1C RabbitMQ

CODENAME «YELLOW RABBIT MQ»

30.03.2018

СЕРЕБРЯНАЯ ПУЛЯ (С)

Предисловие

Подсистема для организации обмена между конфигурациями 1С, а также сторонними системами на любом языке в режиме "реального времени" с помощью концепции «Event Driven Architecture». На базе сервера очередей «RabbitMQ» при помощи протоколов «AMQP», «MQTT». Для «Windows» / «Linux».

Мир развивается и всем нужна интеграция

- Хотите ли Вы передать розничный чек также быстро, как это делает оператор фискальных данных? Причем, если Ваш магазин находится в Благовещенске, а центральный сервер в Калининграде?
- Вы хотите получать события телеметрии с устройств «IoT» внутрь Вашей 1С конфигурации по протоколу «MQTT», чтобы вести учет в мире "интернета вещей»?
- Вы хотите реализовать принципы слабой связанности между своими конфигурациями? Когда данные передаются с гарантией доставки.
- Вы хотите выводить на обслуживание свою «Управление торговлей» и не терять заказы с сайта?

Продукт «Yellow Rabbit MQ» от команды «SilverBulleters»

Команда **"SilverBulleters"** представляет свой продукт, позволяющий добиться интеграции в режиме реального времени, сохранив слабую связанность систем, а также:

- повторного использования уже разработанной топологии интеграции;
- возможность вывода отдельных систем на эксплуатацию без потери передаваемых данных;
- обеспечить предпосылки к внедрению шин данных;
- интегрировать в 1С устройства "интернета вещей" посредством моста «MQTT-to-AMQP».

Продукт поставляется в виде встраиваемого набора компонент/подсистем для конфигураций 1С:Предприятие 8, реализующих взаимодействие с сервером очередей RabbitMQ и упрощающих построение событийных интеграционных схем для конфигураций на базе 1С:Предприятие 8.

Оглавление

Состав продукта

Внешняя компонента V8RMQClient

- Реализует низкоуровневое взаимодействие с сервером очередей RabbitMQ по протоколу AMQP
  - включая возможность работы в режиме удаленного вызова процедур (RPC)
- Предоставляет методы AMQP на уровне прикладного кода 1С:Предприятие 8
  - с русскоязычными/англоязычными методами

Встраиваемая в 1С подсистема «Очереди сообщений RMQ»

- Включает в себя внешнюю компоненту V8RMQClient
  - в виде общего макета
- Предоставляет высокоуровневые средства построения интеграции с учетом специфики 1С
- Содержит готовые прикладные объекты для работы с семантикой событийного обмена

Справка и поддержка

- Руководство по внедрению
- Описание методов внешней компоненты
- Форум технической поддержки

Руководство по внедрению

Документ содержит описание внедрения подсистемы «Очереди сообщений RMQ» в целевую конфигурацию

Установка подсистемы

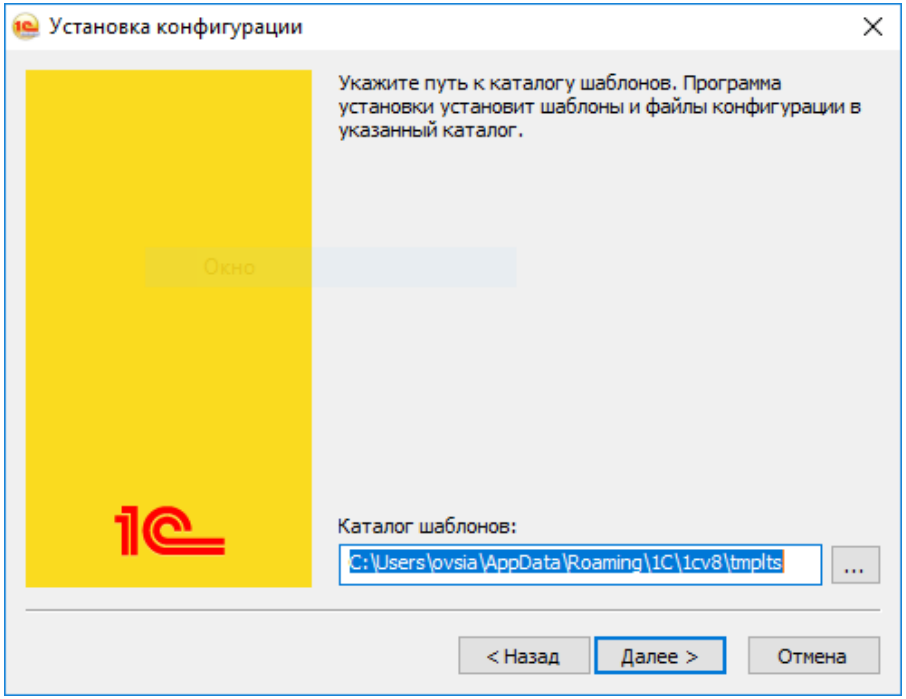
После загрузки и распаковки архива с персональной версией продукта в каталоге распаковки будут находиться:

- Файлы лицензий *.txt и *.md
- Файл документации *.pdf
- Каталог «**dlls**» – содержащий варианты собранной внешней компоненты для различных архитектур и операционных систем
- Каталог «**setup**» - содержащий инсталлятор подсистемы «Очереди сообщений RMQ»
- Каталог «**template**» - содержащий готовый двоичный макет для добавления в любую конфигурацию 1С

Установка шаблона подсистемы

Перейдите в каталог «setup» и запустите файл setup.exe. Этот инсталлятор установит шаблон конфигурации в каталог шаблонов системы 1С:Предприятие 8. По умолчанию этот каталог имеет имя *«%APPDATA%\1C\1cv8\tmplts»*

