
```

Интерфейс LoadQualityBatch. Реализуется на стороне Archiving, вызывается со стороны системы-источника либо непосредственно с соблюдением контракта, либо - с использованием клиентской библиотеки Archiving, которая инкапсулирует формирование потока и всех необходимых хешей и идентификаторов.

Фиксируется следующий контракт:

1. В одном пакете QualityBatchResult содержится набор байт, представляющих фрагмент контейнера **LoadResult**, контейнер сериализован в поток с использованием Java сериализации.
2. Количество байт в одном сообщении не превышает 900 кБайт (946176 байт).
3. Фрагменты отправляются последовательно в порядке, соответствующем порядку следования фрагментов при их разбиении на стороне системы-источника.
4. Поле responseId в контейнере заполняется монотонно возрастающими значениями, уникальными в пределах одного запроса requestId, порядок сортировки этих идентификаторов соответствует порядку частей контейнера для его последующего склеивания на стороне Archiving и десериализации.
5. Хеш partHash считается от всего объекта QualityBatchResult для каждой партиции.
6. Хеш hash считается от полного набора байт всего сериализованного контейнера LoadResult до его разбивки на части.

Библиотека Archiving для формирования объектов потока семпла идентификаторов и полных версий объектов

Для удобства и унификации реализации клиентской части Archiving предоставляет библиотеку, инкапсулирующую формирование ответного потока данных в Archiving в части серилизации и разбиения данных:

1. Списка идентификаторов на части, в соответствии с контрактом.
2. Транспортного Контейнера LoadResult на части в соответствии с контрактом.

Классы данной библиотеки содержатся в модуле **data-transport-api**.

Библиотека предствляет несколько основных классов, каждый из которых будет подробно рассмотрен ниже:

1. InitDataSampleService - реализация интерфейса *InitDataSampleApi*, содержащая общую логику серилизации и отпарвки данных начальной загрузки.
2. QualityDataSampleService - реализация интерфейса *QualityDataSampleApi*, содержащая общую логику серилизации и отпарвки данных ТКД.

А также несколько утилитарных классов серилизации *JavaSerializationUtils, KryoSerializationUtils* и поулчения хеша по алгоритму SHA-256 *HashingUtils*.

InitDataSampleService

Класс **InitDataSampleService** реализует интерфейс **InitDataSampleApi** и содержит общую логику отправки начальных данных системы-источника в Archiving. При отправке данных системы-источника обеспечивается их серилизация и разбиение на партиции. Разбиение и отправка осуществляются асинхронно, пул потоков может быть передан через конструктор. Если использовать конструктор без пула потоков, то будет создан пул по умолчанию, который содержит InitDataSampleService*#DEFAULT_NUMBER_OF_THREADS* потоков. Собственая логика функционирования сервиса реализуется системой-источником в виде набора функций и Консюмера, которые передаются сервису в конструктор.

Полный конструктор сервиса:

```
/**
* Конструктор.
*
* @param tsaModule Идентификатор модуля Archiving, реализующего DataSampleLoadApi.
* Всегда pprbod-offline-dq-collector-v4.
* @param sourceZoneId MMT зона, в которой находится экземпляр данного сервиса.
* @param executor Пулл потоков для асинхронного разбиения и отправки в Archiving.
* @param loadRequest Реализация интерфейса запросов.
* @param initLoad Функция обработки запросов инициализации начальной загрузки.
* @param batchCount Функция обработки запросов оценки объёма начальной загрузки.
* @param loadBatchAsync Функция обработки запросов сэмпла данных начальной загрузки.
* @param abort Консюмер обработки запросов сброса инициализации начальной загрузки.
*/
public InitDataSampleService(String tsaModule,
```

```
String sourceZoneId,
ExecutorService executor,
InitDataSampleLoadRequest loadRequest,
Function<String, String> initLoad,
Function<String, EstimateResult> batchCount,
BiFunction<String, Integer, LoadBatchAsyncResultPair> loadBatchAsync,
Consumer<String> abort)
```

Функция обработки запросов инициализации начальной загрузки

Обработку запрса инициализации начальной загрузки система-источник осуществляет через функцию обработки запросов инициализации начальной загрузки. На вход данная функция принимает тип сущности, для которой инициализируется начальная загрузка. Функция возвращает идентификатор загрузки, который используется для запросов оценки объема начальной загрузки, запроса сэмпла данных начальной загрузки или запроса сброса инициализации начальной загрузки.

Функция обработки запросов оценки объёма начальной загрузки

Обработку запроса оценки объёма начальной загрузки система-источник осуществляет через функцию обработки запросов оценки объёма начальной загрузки. На вход данная функция принимает идентификтор начальной загрузки, полученный при выполнении запроса инициализации начальной загрузки `InitDataSampleApi#getBatchCount(String)`. Результат функции `EstimateResult` содержит **количество партиций**, на которые разбиты данные начальной загрузки.

Функция обработки запросов сэмпла данных начальной загрузки

Обработку запроса сэмпла данных начальной загрузки система-источник осуществляет через функцию обработки запросов сэмпла данных начальной загрузки. На вход данная функция принимает идентификатор инициализации начальной загрузки, полученный при выполнении запроса инициализации начальной загрузки, а так же индекс требуемой партиции сэмпла данных. Результат работы функции `LoadBatchAsyncResultPair` содержит тип сущности, а также `LoadResult`, содержащий список объектов, вохдящих в партицию сэмпла данных. Для серелизации объекта `LoadResult` используется утилитный класс `JavaSerializationUtils`. `LoadResult`, в свою очередь, будет разбит на партиции не

превыщающие InitDataSampleService#DEFAULT_MAX_PARTITION_SIZE байт и передан через метод InitDataSampleLoadRequest#loadInitBatch(InitBatchResult).

Общий хеш ответа системы-источника расчитывается по экземпляру LoadResult. хеш партиции расчитывается от массива байт содержащихся в ответе партиции.

Консюмер обработки запросов сброса инициализации начальной загрузки

Обработку запроса сброса инициализации начальной загрузки система-источник реализует через Консюмер обработки запросов сброса инициализации начальной загрузки. На вход данный Консюмер принимает идентификатор инициализации начальной загрузки, полученный при выполнении запроса инициализации начальной загрузки.

QualityDataSampleService

Класс QualityDataSampleService реализует интерфейс QualityDataSampleApi и содержит общую логику серилизации, разбиения на партиции и отправки данных ТКД системы-источника. Разбиение и отправка осуществляются асинхронно, пул потоков может быть передан через конструктор. Если использовать конструктор без пула потоков, то будет создан пул поумолчанию, который содержит QualityDataSamplkeService#DEFAULT_NUMBER_OF_THREADS потоков. Данные, предназначенные для отправки в Archiving, будут получены через ряд фунцкций, требуемых для создания экземпляра класса QualityDataSampleService.

Полный конструктор сервиса:

```
/**
* Конструктор.
*
* @param tsaModule Идентификатор модуля Archiving, реализующего DataSampleLoadApi.
* Всегда pprbod-offline-dq-collector-v4.
* @param sourceZoneId MMT зона, в которой находится экземпляр данного сервиса.
* @param executor Пулл потоков для асинхронного разбиения и отправки в Archiving.
* @param sampleLoadRequest Реализация интерфейса запросов.
* @param batchEstimate Функция обработки запросов оценки объёма данных.
* @param loadKeys Функция обработки запроса ключей.
* @param loadData Функция обработки запроса сэмпла объектов.
*/
public QualityDataSampleService(String tsaModule,
String sourceZoneId,
```

```
ExecutorService executor,
QualityDataSampleLoadRequest sampleLoadRequest,
Function<String, EstimateResult> batchEstimate,
BiFunction<String, Integer, Iterable<String>> loadKeys,
BiFunction<List<String>, String, LoadResult> loadData)
```

Функция обработки запросов оценки объёма данных

Обработку запроса оценки количества ключей система-источник реализует через функцию обработки запросов оценки объёма данных. На вход данная функция принимает тип сущности, для которой необходимо проевести оценку объёма данных на текущую дату. Функция возвращает экземпляр `EstimateResult`, который может содержать **количество пачек доступных для загрузки объектов** заданного типа на текущую дату.

Функция обработки запроса ключей

Обработку запроса ключей система-источник реализует через функцию обработки запроса ключей. В качестве входных параметров данная функция принимиает тип запрашиваемых сущностей, а также количество запрашиваемых ключей. Ожидается, что функция вернет итератор, содержащий не больше ключей, чем было запрошено, актуальных на текущую дату. Ключи будут разбиты на партиции, содержащие не более `QualityDataSampleService#DEFAULT_MAX_KEY_CHARS_COUNT_IN_PARTITION` символов и переданы через метод `QualityDataSampleLoadRequest#loadQualitySample(SampleResult)`.

Общий хеш ответа системы-источника расчитывается по всем ключам итератора. хеш партиции расчитывается от `ArrayList<String>`, содержащего ключи партиции.

Функция обработки запроса сэмпла объектов

Обработку запроса сэмпла данных система-источник реализует через функцию обработки запроса сэмпла объектов. В качестве аргументов данная функция принимает список ключей запрашиваемых экземпляров сущности, а также тип сущности. Результат `LoadResult` содержит список запрашиваемых объектов, актуальных на текущую дату. Для серелизации объекта `LoadResult` используется утилитный класс `JavaSerializationUtils`. Результат будет разбит на партиции, не превыщающие `QualityDataSampleService#DEFAULT_MAX_PARTITION_SIZE` байт и передан через метод *QualityDataSampleLoadRequest#loadQualityBatch(QualityBatchResult)*.

Общий хеш ответа системы-источника расчитывается по экземпляру *LoadResult*. хеш партиции расчитывается от массива байт содержащихся в ответе партиции.

Общие замечания

Общие параметры конструкторов сервисов:

tsaModule - идентификатор модуля Archiving, реализующего DataSampleLoadApi. Всегда pprbod-offline-dq-collector-v4. **sourceZoneId** - MMT зона, в которой находится экземпляр данного сервиса.

Хеш код для соответвующих полей ответа системы-источника расчитывается по алгоритму SHA-256.

Количество потоков, количество символов в партиции ответа на запрос ключей, а так же количество байт в партиции ответа на запрос сэмпла объектов могут быть изменены. Для этого необходимо унаследоваться от `InitDataSampleService` или `QualityDataSampleService` и переопределить соответствующие методы `QulityDataSampleService*#numberOfThreads`, `QulityDataSampleService*#getMaxKeyCharsCountInPartition*`, `QulityDataSampleService*#getMaxPartitionSize*`.

API offline ТКД

Api offline ТКД - спецификация gRPC API для процесса offline ТКД для использования без MMT.

API представлено спецификацией Java транспортных типов и вспомогательной клиентской библиотекой, входящей в артефакт API, которая инкапсулирует внутри себя разбиение результирующего массива объектов с полными версиями систем-источников на фрагменты. Для построения транспортного объекта и его разбиения применяется единая библиотека, входящая в артефакт Archiving data-transport-api.

API системы-источника представляет собой два потоковых gRPC - сервиса:

1. сервис получения семпла идентификаторов по перечню типов;
2. сервис получения семпла системы-источника (полных версий объектов) по списку (семплу) идентификаторов и типу.

Сервис получения семпла идентификаторов

Сервис получения семпла идентификаторов предназначен для получения от системы-источника перечня идентификаторов по запрошенному типу данных. Сервис является потоковым - ответ на один запрос представляет собой от 1 до N ответов в потоке. Если система-источник располагается в разных зонах (shard) - то сервис так же должен присутствовать в каждой из зон, так как Archiving выполняет запрос во все зоны системы-источника, с последующим объединением идентификатора и сохранением информации о том, из какой зоны какие идентификаторы были фактически получены.

Сервис является синхронным: Archiving после отправки ответа ожидает получения ответа в потоке, вызов является блокирующим - до получения всего потока (либо истечения таймаута получения элемента потока) запрос считается выполняющимся, ответ в ФД не отдается.

Формат сообщения запроса семпла идентификаторов по типу:

```
message PprbOffDqSampleRequestItem {
string entry_type = 1; // Тип даных
string global_type = 2; // Глобальный тип данных
uint32 sample_threshold = 3; // Ограничение на размер семпла
uint64 request_id = 4; // ИД запроса
Timestamp request_timestamp = 5; // Время запроса
}
```

Идентификатор запроса - обязательный атрибут, по нему Archiving при обработке потока выполняет построение корреляции полученных ответов в потоке с исходным запросом. Параметр EntryType - тип, по которому запрошены данные. Параметр `global_type` - корневой (глобальный тип), может быть пустым, в таком случае не учитывается. Параметр `sample_thershold` определяет максимальное количество идентификаторов, которые должны войти в семпл (в штуках).

Формат элемента потока ответа

Потоковый ответ - семпл идентификаторов:

```
message PprbOffDqSampleResponse {
uint64 response_id = 1; // ИД партиции ответа
uint64 request_id = 2; // ИД запроса с которым коррелирует ответ
Timestamp response_timestamp = 3; // Время формирования части ответа
uint32 part_total = 4; // Количество партиций на которые разделен ответ (N)
```

```
uint32 part_current = 5; // Номер текущей партиции (0..N-1)
string part_hash = 6; // хеш текущей части семпла для проверки на стороне приемника
string entry_type = 7; // Тип объекта
string zone_id = 8; // MMT зона системы-источника, откуда были получены
идентификаторы
repeated string sample_data = 9; // Данные ответа (сам список идентификаторов), одна
строка - один идентификатор
enum SampleState {
FINAL = 0;
PROCESS = 1;
FAIL = 2;
};
SampleState sample_state = 10; // Статус потока (партиции)
string hash = 11; // хеш всего образца для проверки на стороне приемника после
получения всех партиций
}
```

Сообщение потокового ответа предполагает разбивку ответа на стороне системы-источника при формировании ответа, в зависимости от длины полученного списка идентификаторов. Т.к. идентификаторо всегда либо представляет собой строку, либо всегда приводится к ней - процедура разбиения длинного списка идентификаторов тривиальна, и не требует какой - либо библиотечной инкапсуляции. В качестве контракта фиксируется, что длина одного фрагмента списка идентификаторов должна быть не более 1000 символов. Если в результате построения семпла на стороне системы-источника список идентификаторов превышает эту величину - необходимо выполнять отправку этого списка в несколько партиций.

Отправляемый пакет всегда должен содержать идентификатор исходного запроса (того, в ответ на который сформирован пакет). Отправляемые пакеты должны нумероваться монотонно возрастающей последовательностью целых чисел, начиная с 0.

Если пакетов больше одного, то все пакеты, за исключением финального, должны иметь значение `sample_state=PROCESS`, последний (или единственный) пакет всегда принимает значение `sample_state=FINAL`. Если при формировании пакета на стороне системы-источника возникает исключение либо иная ситуация, в результате которой сформировать пакет невозможно - должен быть отправлен пакет со значением `sample_state=FAIL`: обработка потока на стороне Archiving будет остановлена.

Для контроля целостности каждый пакет потока ответа сопровождается хешами:

1. hash - хеш полного семпла, вычисленный как значение хеша от строки - всех идентификаторов, вошедших в выгрузку, сконкатенированных в единую строку через разделитель точку с запятой, порядок конкатенации строго соответствует порядку следования идентификаторов в пакете.
2. part_hash - хеш текущего пакета, вычисленный как значение хеша от всех идентификаторов, вошедших в текущий пакет, сконкатенированных в единую строку через разделитель точку с запятой, порядок конкатенации строго соответствует порядку следования идентификаторов в пакете.

Внимание:

В случае отсутствия идентификаторов в зоне, в которую был получен запрос, ответ должен содержать в потоке единственный пакет, с нулевым количеством `sample_data` и значение хешей, посчитанное от пустой строки.

Потоковый сервис построения семпла идентификаторов имеет следующую спецификацию

Транспортное сообщение запроса семпла одного типа:

```
service PprbOffDqSampleService {
rpc ComputeSample (PprbOffDqSampleRequest) returns (stream PprbOffDqSampleResponse);
}
```

Сервис получения семпла системы-источника (полных версий объектов)

Сервис предназначен для получения Archiving от системы-источника полных версий объектов по заданному типу и запрошенному перечню идентификаторов.

Формат сообщения запроса размера семпла системы-источника:

```
message PprbOffDqEstimateBatchRequest {
uint64 request_id = 1; // ИД запроса списка объектов по списку идентификаторов
string entry_type = 2; // Тип объекта
repeated string sample_data = 3; // Список идентификаторов для оценки размера
партиции готовых объектов по списку
}
```

Идентификатор запроса - обязательный атрибут, по нему Archiving при обработке потока выполняет построение корреляции полученных ответов в потоке с исходным

запросом. Параметр EntryType - тип, по которому запрошены данные. Параметр global_type - корневой (глобальный тип), может быть пустым, в таком случае не учитывается. Парамет sample_data определяет список идентификаторов, минимум должен быть один, максимум не фиксируется контрактом, но не более 1000.

Формат ответа на запрос размера семпла:

```
message PprbOffDqEstimateBatchResult{
uint32 sample_size = 1; // Размер партиции в штуках объектов системы-источника
enum PartitionEstimateCode{
PS_DEFAULT = 0; // Система-источник не умеет оценивать по списку ключей размер
партиции
PS_ADAPTIVE_READY = 1; // Система-источник оценила размер портиции и подготовила
данные для выгрузки
PS_ADAPTIVE_PENDING = 2; // Система-источник подготавливает данные;
};
PartitionEstimateCode estimate_result = 1; // Статус оценки
}
```

Формат запроса семпла системы-источника:

```
message PprbOffDqLoadRequest {
uint64 request_id = 1; // ИД запроса списка объектов по списку идентификаторов
string entry_type = 2; // Тип объекта
repeated string sample_data = 3; // Список идентификаторов для запроса готовых
объектов по списку
}
```

Идентификатор запроса - обязательный атрибут, по нему Archiving при обработке потока выполняет построение корреляции полученных ответов в потоке с исходным запросом. Должен совпадать с идентификатором, по которому ранее был выполнен запрос размера семпла. Параметр EntryType - тип, по которому запрошены данные. Параметр global_type - корневой (глобальный тип), может быть пустым, в таком случае не учитывается. Параметр sample_data определяет список идентификаторов, минимум должен быть один, максимум не фиксируется контрактом но не более 1000.

Формат элемента потока семпла системы-источника:

```
message PprbOffDqLoadResult {
uint64 response_id = 1; // ИД партиции ответа
uint64 request_id = 2; // ИД запроса списка объектов по списку идентификаторов
```

```
Timestamp response_timestamp = 3; // Время запроса
uint32 part_total = 4; // Количество партиций на которые разделен ответ (N)
uint32 part_current = 5; // Номер текущей партиции (0..N-1)
string part_hash = 6; // хеш текущей части семпла для проверки на стороне приемника
string entry_type = 7; // Тип объекта
string zone_id = 8; // ММТ зона ичточника откуда приехали объекты
bytes part = 9; // Данные ответа (KRYO- сериализованный объект LoadResult)
enum DataState { // Статус отправки
FINAL = 0; // Последний фрагмент, остановка потока
PROCESS = 1; // Есть еще фрагменты
FAIL = 2; // Ошибка, остановка потока
};
DataState sample_state = 10; // Статус потока (партиции)
string hash = 11; // хеш всего образца для проверки на стороне приемника после
получения всех партиций
}
```

Сервис получения размера семпла (синхронный, унарный) и собственно потока семпла (потоковый)

Потоковый ответ, семпл системы-источника:

```
service PprbOffDqEstimateQualityBatch {
rpc ComputeSample (PprbOffDqEstimateBatchRequest) returns
(PprbOffDqEstimateBatchResult);
}
// Загрузка партиции
service PprbOffDqLoadQualityBatch {
rpc LoadQualityBatch (PprbOffDqLoadRequest) returns (stream PprbOffDqLoadResult);
}
```

Артефакты API

Примечание:

Описанное в данном документе API реализовано только в данной версии продукта Archiving.

Maven артефакты DTO, MMT API интерфейсов и вспомогательных библиотек

```
<dependency>
<groupId>sbp.ts.pprbod<groupId>
<artifactId>data-transport-api<artifactId>
<version>04.005.07<version>
<dependency>
```

Протокол offline ТКД

Пример:

```
syntax="proto3";
import "google/protobuf/timestamp.proto";
import "google/protobuf/struct.proto";
package "com.sbt.pprbod.data.transport";
// Сообщения и типы для обработки семплов системы-источника
// Элемент запроса семпла: запрос по одному типу с граничными условиями
message PprbOffDqSampleRequest {
string entry_type = 1; // Тип даных
string global_type = 2; // Глобальный тип данных
uint32 sample_threshold = 3; // Ограничение на размер семпла
uint64 request_id = 4; // ИД запроса
Timestamp request_timestamp = 5; // Время запроса
}
// Часть семпла - сообщение содержащее один или несколько идентификаторов,
составляющих семпл
message PprbOffDqSampleResponse {
uint64 response_id = 1; // ИД партиции ответа
uint64 request_id = 2; // ИД запроса с которым коррелирует ответ
Timestamp response_timestamp = 3; // Время формирования части ответа
uint32 part_total = 4; // Количество партиций на которые разделен ответ (N)
uint32 part_current = 5; // Номер текущей партиции (0..N-1)
string part_hash = 6; // хеш текущей части семпла для проверки на стороне приемника
string entry_type = 7; // Тип объекта
string zone_id = 8; // MMT зона ичточника откуда приехали и дентификаторы
repeated string sample_data = 9; // Данные ответа (сам список идентификаторов)
```

```
enum SampleState {
FINAL = 0;
PROCESS = 1;
FAIL = 2;
};
SampleState sample_state = 10; // Статус потока (партиции)
string hash = 11; // хеш всего образца для проверки на стороне приемника после
получения всех партиций
}
// Сервисы системы-источника
// Запрос семпла по системе-источнику - потоковый сервис
service PprbOffDqSampleService {
rpc ComputeSample (PprbOffDqSampleRequest) returns (stream PprbOffDqSampleResponse);
}
// Запросы полной версии объекта по списку идентификаторов (с учетом
партиционирования)
// Оценка размера пакета данных объектов по семплу
message PprbOffDqEstimateBatchRequest {
uint64 request_id = 1; // ИД запроса списка объектов по списку идентификаторов
string entry_type = 2; // Тип объекта
repeated string sample_data = 3; // Список идентификаторов для оценки размера
партиции готовых объектов по списку
}
message PprbOffDqEstimateBatchResult{
uint32 sample_size = 1; // Размер партиции в штуках объектов системы-источника
enum PartitionEstimateCode{
PS_DEFAULT = 0; // Система-источник не умеет оценивать по списку ключей размер
партиции
PS_ADAPTIVE_READY = 1; // Система-источник оценила размер портиции и подготовила
данные для выгрузки
PS_ADAPTIVE_PENDING = 2; // Система-источник подготавливает данные;
};
PartitionEstimateCode estimate_result = 1; // Статус оценки
}
// запрос партиции по списку объектов
message PprbOffDqLoadRequest {
uint64 request_id = 1; // ИД запроса списка объектов по списку идентификаторов
```