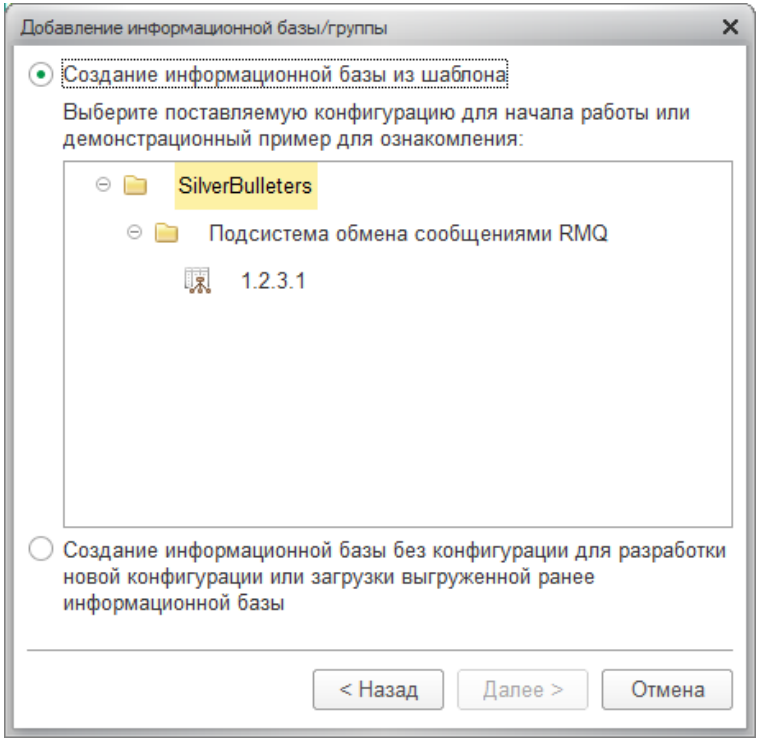
Установите конфигурацию обычным образом, как это делается для типовых конфигураций фирмы 1С.



Дождитесь завершения установки. Теперь, в каталоге шаблонов будет видна подсистема очередей сообщений.

Запустите 1С предприятие и в окне запуска нажмите кнопку «Добавить», затем, перейдите к созданию новой информационной базы.

В окне выбора шаблона базы выберите «Подсистема обмена сообщениями RMQ» и настройте размещение будущей базы.



Установка завершена. После создания базы вы можете запустить ее в режиме «Конфигуратор» и изучить ее составные части.

Установка необходимых зависимостей

Windows

Перед использованием подсистемы или внешней компоненты требуется установить зависимости

- Visual C++ Redistributable Packages - https://www.microsoft.com/ru-RU/download/details.aspx?id=40784

Обратите внимание ваших администраторов – необходимо устанавливать пакеты для архитектур x86 и x64.

Так как компонента работает в контексте 1С, где в общем случае может быть запущены как 32-битные компоненты сервера (клиента), так и 64 битные службы сервера, такие как rphost, rmngr и т.д.

Рекомендуется использоваться пакетный инсталлятор «chocolatey» и команду для установки необходимого пакета

```
C:\> choco install vcredist2013
```

Указанная команда выполнит установку/обновление x32- и x64-компонент Visual C++ Redistributable Packages.

Linux (Server, Desktop)

Установки дополнительных зависимостей не требуется.

Операционная система должна быть основана на GCC – Debian, Ubuntu, CentOS, RedHat, т.п.

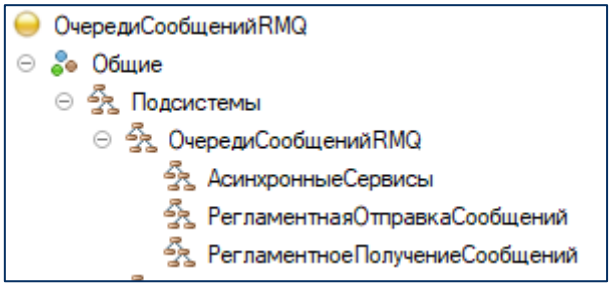
Состав подсистемы

Подсистема состоит из двух основных частей:

1. Основной код, внедряемый в конфигурацию-потребитель
2. Демонстрационный код, предназначенный для изучения подсистемы при самостоятельном запуске. Объекты метаданных, относящиеся к демонстрационным помечены префиксом «Демо» и включены в отдельную подсистему «Демо_ПроверкаРаботы»

В свою очередь, внедряемая часть также разделена на подсистемы, отвечающие за конкретные области интеграции. К таким подсистемам относятся:

- Регламентное получение сообщений
- Регламентная отправка сообщений
- Асинхронные сервисы
- Очереди сообщений RMQ (общие компоненты)



Основной и обязательной к внедрению является подсистема «ОчередиСообщенийRMQ». Остальные подсистемы являются независимыми. В принципе, они могут не внедряться в конфигурацию-потребитель, но делать это не рекомендуется, т.к. подсистема «Очереди сообщений RMQ» является целостной подсистемой, а не набором концептуально независимых подсистем, как, например, БСП фирмы «1С».

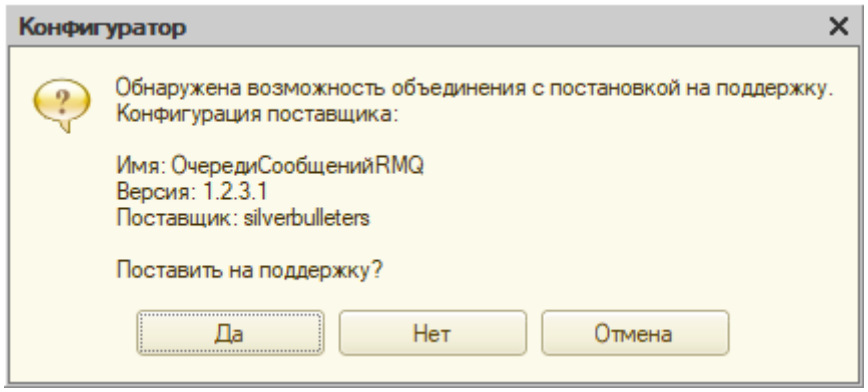
Не исключено, что в будущем, по мере развития программного продукта, подсистемы низшего уровня будут использовать друг друга и станут ссылаться на компоненты друг друга.

Внедрение в конфигурацию

При первом внедрении необходимо перенести объекты метаданных из файла поставки в конфигурацию-потребитель

Для выполнения переноса объектов необходимо открыть конфигуратор конфигурации-потребителя и выбрать режим «Сравнить, объединить с конфигурацией из файла». В качестве файла необходимо указать шаблон конфигурации «Очереди сообщений RMQ» (файл 1Cv8.cf), установленный в каталог шаблонов (см. раздел «Установка подсистемы»)

На вопрос о постановке на поддержку требуется ответить утвердительно.



В открывшемся окне сравнения и объединения конфигураций снять все флажки и отметить переносимые объекты с помощью команды «Действия – Отметить по подсистемам файла.» Далее выбрать подсистему «ОчередиСообщенийRMQ» и все подчиненные ей подсистемы. После чего нажать «Установить»

Далее выбрать режим объединения «Взять из файла» (Действия – Установить режим для всех...) и нажать кнопку «Выполнить».

В окне со списком зависимых объектов нажать кнопку «Продолжить».

Если при попытке объединения конфигураций будет выдано предупреждение об объектах, непомеченных на участие в объединении с одним единственным общим модулем «СтроковыеФункцииГлобальный», тогда нужно нажать «Продолжить».

Особенности внедрения на устаревших версиях платформы 1С:Предприятия

Подсистема «Очереди сообщений» поддерживается на платформах 1С версии 8.2.16 и старше. Прикладной код подсистемы активно использует функции, которые появились значительно позже. Для упрощения внедрения в подсистему добавлен общий модуль «СтроковыеФункцииГлобальный».

Данный модуль по умолчанию не будет включен в конфигурацию (флажок в окне «Сравнения и объединения» для него будет снят).

Если подсистема внедряется в конфигурацию, которая должна работать на платформе версии младше, чем 8.3.6, то для данного модуля необходимо установить флажок вручную.

Удаление ссылок на демонстрационный код

Часть переопределяемых модулей содержит вызовы демонстрационной подсистемы. При внедрении в рабочую среду эти вызовы необходимо удалить. Выполните глобальный поиск строки «// Подсистема.ДемонстрацияРаботы» по текстам конфигурации. Эти вызовы могут располагаться только в переопределяемых модулях подсистемы.

После нахождения – удалите все фрагменты кода между вхождениями строки // Подсистема.ДемонстрацияРаботы.

Внедрение в конфигурации на базе «Библиотеки стандартных подсистем»

При внедрении в конфигурацию, которая использует «Библиотеку стандартных подсистем» фирмы 1С, рекомендуется добавить подсистему «Очереди сообщений RMQ» в список подсистем. Для этого откройте общий модуль БСП

«ПодсистемыКонфигурацииПереопределяемый» и в процедуре «ПриДобавленииПодсистем» добавьте регистрацию модуля «ОбновлениеИнформационнойБазыYRMQ» согласно комментариям к этой процедуре.

Это позволит вам при выходе новых версий подсистемы «Очереди сообщений RMQ» выполнять автоматическое обновление на новые версии.

Описание подсистемы

Подсистема «Регламентное получение сообщений»

Подсистема предназначена для подписки на события от внешних источников. Ядром подсистемы является справочник «ПодпискиНаОчередиСообщений». Каждый элемент этого справочника отвечает за подписку на одну конкретную очередь сообщений сервера RabbitMQ.

Для каждой очереди сообщений в конфигурации-потребителе должен быть определен метод общего модуля, в котором будет реализована обработка входящего сообщения.

Поддерживаются несколько протоколов реализации метода-обработчика.

Обработчик согласно протоколу версии 2 (рекомендуемый):

- Метод должен быть экспортным
- Метод должен принимать 5 параметров:
  - Данные – Строка - текст/тело принятого сообщения;
  - СвойстваСообщения - Соответствие - коллекция свойств сообщения из очереди;
  - ПодпискаНаОчередь - СправочникСсылка.ПодпискиНаОчередиСообщений - Ссылка на элемент справочника обрабатываемой подписки на очередь;
  - Отказ - Булево.
    - Отказ = Ложь отправляет на сервер «RabbitMQ» вызов «BasicAck».
    - Отказ = Истина отправляет на сервер «RabbitMQ» вызов «BasicReject(ВернутьВОчередь)»
    - Оба этих ответа считаются обработкой сообщения.
    - В обоих случаях сообщение удаляется из очереди.
    - Но если есть «dead-letters policy» (задана политика управления подобными сообщениями)
    - И параметр «ВернутьВОчередь» равен значению «Истина»
      - то сообщение, обработанное через «BasicReject», после удаления из очереди перекладывается в другую точку обмена.
    - Подробнее в FAQ подсистемы.

        - Значение по умолчанию - Ложь.
    - ВернутьВОчередь - Булево –
        - Если признак ВернутьВОчередь будет установлен в Истина и Отказ равен Истина,
        - то сообщение считается обработанным и будет перемещено в текущеее место очереди или конец очереди сервера. Подробнее в FAQ подсистемы.
        - Значение по умолчанию - Ложь.

Обработчик согласно протоколу версии 1 (устаревший):

- Метод должен быть экспортным и иметь 2 параметра: Данные и Отказ.

При получении события подсистема будет вызывать указанный прикладной метод, который выполнит полезную работу по интерпретации сообщения.

Регистрация обработчика

За регистрацию обработчиков отвечает переопределяемый модуль «ПолучениеСообщенийПереопределяемый». В этом модуле необходимо разместить подписки на обработчики принимаемых сообщений.