```
string entry_type = 2; // Тип объекта
repeated string sample_data = 3; // Список идентификаторов для запроса готовых
объектов по списку
}
// Пакет сообщения (контейнер)
message PprbOffDqLoadResult {
uint64 response_id = 1; // ИД партиции ответа
uint64 request_id = 2; // ИД запроса списка объектов по списку идентификаторов
Timestamp response_timestamp = 3; // Время запроса
uint32 part_total = 4; // Количество партиций на которые разделен ответ (N)
uint32 part_current = 5; // Номер текущей партиции (0..N-1)
string part_hash = 6; // хеш текущей части семпла для проверки на стороне приемника
string entry_type = 7; // Тип объекта
string zone_id = 8; // ММТ зона ичточника откуда приехали объекты
bytes part = 9; // Данные ответа (KRYO- сериализованный объект LoadResult)
enum DataState { // Статус отправки
FINAL = 0; // Последний фрагмент, остановка потока
PROCESS = 1; // Есть еще фрагменты
FAIL = 2; // Ошибка, остановка потока
};
DataState sample_state = 10; // Статус потока (партиции)
string hash = 11; // хеш всего образца для проверки на стороне приемника после
получения всех партиций
}
// Сервисы
// Оценка размера
service PprbOffDqEstimateQualityBatch {
rpc ComputeSample (PprbOffDqEstimateBatchRequest) returns
(PprbOffDqEstimateBatchResult);
}
// Загрузка партиции
service PprbOffDqLoadQualityBatch {
rpc LoadQualityBatch (PprbOffDqLoadRequest) returns (stream PprbOffDqLoadResult);
}
```

Транспортный контейнер

Пример:

```
{
"name": "PprbOffDqSampleDataContainer",
"namespace": "com.sbt.pprbod.Avro.journal.classes",
"type": "record",
"fields": [
{"name": "data_type", "type": "string" }, // "PprbData_1.0"
{"name": "message_id", "type": "string"}, // Id сообщения, уникален для каждого
контейнера
{"name": "pprbod_client_id", "type": "string"}, //pprbod_client_id - id клиента-
отправителя (уникален для каждого экземпляра Archiving)
{"name": "message_timestamp", "type": "long"}, // время на момент формирования
контейнера
{"name": "global_type", "type": "string"}, // Тип объекта
{"name": "zone_id", "type": "string"}, // ММТ зона ичточника откуда приехали и
дентификаторы
{"name": "PprbOffDqSampleResponse", "type":{ // Массив идентификаторов (собственно
семпл)
"type": "array",
"items": "string"
}
},
{"name": "cont_hash", "type": "bytes"} // хеш контейнера
]
}
```

Сбор метрик с использованием возможностей библиотеки data-transport-api

Устройство библиотеки data-transport-api в части сбора метрик Init и ТКД

В библиотеке был предусмотрен ряд абстрактных интерфейсов для сбора метрик:

```
public interface TsaStopwatch {
void start();
void stop();
}
public interface TsaCounter {
void inc();
}
```

```
public interface TsaMonitoringService {
TsaCounter counter(String name, TsaTags tsaTags);
TsaStopwatch stopwatch(String name, TsaTags tsaTags);
}
public final class TsaTags {
private final TagsType tagsType;
private final Map<String, String> tags;
public TsaTags(TagsType tagsType, Map<String, String> tags) {
this.tagsType = tagsType;
this.tags = tags;
}
}
```

Данные интерфейсы имеют 3 набора реализаций. Точкой входа является реализация класса `com.sbt.pprbod.data.monitoring.TsaMonitoringService`:

1. `com.sbt.pprbod.data.monitoring.custodian.CustodianMonitoringService` - для сбора метрик на Wildfly.
2. `com.sbt.pprbod.data.monitoring.micrometer.MicrometerMonitoringService` - для сбора метрик систем-источников на стеке k8s или OpenShift (опционально).
3. `com.sbt.pprbod.data.monitoring.noop.NoOpMonitoringService` - пустая реализация, не делающая ничего (обратная совместимость).

Настройка метрик на системе-источнике

Для сбора метрик используется библиотека `io.micrometer`. Чтобы библиотека могла выполнять сбор метрик, нужно в контекст spring-boot приложения добавить бин:

```
@Bean
public TsaMonitoringService tsaMonitoringService(MeterRegistry meterRegistry) {
return new MicrometerMonitoringService(meterRegistry);
}
```

Настройка сбора метрик на системе-источнике (Wildfly)

Класс `com.sbt.pprbod.data.utils.InitDataSampleService` и `com.sbt.pprbod.data.utils.QualityDataSampleService` включают в себя конструкторы:

```
/**
* Конструктор.
*
* @param sourceZoneId ММТ зона, в которой находится экземпляр данного сервиса.
```

```
* @param loadRequest Реализация интерфейса запросов.
* @param initLoad Функция обработки запросов инициализации начальной загрузки.
* @param batchCount Функция обработки запросов оценки объёма начальной загрузки.
* @param loadBatchAsync Функция обработки запросов сэмпла данных начальной загрузки.
* @param abort Консюмер обработки запросов сброса инициализации начальной загрузки.
* @param monitoringService Интерфейс метрик.
* @param sourceMnemonic мнемоника системы-источника
*/
public InitDataSampleService(String sourceZoneId,
InitDataSampleLoadRequest loadRequest,
Function<String, String> initLoad,
Function<String, EstimateResult> batchCount,
BiFunction<String, Integer, LoadBatchAsyncResultPair> loadBatchAsync,
Consumer<String> abort,
TsaMonitoringService monitoringService,
String sourceMnemonic) {
this("pprbod-offline-dq-collector-v4", sourceZoneId,
SplitAndSendUtils.createDefaultExecutor(DEFAULT_NUMBER_OF_THREADS, LOGGER),
loadRequest, initLoad, batchCount, loadBatchAsync, abort,
monitoringService, sourceMnemonic);
}
/**
* Конструктор.
* @param tsaModule Идентификатор модуля Archiving, реализующего DataSampleLoadApi.
* Всегда pprbod-offline-dq-collector-v4.
* @param sourceZoneId MMT зона, в которой находится экземпляр данного сервиса.
* @param loadRequest Реализация интерфейса запросов.
* @param initLoad Функция обработки запросов инициализации начальной загрузки.
* @param batchCount Функция обработки запросов оценки объёма начальной загрузки.
* @param loadBatchAsync Функция обработки запросов сэмпла данных начальной загрузки.
* @param abort Консюмер обработки запросов сброса инициализации начальной загрузки.
* @param monitoringService Интерфейс метрик.
* @param sourceMnemonic мнемоника системы-источника
*/
public InitDataSampleService(String tsaModule,
String sourceZoneId,
InitDataSampleLoadRequest loadRequest,
```