Для этого, в процедуре «ПриРегистрацииОбработчиковСообщений» необходимо добавить строки согласно примеру:

```
////////////////////////////////////////////////////////////////////////////////
// ПРОГРАММНЫЙ ИНТЕРФЕЙС

// Процедура позволяет конфигурации-потребителю событий зарегистрировать //...
Процедура ПриРегистрацииОбработчиковСообщений(Знач Обработчики) Экспорт

    // Для регистрации обработчика необходимо использовать следующий код:
    //
    //      ПолучениеСообщенийСлужебный.ДобавитьОбработчикСобытия(Обработчики, "rmq.my_queue_name", "МойОбщийМодуль.МойМетодОбработки", "2");
    //      ПолучениеСообщенийСлужебный.ДобавитьОбработчикСобытия(Обработчики, "rmq.my_queue_name", "МойОбщийМодуль.МойМетодОбработки");
    //
    //  Последним параметром может передаваться версия обработчика в виже чисда - например, 1 или 2
    //  Если этот параметр не указан, используется версия, возвращаемая методом ВерсияПротоколаПоУмолчанию()
    //
    // Метод обработчика для протокола версии 2 должен иметь параметры:
    //  - ТекстСообщения - Строка - Тело принятого сообщения.
    //  - СвойстваСообщения - Соответствие - коллекция свойств сообщения из очереди.
    //  - ПодпискаНаОчередь - СправочникСсылка.ПодпискиНаОчередиСообщений - Ссылка на элемент справочника обрабатываемой подписки на очередь.
    //  - Отказ - Булево -
    //      Отказ = Ложь отправляет на сервер «RabbitMQ» вызов «BasicAck».
    //      Отказ = Истина отправляет на сервер «RabbitMQ» вызов «BasicReject(ВернутьВОчередь)»
    //      Оба этих ответа считаются обработкой сообщения.
    //      В обоих случаях сообщение удаляется из очереди.
    //      Но если есть «dead-letters policy» (задана политика управления подобными сообщениями)
    //      И параметр «ВернутьВОчередь» равен значению «Истина»
    //          то сообщение, обработанное через «BasicReject», после удаления из очереди перекладывается в другую точку обмена.
    //      Подробнее в FAQ подсистемы.
    //      Значение по умолчанию - Ложь.
    //  - ВернутьВОчередь - Булево - Если признак ВернутьВОчередь будет установлен в Истина и Отказ равен Истина,
    //      то сообщение считается обработанным и будет перемещено в текущеее место очереди или конец очереди сервера. Подробнее в FAQ подсистемы.
    //      Значение по умолчанию - Ложь.
    //
    // Код обработчика должен быть написан таким образом, чтобы выбрасывать исключения только в том случае,
    //  если произошла действительно серьезная ошибка, обработка сообщения сейчас невозможна,
    //  а сервер очередей должен повторить отправку этого же сообщения еще раз.
    //
    // Метод обработчика для протокола версии 1 должен иметь параметры:
    //  - ТекстСообщения - Строка - Тело принятого сообщения.
    //  - Отказ - Булево -
    //      Отказ = Ложь отправляет на сервер «RabbitMQ» вызов «BasicAck».
    //      Отказ = Истина отправляет на сервер «RabbitMQ» вызов «BasicReject(Ложь)»
    //      Оба этих ответа считаются обработкой сообщения.
    //      В обоих случаях сообщение удаляется из очереди.
    //      Подробнее в FAQ подсистемы.
    //      Значение по умолчанию - Ложь.
    //
    // Код обработчика должен быть написан таким образом, чтобы выбрасывать исключения только в том случае,
    //  если произошла действительно серьезная ошибка, обработка сообщения сейчас невозможна,
    //  а сервер очередей должен повторить отправку этого же сообщения еще раз.

КонецПроцедуры
```

Процедура обработчика должна быть написана таким образом, чтобы выбрасывать исключения **только** в том случае, если произошла действительно серьезная ошибка, обработка сообщения сейчас невозможна, а сервер очередей **должен повторить** отправку этого же сообщения еще раз или сообщение может быть переслано по dead-letter каналу, если он настроен.

Если сообщение не может быть обработано, поскольку оно некорректно, и повторная его отправка не требуется, то процедура-обработчик должна присвоить параметру «Отказ» значение «Истина». В этом случае сервер очередей удалит ошибочное сообщение.

Если сообщение не может быть обработано, поскольку оно некорректно, и требуется повторная его отправка, то процедура-обработчик (с версией протокола 2) должна присвоить параметру «ВернутьВОчередь» значение «Истина». В этом случае сообщение считается обработанным и будет размещено в очереди сервера для возможной повторной обработки.

Подробнее смотрите F.A.Q. в разделе «Документация».

При возникновении исключения будет завершен сеанс «слушающего» регламентного задания. Следующее его срабатывание по расписанию получит то же самое сообщение, которое вызвало исключение.

Подсистема «Асинхронные сервисы»

Подсистема предназначена для реализации интеграции посредством «удаленного вызова процедур» (RPC). В такой схеме система 1С может выступать как клиент (вызывающая сторона) и как сервис (принимающая сторона).

Для встраивания в конфигурацию необходимо при сравнении/объединении отметить объекты подсистемы «АсинхронныеСервисы».

Регистрация сервиса

Сервис – это принимающая сторона, реализующая некоторую бизнес-логику. Технически сервис представляет собой процедуру/функцию, которую можно вызвать из других систем.

Для того, чтобы подсистема очередей могла знать, какая именно процедура является «логикой» сервиса, эту процедуру необходимо зарегистрировать в процедуре «ПриРегистрацииОбработчиковСообщений» модуля «АсинхронныеСервисыПереопределяемый».

```
// Процедура позволяет конфигурации-потребителю событий зарегистрировать //...
Процедура ПриРегистрацииОбработчиковСообщений(Знач Обработчики) Экспорт

    // Для регистрации обработчика необходимо использовать следующий код:
    //
    АсинхронныеСервисыСлужебный.ДобавитьОбработчикВызова(Обработчики,
        "rmq.my_queue_name",
        "check",
        "МойОбщийМодуль.МойМетодОбработки");


КонецПроцедуры
```

В данном примере регистрируется в качестве обработчика сервиса метод «МойОбщийМодуль.МойМетодОбработки». Этот метод будет вызываться каждый раз, когда в очереди «rmq.my_queue_name» будет появляться сообщение с параметрами вызова.

Сама функция «МойМетодОбработки» должна иметь один параметр, в который будет передано значение от клиента. Это может быть произвольное значение, а конкретный тип значения определяет автор сервиса.

Подсистема очередей использует для обмена стандартную XML-сериализацию, поэтому передаваемое в качестве параметров значение должно поддерживать XML-сериализацию. Выполнить вызов с несериализуемым типом параметра будет невозможно.

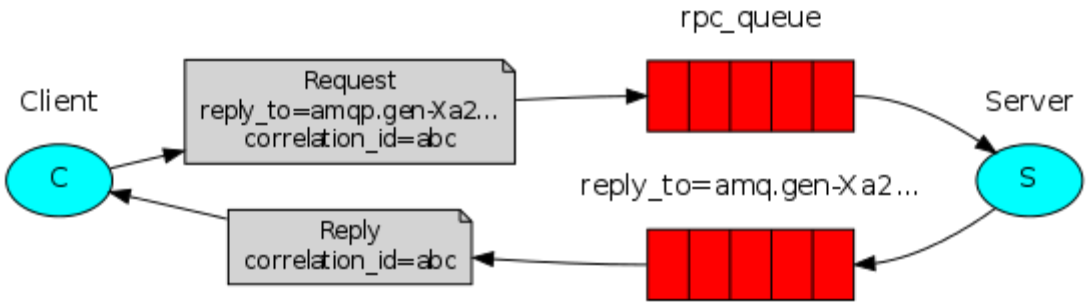
Аналогично метод-обработчик должен возвращать значение, которое может быть сериализовано в XML.

Описание принципов работы RPC-протокола

Протокол вызова процедур, реализуемый в подсистеме, работает по асинхронной схеме:

- Вызов выполняется через промежуточную очередь
- При вызове передается, помимо прочего, имя точки обмена, в которую надо направить ответ сервиса.
- Сервис после обработки вызова формирует ответ и отправляет его в указанную точку обмена.

Клиент, выполнявший вызов, самостоятельно регистрирует подписку на очередь ответа и читает из нее ответы сервиса. Каждый вызов помечается уникальным идентификатором, по которому можно определить – к какому вызову относится ответ.



Разработка клиента сервиса

Для выполнения вызова по описанной схеме разработчик должен воспользоваться API «КлиентАсинхронногоСервиса».

```
Клиент = ОчередьСообщений.НовыйКлиентАсинхронногоСервиса("SendSMS");

Параметры = Новый Структура;
Параметры.Вставить("Телефон","555-55-55");
Параметры.Вставить("Текст"  ,"Привет!");

КлючВызова = Клиент.Вызвать("ОтправитьСМС", Параметры, "");
Ответ = Клиент.ОжидатьОтвет(5);
Если Ответ = Неопределено Тогда
    // не дождались, можем подписаться в фоновом задании или выдать ошибку
Иначе
    СделатьЧтоТоСОтветом(Ответ);
    Сообщить("Ура");
КонецЕсли;
```

Клиент создается вызовом метода «НовыйКлиентАсинхронногоСервиса» фабрики «ОчередьСообщений». В приведенном примере вызывается гипотетический сервис отправки СМС-сообщений. Клиент посредством метода «Вызвать» вызывает метод сервиса, передавая в него параметры. Метод «Вызвать» возвращает идентификатор вызова, по которому можно определить ответ.

Ответ сервиса представляет собой структуру с полями «status» и «value». Поле «status» может принимать значения «ok» или «error» в случае успеха и ошибки соответственно. Если вызов был успешен, то поле «value» содержит ответ сервиса. Если произошла ошибка, то поле «value» содержит описание ошибки.

Документация

- Статья «Подсистема интеграции "реального времени" на базе RabbitMQ для 1С» **http://infostart.ru/public/570477/**
- **Форум поддержки** https://xdd.silverbulleters.org/c/integracziya/yrabbitmq
- Официальный сайт «RabbitMQ» https://www.rabbitmq.com/

Часто задаваемые вопросы (F.A.Q.)

Часто задаваемые вопросы покупателя

- **Как защищена компонента?**
  - компонента и подсистема поставляется в режиме персональной сборки с включением идентификации по клиенту;
  - обновления поставляются только легальным пользователям, не включенным в черный список;
  - в черный список попадают пользователи, чья подсистема и компонента обнаружены в не принадлежащих им конфигурациях 1С или опубликованы публично;
  - черный список публичен;
  - такой порядок сделан для того, чтобы не добавлять проблем с ключами защиты на продуктивных серверах 1С.
- **Можно ли купить подсистему без технической поддержки?**
  - Да, конечно, методы компоненты и подсистемы описаны достаточно понятно, в целом, позволяют их использовать без технической поддержки.

- **Что такое полугодовая подписка на обновление?**
  - Мы активно развиваем подсистему и компоненту;
  - Вместе с этим часть заработанных средств мы передаём на развитие сообщества **oscript.io** и на его поддержку.
  - Чтобы иметь актуальную версию всегда и одновременно поддержать сообщество oscript.io, подписаться на обновление всё же стоит.
  - С другой стороны – подсистема используется в продуктивных контурах уже более года, поэтому подписаться на обновление можно в любой момент, а не сразу при покупке.
  - В первоначальную стоимость продукта подписка на обновление не входит.
- **Как передается дистрибутив и обновление покупателю?**
  - Передача происходит полностью в автоматическом режиме с помощью сервера сборок компании «Серебряная Пуля».
  - Выполняется отправка дистрибутивов на электронную почту покупателя.

Часто задаваемые вопросы разработчика

- **Почему «RabbitMQ»?**
  - По нашим тестам за последние 4 года совокупная стоимость владения контуром «RabbitMQ» наиболее низкая из всех серверов, реализующих протокол AMQP.
  - Также важный момент – «RabbitMQ» используется для интеграции телеметрии внутри платформы «OpenStack» и создавался изначально, чтобы быть максимально производительным в режимах высокой доступности и отказоустойчивости.