```
Function<String, String> initLoad,
Function<String, EstimateResult> batchCount,
BiFunction<String, Integer, LoadBatchAsyncResultPair> loadBatchAsync,
Consumer<String> abort,
TsaMonitoringService monitoringService,
String sourceMnemonic) {
this(tsaModule, sourceZoneId,
SplitAndSendUtils.createDefaultExecutor(DEFAULT_NUMBER_OF_THREADS, LOGGER),
loadRequest, initLoad, batchCount, loadBatchAsync, abort,
monitoringService, sourceMnemonic);
}
/**
* Конструктор.
*
* @param tsaModule Идентификатор модуля Archiving, реализующего DataSampleLoadApi.
* Всегда pprbod-offline-dq-collector-v4.
* @param sourceZoneId MMT зона, в которой находится экземпляр данного сервиса.
* @param executor Пулл потоков для асинхронного разбиения и отправки в Archiving.
* @param loadRequest Реализация интерфейса запросов.
* @param initLoad Функция обработки запросов инициализации начальной загрузки.
* @param batchCount Функция обработки запросов оценки объёма начальной загрузки.
* @param loadBatchAsync Функция обработки запросов сэмпла данных начальной загрузки.
* @param abort Консюмер обработки запросов сброса инициализации начальной загрузки.
* @param monitoringService Интерфейс метрик.
* @param sourceMnemonic мнемоника системы-источника
*/
public InitDataSampleService(String tsaModule,
String sourceZoneId,
ExecutorService executor,
InitDataSampleLoadRequest loadRequest,
Function<String, String> initLoad,
Function<String, EstimateResult> batchCount,
BiFunction<String, Integer, LoadBatchAsyncResultPair> loadBatchAsync,
Consumer<String> abort,
TsaMonitoringService monitoringService,
String sourceMnemonic) {
this.executor = executor;
```

```
this.tsaModule = tsaModule;
this.sourceZoneId = sourceZoneId;
this.loadRequest = loadRequest;
this.initLoad = initLoad;
this.batchCount = batchCount;
this.loadBatchAsync = loadBatchAsync;
this.abort = abort;
this.monitoringService = monitoringService;
this.sourceMnemonic = sourceMnemonic;
}
```

И

```
/**
* Конструктор.
*
* @param sourceZoneId MMT зона, в которой находится экземпляр данного сервиса.
* @param sampleLoadRequest Реализация интерфейса запросов.
* @param batchEstimate Функция обработки запросов оценки объёма данных.
* @param loadKeys Функция обработки запроса ключей.
* @param loadData Функция обработки запроса сэмпла объектов.
* @param monitoringService Интерфейс метрик.
* @param sourceMnemonic мнемоника системы-источника
*/
public QualityDataSampleService(String sourceZoneId,
QualityDataSampleLoadRequest sampleLoadRequest,
Function<String, EstimateResult> batchEstimate,
BiFunction<String, Integer, Iterable<String>> loadKeys,
BiFunction<List<String>, String, LoadResult> loadData,
TsaMonitoringService monitoringService,
String sourceMnemonic) {
this("pprbod-offline-dq-collector-v4", sourceZoneId,
SplitAndSendUtils.createDefaultExecutor(DEFAULT_NUMBER_OF_THREADS, LOGGER),
sampleLoadRequest, batchEstimate, loadKeys, loadData, monitoringService,
sourceMnemonic);
}
/**
* Конструктор.
*
```

```
* @param tsaModule Идентификатор модуля Archiving, реализующего DataSampleLoadApi.
* Всегда pprbod-offline-dq-collector-v4.
* @param sourceZoneId MMT зона, в которой находится экземпляр данного сервиса.
* @param sampleLoadRequest Реализация интерфейса запросов.
* @param batchEstimate Функция обработки запросов оценки объёма данных.
* @param loadKeys Функция обработки запроса ключей.
* @param loadData Функция обработки запроса сэмпла объектов.
* @param monitoringService Интерфейс метрик.
* @param sourceMnemonic мнемоника системы-источника
*/
public QualityDataSampleService(String tsaModule,
String sourceZoneId,
QualityDataSampleLoadRequest sampleLoadRequest,
Function<String, EstimateResult> batchEstimate,
BiFunction<String, Integer, Iterable<String>> loadKeys,
BiFunction<List<String>, String, LoadResult> loadData,
TsaMonitoringService monitoringService,
String sourceMnemonic) {
this(tsaModule, sourceZoneId,
SplitAndSendUtils.createDefaultExecutor(DEFAULT_NUMBER_OF_THREADS, LOGGER),
sampleLoadRequest, batchEstimate, loadKeys, loadData, monitoringService,
sourceMnemonic);
}
/**
* Конструктор.
*
* @param tsaModule Идентификатор модуля Archiving, реализующего DataSampleLoadApi.
* Всегда pprbod-offline-dq-collector-v4.
* @param sourceZoneId MMT зона, в которой находится экземпляр данного сервиса.
* @param executor Пулл потоков для асинхронного разбиения и отправки в Archiving.
* @param sampleLoadRequest Реализация интерфейса запросов.
* @param batchEstimate Функция обработки запросов оценки объёма данных.
* @param loadKeys Функция обработки запроса ключей.
* @param loadData Функция обработки запроса сэмпла объектов.
* @param monitoringService Интерфейс метрик.
* @param sourceMnemonic мнемоника системы-источника
*/
```

```
public QualityDataSampleService(String tsaModule,
String sourceZoneId,
ExecutorService executor,
QualityDataSampleLoadRequest sampleLoadRequest,
Function<String, EstimateResult> batchEstimate,
BiFunction<List<String>, String, LoadResult> loadData,
TsaMonitoringService monitoringService,
String sourceMnemonic) {
this.executor = executor;
this.tsaModule = tsaModule;
this.sourceZoneId = sourceZoneId;
this.sampleLoadRequest = sampleLoadRequest;
this.batchEstimate = batchEstimate;
this.loadKeys = loadKeys;
this.loadData = loadData;
this.monitoringService = monitoringService;
this.sourceMnemonic = sourceMnemonic;
}
```

В качестве новых параметров `monitoringService` и `sourceMnemonic` нужно передавать настроенный бин `com.sbt.pprbod.data.monitoring.custodian.CustodianMonitoringService` и мнемонику системы-источника соответственно.

Настройка бина `com.sbt.pprbod.data.monitoring.custodian.CustodianMonitoringService` заключается просто в его создании, описания метрик он подтянет из ресурсов библиотеки сам:

```
@Bean
public TsaMonitoringService tsaMonitoringService() {
return new CustodianMonitoringService();
}
```

Примечание:

Если не использовать новые конструкторы, то в качестве `monitoringService` будет использован `com.sbt.pprbod.data.monitoring.noop.NoOpMonitoringService`, который не делает ничего.

Зависимость

```
<dependency>
<groupId>sbp.ts.pprbod<groupId>
<artifactId>data-transport-api<artifactId>
<version>04.005.01<version>
<dependency>
```

Миграция на текущую версию

На клиентской части в Archiving поддерживается обратная совместимость. Это позволяет после обновления версии работать также и с системами-источниками прежней версии.

Таким образом для миграции систем-источников на новую версию Archiving никаких действий не требуется.

Быстрый старт

Раздел представляет собой сокращённый вариант раздела 2 по установке и конфигурированию системы-источника - список необходимых действий без подробностей.

Система-источник должна соответствовать требованиям для интеграции системы-источника в рамках целевой DevOps схемы.

Стадия сборки дистрибутива

Этапы сборки:

1. Для систем-источников, использующих Java DTO классы - подключение в проект плагина Archiving для генерации дескриптора модели. Для таких систем-источников предусмотрена возможность разметки физической модели системы-источника аннотациями для унификации работы Archiving и устранения необходимости кодирования данной разметки в коде Archiving.
2. Для систем-источников, использующих иное представление модели данных, кроме DTO - подключение плагина не требуется.
3. Подключение фазы B-Pipeline Сборка конфигурации (конфигурационного архива) для Archiving для сбора от системы-источника и из метамодели и включения в свой собственный дистрибутив параметров и артефактов для Archiving - согласно требования к составу дистрибутива.

Стадия развертывания дистрибутива

Этапы развертывания:

1. проверка дистрибутива;
2. конфигурирование Archiving для системы-источника;
3. выполнение smoke-regress тестов системы-источника.

По результатам каждой фазы проставляются соответствующие Quality Gate.

Дистрибутив системы-источника Archiving поставляется в виде zip-архива. Для Archiving дистрибутив системы-источника включает артефакт с конфигурацией системы-источника. Это файл, содержащий артефакты конфигурации, необходимые для корректной

работы Archiving с версией модели данных системы-источника, соответствующей текущему выпускаемому дистрибутиву.

Разметка физической модели системы-источника аннотациями - для систем-источников, использующих классические Java DTO объекты, Archiving фиксирует и предлагает к работе набор аннотаций, с помощью которых можно разметить классы и их поля, тем самым на уровне кода указав назначение полей.

pprbod-descriptor-generator-plugin - модуль, представляющий собой maven plugin, который необходимо подключить к проекту для генерации дескриптора модели на стадии сборки. Плагин необходимо использовать только для генерации дескриптора модели систем-источников, работающих на основе DTO-классов. Плагин подключается к корневому pom.xml который является parent'ом для дочерних модулей. После работы плагина создается папка pprbod_model_descriptor в корне проекта. Название файла основывется на параметрах плагина: <sourceName>.model.<modelVersion>.json.

Сборка конфигурации (конфигурационного архива) для Archiving

Сборка конфигурации предполагает дополнение сборочного архива системы-источника файлами, предназначенными для настройки Archiving. Внутренний состав сборки никак не затрагивается, собирается командой системы-источника как обычно, без изменения технологического процесса.

Функциональность по сборке конфигурации для Archiving реализована в виде библиотеки Jenkins (далее по тексту - JSL).

Назначение функциональности: сборка артефактов конфигурации Archiving, необходимых для конфигурирования экземпляра Archiving на целевых решениях.

Функциональность (в виде JSL) предназначена для встраивания в сборочный Pipeline системы-источника непосредственно после сборки артефактов самой системой-источником после того, как:

- Артефакты системы-источника собраны.
- Плагином Archiving на этапе выполнения Maven-сборки создан файл дескриптора модели.

Внимание:

Файл дескриптора модели генерируется плагином только для систем-источников, использующих в качестве DTO артефактов java-классы. Для систем-источников, использующих Platform V Persistence или Avro, в качестве дескриптора должно передаваться XML-описание объектов Platform V Persistence либо Avro-схема соответственно.

Подключение JSL сборки конфигурации Archiving

Для сборки дистрибутива системы-источника был разработан компонент библиотеки JSL.

Внимание:

Для систем-источников, не интегрированных с метамоделью необходимо, чтобы в рабочем пространстве сборочной задачи системы-источника присутствовали файлы дескриптора модели и белого списка, поименованные согласно инструкции.

Результатом работы компонента библиотеки является появление в рабочем пространстве сборочной Jenkins-задачи, в которой подключена библиотека, следующих файлов:

1. архива конфигурации (**<имя системы-источника>.model.<artifactVersion>.zip**);
2. дескриптора версий **version.info.yml**;
3. Avro-схемы.

Эти файлы должны быть помещены в папку **/other/model** дистрибутива системы-источника.

1. Порядок подключения сборочной библиотеки

- Зарегистрировать библиотеку в Jenkins (если еще не сделано).
- Добавить вызов компонента библиотеки сборки архива.

2. Подключение компонента генерации Avro-схем

Внимание:

Необходимо, чтобы в рабочем пространстве сборочной задачи системы-источника присутствовали файлы дескриптора модели и белого списка, поименованные согласно инструкции.

Результатом работы компонента библиотеки является появление в рабочем пространстве сборочной Jenkins-задачи, в которой подключена библиотека, следующих файлов:

1. **model_schemas_full.txt** - содержит Avro-схемы классов из дескриптора модели.
2. **model_schemas.txt** - содержит только те Avro-схемы классов издескриптора модели, которые присутствуют в белом списке. Если в архиве не будет файла белого списка, то в данном файле будет отображена ошибка, а сам файл в дальнейшем не провалидируется.

В случае, если в процессе генерации схем возникнет ошибка, ее стектрейс будет записан в выходные файлы, а сборка завершится неудачей.

Эти файлы должны быть помещены системой-источником в папку **/other/model** дистрибутива системы-источника.

Валидация дистрибутива системы-источника

Функциональность валидации дистрибутива системы-источника реализована в виде Jenkins-библиотеки.

Фаза конфигурирования экземпляра Archiving

Функциональность конфигурирования Archiving на основе дистрибутива системы-источника реализована в виде Jenkins-библиотеки (далее по тексту - JSL).

Назначение функциональности: конфигурирование Archiving для работы с системой-источником выпускаемой версии. Конфигурирование выполняется на основе артефакта, включенного в дистрибутив системы-источника. Задача конфигурирования выполняется только в том случае, если выставлен флаг Quality Gate `pprbod_ok_need_config`. Наличие такого флага означает, что:

1. Дистрибутив успешно прошел валидацию.
2. Содержание дистрибутива отличается от текущей конфигурации системы-источника (либо текущая конфигурация отсутствует, т.е. система-источник впервые подключается к Archiving).

Вызов компонента библиотеки JSL конфигурирования дистрибутива осуществляется следующим образом:

```
distributive.tsa_configure(sourceName, polygonName, distrArchivePath, tuz,
sudirUserId, nexusGroupId)
```

Smoke-Regress тесты

Назначение задачи - выполнение **Smoke-regress** автоматизированных тестов с целью проверки корректности работы системы-источника на Archiving после применения обновленной модели системы-источника и настроек. Задача тестов - проверить обратную совместимость, работоспособность репликации после обновления модели.

Запуск Smoke-regress тестов выполняется до публикации Avro-схем. В случае ошибки тестов схемы не публикуются, подготовка дистрибутива не инициируется. Обязательным условием дальнейшего движения дистрибутива системы-источника далее по конвейеру является успешное прохождение фазы smoke-regress тестирования. По результату выполнения тестов проставляется Quality Gate флаг `pprbod.smoke.regress`.

Настройка DPM (Platform V DevOps Pipeline Management)

Для оформления DPM фазы необходимо наличие Jenkins Job, реализующего данную фазу (по определению фазы DPM). Archiving предоставляет набор Jenkins Job в Jenkins соответствующей фазы, которым может воспользоваться любой источник для создания у себя DPM фазы. Для использования этих Jenkins Job не требуется специальных разрешений, так как DPM имеет все необходимые разрешения для ее вызова.

Использование частного (реализуемого системой-источником) задания Jenkins

Создайте задание Jenkins, выполняющее те же функции, что и коммунальный Job. Это задание будет использоваться на соответствующих фазах конвейера. Далее рассмотрено создание Jenkins Job на примере задачи для фазы проверки дистрибутива (tsa_validate).

В конфигурации используйте входящие параметры:

1. `distrUrl`;
2. `polygonName`;
3. `srcName`;
4. `ArtifactVersion`;
5. `ArtifactId`.

Настройка сервиса и конвейера в DPM

Как в случае использования общей Job, так и в случае написания собственного по инструкции выше, создайте фазу DPM конвейера, использующего либо общий, либо написанный ранее специализированный Job системы-источника. Для создания фазы необходимо наличие релизного конвейера в DPM, в который будет включаться фаза согласно требуемого процесса.

Создание фаз tsa_validate, tsa_configure

В ранее созданном (или существующем) конвейре внутри фазы ИФТ добавьте соответствующие этапы.

Значения флагов

Добавьте проверку и перечислите, при каких значениях флага будет выполнено предусловие и запущена фаза конфигурирования:

1. Если флаг `pprbod.ift` в статусе `ok_need_config`, то фаза запускается и выполняется конфигурирование.
2. Если флаг `pprbod.ift` в статусе `ok_skip_config`, то это говорит о том, что запуск фазы конфигурирования не требуется и его следует пропустить. Так как фаза опциональная, то конвейер DPM продолжит работу и перейдет к следущему этапу.

Настройка фазы tsa_smoke_regress

После настройки конфигурирования настраивается этап тестирования. Для этого воспользуйтесь Jenkins-задачей запуска Smoke-regress тестов.

Job принимает на вход два параметра:

1. **STAND** - то же самое, что polygonName из предыдущих этапов.
2. **SUITE** - путь для автотестов.

API для offline ТКД с построением семпла на стороне системы-источника

Взаимодействие выполняется через интеграционную Kafka Archiving, для обмена информацией используются топики:

1. _offdq - для запросов к Archiving. В топик должны помещаться сообщения типа **PprbOffdqKeyContainer** - запрос на получение полных версий объектов по списку идентификаторов.

2. _offdq_response - топик ответов Archiving. В него со стороны Archiving должны помещаться сообщения типа **PprbTransportContainer** - универсальный контейнер, содержащий объекты системы-источника.

Типы сообщений **PprbTransportContainer** не меняются. PprbTransportContainer дополняется опциональной зоной, которая заполняется в случае, если передаваемый идентификатор ранее был получен от системы-источника в эту итерацию ТКД по семплу.

API для online ТКД

Взаимодействие выполняется через интеграционную Kafka Archiving, для обмена информацией используются топики:

1. _dq - для запросов к Archiving. В топик могут помещаться сообщения типа **PprbKeyContainer** - запрос на получение полной версии объекта по списку по типу идентификатору и зоне.

2. _dq_response - топик ответов Archiving. В него со стороны Archiving могут помещаться сообщения типа **PprbTransportContainer** - универсальный контейнер, содержащий объекты системы-источника.

Типы сообщений **PprbTransportContainer** не меняются.

Часто встречающиеся проблемы и пути их устранения

В текущем релизе отсутствует статистика по FAQ.