- **Какой формат передачи данных выбрать?**
  - Любой - начиная от «JSON» и заканчивая «EDI»
    - Единственная рекомендация - используйте максимально маленькие сообщения, т.к. «RabbitMQ» создан для обеспечения высокоскоростной передачи событий, а не файлов.
  - В целом полезно изучить следующие ссылки или переслать их администраторам:
    - http://www.rabbitmq.com/blog/2012/04/17/rabbitmq-performance-measurements-part-1/
    - http://www.rabbitmq.com/blog/2012/04/25/rabbitmq-performance-measurements-part-2/
- **Как мониторить «RabbitMQ»?**
  - мы советуем использовать уже готовый шаблон «Zabbix» https://github.com/jasonmcintosh/rabbitmq-zabbix;
  - или использовать облачный сервис CloudAMQP со встроенным мониторингом.
- **Как настроить высокую доступность и отказоустойчивость «RabbitMQ»?**
  - используйте https://github.com/silverbulleters-forks/docker-rabbitmq-cluster/blob/master/docs/RU-README.md в качестве персональных экспериментов;
  - попросите своих администраторов настроить контур «RabbitMQ» согласно штатной инструкции;
  - используйте облачный сервис CloudAMQP со встроенной поддержкой отказоустойчивости и высокой доступности.
- **Как обеспечить гарантированность доставки ?**
  - для сообщений, которые вы хотите доставить «обязательно», необходимо:
    - установить флаг «persistence», равным «Истина», при публикации сообщения в методе «BasicPublish»;
    - очередь, которую вы определяете для хранения сообщений, должна быть создана с включенным признаком «durable» (передается как «Истина») в методе «DeclareQueue»;
    - в этом случае очередь будет гарантированно сохранена на диск "в случае аварии"
    - и сообщения, которые в ней хранятся, будет сохранены в базе RabbitMQ (Mnesia).
  - в подсистеме 1С - по умолчанию все сообщения и очереди создаются как гарантированные к доставке;
  - в облачном сервис CloudAMQP дополнительно настроена политика "высокой доступности" –
    - когда очереди и сообщения сохраняются между нодами кластера
    - и сообщения гарантированно будут доставлены, даже если обрушилось "железо";
  - полезно изучить следующий материал https://www.rabbitmq.com/ha.html.

- **Что такое событие?**
  - "документ проведен" – это, конечно, событие, но лучше всего использовать конкретные бизнес-события "в заказе покупателя добавлена новая строка".
- **Я интегрируюсь через COM соединение, SOAP, HTTP, почему же через вашу компоненту интеграция лучше?**
  - Протокол AMQP и C++ компонента ориентирована на высокую нагрузку – «миллионы» событий в секунду.
- **Поясните разницу при программной обработке сообщений – установка параметра Отказ в Ложь/Истина или выброс исключения**
  - Вызов исключения, по сути, разрывает коннект с сервером «RabbitMQ» по таймауту
    - от потребителя не приходит подтверждение («ACK»),
    - сервер «RabbitMQ» считает, что сообщение вообще не обработано.
  - Отказ = Ложь отправляет на сервер «RabbitMQ» вызов «BasicAck».
  - Отказ = Истина отправляет на сервер «RabbitMQ» вызов «BasicReject(ВернутьВОчередь)»
    - Где параметр «ВернутьВОчередь» может принимать значения «Ложь» или «Истина»
  - Оба этих ответа считаются обработкой сообщения.
  - В обоих случаях сообщение удаляется из очереди.
  - Но если есть «dead-letters policy» (задана политика управления подобными сообщениями)
    И параметр «ВернутьВОчередь» параметр «ВернутьВОчередь» в значение «Истина»
    - то сообщение, обработанное через «BasicReject(Истина)», после удаления из очереди перекладывается в другую точку обмена;
  - Подробнее на форуме поддержки.

- **Как правильно переместить сообщение в конец очереди?**
  - Например, не работает метод «Клиент.BasicReject(ПоставитьВОчередь)» с проставленным параметром «ПоставитьВОчередь = Истина», т.к. сообщение не перемещается в конец очереди, а остается сверху.
  - Ответ:
    - Дело в том, что до версии 2.7 сервер «RabbitMQ» всегда ставил сообщение в конец очереди при отказе от сообщения.
      Начиная с версии 2.7, наоборот, сервер старается хранить сообщения в очереди в порядке их поступления и поддерживает этот порядок, насколько это возможно.

- Если сообщение при отказе можно вернуть в исходное место очереди - сервер сделает именно так. Исключение составляют сложные случаи, когда у очереди несколько подписчиков, разбирающих сообщения из очереди одновременно, с разными вариантами подтверждения/неподтверждения. Тогда сервер «RabbitMQ» может принять решение не поддерживать порядок и поставить сообщение в конец очереди.
- Если резюмировать, то на версиях «RabbitMQ» 2.7 и старше нельзя однозначно сказать - будет ли сообщение добавлено в конец или начало при отказе от сообщения с повторной постановкой в очередь.
- конкретная цитата
  - When a message is requeued, it will be placed to its original position in its queue, if possible. If not (due to concurrent deliveries and acknowledgements from other consumers when multiple consumers share a queue), the message will be requeued to a position closer to queue head.
  - Messages can be returned to the queue using AMQP methods that feature a requeue parameter (basic.recover, basic.reject and basic.nack), or due to a channel closing while holding unacknowledged messages. Any of these scenarios caused messages to be requeued at the back of the queue for RabbitMQ releases earlier than 2.7.0. From RabbitMQ release 2.7.0, messages are always held in the queue in publication order, even in the presence of requeueing or channel closure.
- Выводы
  - Надежно поставить сообщение в конец очереди можно только через явную переотправку. Например, с помощью механики «dead-letters-exchange» или полностью ручной переотправкой в ту же самую очередь методом «BasicPublish».
- Подробнее на форуме поддержки.

Техническая информация

- проверена работа во всех релизах, начиная с 8.2.19 и вплоть до 8.3.11 включительно;
- компонента поддерживает архитектуры x64 и x32
  - Windows контуров 1С;
  - Linux контуров на базе Debian (Ubuntu).
- работа подсистемы проверена на серверах RabbitMQ в режимах:
  - простой локальной установки;
  - сложной топологии с применением плагинов Federation и Shovel;
  - в режиме "высокодоступного кластера";
  - в режимах гибридного кластера с применением облачного сервиса CloudAMQP.

Changelog / Изменение в версиях

Подсистема «Очереди сообщений RMQ»

```
# 1.4.0

## Общее

* Регистр сведений `Сервера очередей` заменен на справочник `Серверы очередей RMQ`.
  * Справочник содержит предопределенный элемент `Основной`, который будет использоваться по умолчанию.
  * Предусмотрена миграция параметров подключения из регистра сведений в справочник.
  * Модуль менеджера справочника содержит методы для получения ссылок на основании строковых "ключей соединения".
* Обработка `КлиентОчередиСообщений` - метод `ПолучитьСвойстваСообщения` научился вычитывать все имеющиеся свойства
```

```
* В модуль `ОчередьСообщений` добавлена функция `ВозможныеСвойстваСообщения`, возвращающая возможные имена поддерживаемых свойств сообщения.
* Обработка `ПолучательСобытий` - добавлен метод `ПолучитьСвойстваСообщения`.
* Обработка `ИздательСобытий` - добавлены методы `УстановитьСвойстваСообщения` и `СброситьЗначенияСвойств`

## Получение сообщений

* В справочнике `Подписки на очередь сообщений` строковый реквизит `Ключ соединения` заменен на реквизит `Сервер очередей` - ссылка на справочник `Серверы очередей RMQ`.
* Улучшено внедрение подсистемы для "больших" интеграций.
  * Регламентное задание `Получение сообщений из очередей` больше не требует обязательной привязки обрабатываемых очередей сообщений в коде конфигурации 1С, в процедуре `ПолучениеСообщенийПереопределяемый.ПриРегистрацииОбработчиковСообщений`, все очереди и их обработчики теперь зачитываются из справочника `Подписки на очередь сообщений`.
* Добавлена новая версия (2) протокола для обработчиков получения сообщений. Новая версия протокола содержит дополнительные параметры, такие как `СвойстваСообщения`, `ПодпискаНаОчередь`, `ВернутьВОчередь`.
  * По умолчанию используется первая версия протокола. Глобально переопределить версию можно в процедуре `ПолучениеСообщенийПереопределяемый.ВерсияПротоколаПоУмолчанию`.
    * В следующем релизе версия по умолчанию будет установлена в `2`. Подробнее см. в описании процедуры `ПолучениеСообщенийПереопределяемый.ПриРегистрацииОбработчиковСообщений`.
* Для второй версии протокола обработчиков получения сообщений доступно управление параметром `requeue` при отказе (`reject`) от сообщения.
* Добавлен справочник `Обработчики событий RMQ`, предназначенный для хранения путей к обработчикам получения сообщений из очередей. Данный справочник используется в справочнике `Подписчики на очереди сообщений`.
* В справочнике `Подписки на очередь сообщений` добавлен реквизит `Обработчик` - ссылка на справочник `Обработчики событий RMQ`.
* Исправлена ошибка установки управляемой блокировки в конфигурациях, работающих в автоматическом режиме блокировок.

## Асинхронная отправка сообщений

* Выполнено ускорение отправки сообщений
* Процедура `Опубликовать` модуля менеджера справочника `Исходящие сообщения` объявлена устаревшей. Следует использовать новую процедуру `Справочники.ИсходящиеСообщения.Отправить`.
```

```
  * В следующем релизе эта устаревшая процедура будет удалена.
* В справочник `Исходящие сообщения` добавлен реквизит `Универсальная дата события в миллисекундах`, заполняемый универсальной датой в миллисекундах. Этот же реквизит теперь используется для сортировки отправляемых сообщений при отправке с помощью регламентного задания `Отправка исходящих сообщений`.
* Добавлена возможность устанавливать в отправляемое исходящее сообщение свойства сообщения.
* Исправлена ошибка повторной проверки существования точки обмена на каждое отправляемое сообщение

## Демо-консоль

* Добавлена возможность управления свойствами и заголовками отправляемых сообщений

## Миграция

для конфигураций на базе БСП выполняется автоматическая миграция данных на релиз 1.4.0
```

Описание методов внешней компоненты v1.3 – v1.4

Данный раздел содержит перечень методов внешней компоненты и описывает их использование.

В описании параметров в скобках указаны значения по умолчанию (если они есть)

Метод Connect

Предназначен для установки соединения с сервером RMQ

Параметры:

- host - адрес сервера
- port - порт сервера (5672)
- user - пользователь ("guest")
- password - пароль ("guest")
- vhost - виртуальный хост RMQ ("/")

Метод TuneConnection

Предназначен для тонкой настройки параметров соединения

Параметры:

- maxChannels (0) - максимальное число каналов
- frameSize (65536) - размер кадра AMQP
- heartbeats (0) - интервал отправки пакетов heartbeat.

Метод DeclareExchange

Предназначен для создания новой точки обмена на сервере RMQ.

Параметры:

- name - имя точки обмена
- type - тип точки обмена: fanout|direct|topic
- mustExist (false) - не создавать exchange на сервере, если его нет.
- durable (false) - точка обмена должна кешировать сообщения на диске, на случай падения RMQ
- auto_delete (false) - точка обмена должна быть удалена когда от нее отсоединятся все очереди
- argumentsTable (-1) - идентификатор таблицы аргументов

Метод выбрасывает исключение, если mustExist=true и точки с таким именем на сервере не существует.

МетодBasicPublish

Отправляет сообщение в точку обмена

Параметры:

- exchangeName - имя точки обмена
- routingKey ("") - ключ маршрутизации сообщения (актуально только для topic и direct)
- message - тело сообщения
- TTL - время жизни сообщения в миллисекундах
- message - является ли персистентным

Метод DeclareQueue

Создает новую очередь на сервере RMQ.

Параметры:

- name - имя объявляемой очереди. Если передать пустую строку, то метод вернет имя, сгенерированное сервером.
- mustExist (false) - не создавать очередь с таким именем, использовать существующую.
- * durable (false) - кешировать сообщения на диске, на случай падения RMQ.
- exclusive (false) - только текущее соединение может иметь доступ к этой очереди.
- auto_delete (false) - удалить очередь если к ней был подключен, а затем отключен читающий клиент.
- argumentsTable (-1) - идентификатор таблицы аргументов

Возвращаемое значение

- Строка. Имя очереди, сгенерированное сервером, или заданное явно в параметре.

Метод BindQueue

Устанавливает связь очереди и точки обмена.

Параметры:

- queue_name - имя очереди
- exchange_name - имя точки обмена
- routing_key ("") - ключ маршрутизации. Только сообщения с этим ключом будут попадать в очередь. Для точки обмена topic возможны маски * и #

Метод BasicConsume

Регистрирует читающего клиента (потребителя) на сервере

Параметры:

- queue_name - Имя очереди из которой будем читать.
- consumer_tag ("") - имя потребителя. Если не задан, то имя потребителя сгенерирует сервер и вернет из метода
- no_ack (true) - не ждать подтверждения обработки. Сообщения будут удалены из очереди сразу после отправки на клиента.
- exclusive (false) - монопольно захватить очередь
- prefetch_count (1) - количество сообщений единовременно отправляемых клиенту. Оптимизационный параметр, если > 1 усложняет программирование клиента.

Возвращаемое значение:

- Строка. Имя потребителя, сгенерированное сервером, или заданное явно в параметре.

Метод BasicConsumeMessage

Считывает сообщение из очереди. Метод блокирует текущий поток выполнения, пока не будет получено сообщение или не достигнут таймаут.

Параметры:

- consumer_tag - имя потребителя
- [OUT] result - выходной параметр. Тело сообщения.
- wait_timeout (-1) - таймаут ожидания сообщения в миллисекундах. -1 означает неограниченное ожидание

Возвращаемое значение:

- true, если сообщение получено, false, если истек таймаут.

Метод ConsumeMessageWithBindingKey [УСТАРЕВШЕЕ]

Считывает сообщение из очереди и передает его ключ маршрутизации. Метод блокирует текущий поток выполнения, пока не будет получено сообщение или не достигнут таймаут.

Параметры:

- consumer_tag - имя потребителя
- [OUT] result - выходной параметр. Тело сообщения.
- [OUT] routingKey - выходной параметр. Ключ маршрутизации как он был отправлен в точку обмена.
- wait_timeout (-1) - таймаут ожидания сообщения в миллисекундах. -1 означает неограниченное ожидание

Возвращаемое значение:

- true, если сообщение получено, false, если истек таймаут.

Метод BasicAck

Отсылает серверу подтверждение, что сообщение обработано и его можно удалить.

В API жестко зашит порядок обработки сообщений. Подтвердить можно только последнее прочитанное сообщение.

Таким образом, если prefetchCount больше единицы, то отправлять ack нужно по принципу LIFO (стеком).

Если сервер не получает ack, то он не будет отправлять новых сообщений, а в случае обрыва связи - вышлет неподтвержденные сообщения еще раз.

Метод DeleteQueue

Удаляет очередь с сервера

Параметры:

- queue_name - Имя очереди
- if_unused (false) - удаление будет выполнено, только если очередь не используется
- if_empty (false) - удаление будет выполнено, только если очередь пуста

Метод DeleteExchange

Удаляет точку обмена с сервера.

Параметры

- name - имя точки обмена
- if_unused (false) - удаление будет выполнено, только если точка обмена не используется.

Метод BasicReject

Отказывается от последнего полученного сообщения.

Работает по принципу Ack, но в обратную сторону.

Метод BasicCancel

Отписывает потребителя от очереди.

Параметры

- consumer_tag - имя потребителя

Метод UnbindQueue

Отсоединяет очередь от точки обмена.

Параметры:

- queue_name - имя очереди
- exchange_name_name - имя точки обмена
- routing_key - ключ маршрутизации, указанный при создании привязки в методе BindQueue

Метод GetLastError

Метод позволяет обойти невозможность получения текстов исключений в NativeAPI

Исключение, брошенное компонентой и перехваченное в 1С не содержит текста ошибки (не предусмотрено NativeAPI).

Вызов этого метода позволяет получить текст последней известной ошибки.

Значение последней ошибки обнуляется при очередном успешном вызове любого метода.

Работа со свойствами сообщений

Поддерживаются следующие имена свойств:

- Priority
- ReplyTo
- CorrelationId
- MessageId
- ContentType
- ContentEncoding
- DeliveryMode
- Expiration
- Timestamp
- Type
- AppId
- UserId
- ClusterId
- Headers
- DeliveryTag
- Exchange
- Redelivered
- RoutingKey

Все указанные свойства доступны как свойства объекта, например, «client.ContentType = "text/plain"»

Свойства сообщений можно читать и устанавливать перед отправкой.

Перед отправкой сообщения свойства, которые имеют статус «установленных», передаются в «RabbitMQ» в качестве атрибутов сообщения.

При получении сообщения все свойства «переустанавливаются» из полученного сообщения, а предыдущие значения - сбрасываются.

Пример:

```Код 1С
client.SetProperty("CorrelationId", "corrid-12345");
Сообщить(client.GetProperty("CorrelationId")); // вернет "corrid-12345"

Текст = "";
client.BasicConsumeMessage(consumer_tag, Текст);
// после получения сообщения - все свойства перечитались из него
Сообщить(client.GetProperty("CorrelationId")); // вернет то, что было в сообщении.
```

Метод PropertyIsSet

Проверяет, установлено ли указанное свойство.

Параметры:

- property_name - имя свойства

Метод GetProperty

Получает значение установленного свойства

Параметры:

- property_name - имя свойства

Метод SetProperty

Получает значение установленного свойства

Параметры:

- property_name - имя свойства
- value - значение свойства

Метод ClearProperties

Сбрасывает все свойства в неустановленное состояние