СИСТЕМА АСИНХРОННОГО ОБМЕНА СООБЩЕНИЯМИ АОС-2

Руководство по API Java

Редакция 1.0.0 от 23.11.2020 на 27 листах

Москва 2020

Содержание

1 ВВЕДЕНИЕ

1.1 Область применения

Настоящий документ описывает программный интерфейс к системе АОС-2 для языка программирования Java, см. [3] .

Документ предназначен для разработчиков на языке Java программного обеспечения, использующего обмен сообщениями через систему АОС-2.

1.2 Необходимая эксплуатационная документация

Документ является частью комплекта документации по системе АОС-2. Перед работой с ним необходимо ознакомиться со следующими документами в составе комплекта документации:

«Общее описание системы»;

«Правила именования и соглашения по обмену».

2 НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ

2.1 Назначение интерфейса

Программный интерфейс предназначен для обмена сообщениями информационной системы (далее — прикладной системы), разработанной на языке Java, с другими прикладными системами через систему АОС-2.

2.2 Порядок взаимодействия прикладных систем с системой АОС-2

Прикладная системы, участвующая в обмене сообщений, подключается к определенному узлу системы. После подключения прикладная система может осуществлять обмен сообщениями с другими абонентами системы АОС-2, подключенными к разным узлам.

Прикладная система при передаче сообщений указывает атрибуты и текст сообщения, которые передаются прикладной системе-получателю. Текст сообщения может передаваться порционно.

Параметры подключения к брокеру сообщений узла АОС-2 и имена используемых очередей брокера указываются только в параметрах интерфейса.

При передаче интерфейс записывает сообщение во входную очередь своего узла. Узел после чтения сообщения из входной очереди выполняет маршрутизацию и передает сообщение в очередь прикладной системы получателя или смежный узел. В каждом узле, через который проходит сообщение, могут выполняться сервисные функции изменения атрибутов и текста, копирования, переадресации или блокирования передачи.

Прием сообщения выполняется процедурой с указанием времени ожидания приема. Прием сообщений может осуществляться порционно. После приема сообщения прикладная система выполняет отдельную процедуру подтверждения приема, в результате которой интерфейс удаляет из очереди прочитанное сообщение и передает в узел взаимодействия служебное сообщение с кодом подтверждения.

В общем случае для каждой прикладной системы определяется отдельная очередь приема. Допускается иметь одну очередь приема для нескольких прикладных систем. В этом случае чтение сообщений осуществляется с использованием механизма корреляционной выборки по имени прикладной системы.

Адресация абонентов и форматы сообщений системы АОС-2 описаны в документе «Правила именования и соглашения по обмену» в составе документации по системе АОС-2.

2.3 Программная среда интерфейса

Для работы интерфейса необходимы следующие программные компоненты:

- Java Runtime Environment (JRE) версии 8.0 или выше, см. [5];
- Java Messaging Services (JMS) версии 1.1. Входит в состав J2EE версии 1.4 или выше, см [4] ;
- (Опционально) Реализация JTA с поддержкой ресурсов JMS версии 1.0.1b или выше, см. [6]. Входит в состав J2EE версии 1.4 или выше. Необходимо для работы в транзакционном режиме.
- Клиентская библиотека Artemis ActiveMQ JMS API версии 2.10 или выше, см. [1]; Библиотека включена в состав дистрибутива.
- Интерфейс журналирования SLF4J версии 1.7 или выше, см [7]. Библиотека интерфейса и одна из его реализаций включены в состав дистрибутива.

3 СОСТАВ ДИСТРИБУТИВА

Программный интерфейс системы поставляется в виде одного файла формата архива .tgz (см. [8]). Имя файла имеет вид `stmq-api-java-`*`НомерВерсии`*`.tgz`. Этот архив содержит:

- Следующие библиотеки в каталоге `java`:
  - Библиотеку API АОС-2 `stmq-api-1.0.jar`.
  - Библиотеку JMS API `javax.jms-api-2.0.1.jar` . Библиотека получена из репозитория https://mvnrepository.com/artifact/javax.jms/javax.jms-api/2.0.1 .
  - Библиотеку SLF4J `slf4j-api-1.7.30.jar`. Библиотека получена из репозитория https://mvnrepository.com/artifact/org.slf4j/slf4j-api/1.7.30 .

  - Библиотеки одной из реализаций SLF4J `logback-classic-1.2.3.jar`, `logback-core-1.2.3.jar`. Библиотеки получены из репозитория https://mvnrepository.com/artifact/ch.qos.logback/logback-classic/1.2.3, https://mvnrepository.com/artifact/ch.qos.logback/logback-core/1.2.3 .
  - Библиотека Artemis ActiveMQ JMS Client artemis-jms-client-all-2.13.0.jar. Библиотека получены из репозитория https://mvnrepository.com/artifact/org.apache.activemq/artemis-jms-client-all/2.13.0 .
- Документацию по интерфейсу в каталоге `doc`;
- Примеры использования интерфейса в каталоге `examples.java`.

4 УСТАНОВКА И ИСПОЛЬЗОВАНИЕ ИНТЕРФЕЙСА

Дистрибутив нужно разархивировать в произвольный каталог по выбору пользователя. Далее для определенности это будет каталог

```
/opt/stmq/НомерВерсии/
```

Для использования интерфейса в программе Java нужно, чтобы все библиотеки Java, перечисленные в разделе 3, были бы доступны через переменную CLASSPATH.

В зависимости от среды исполнения программы, пути к библиотекам могут задаваться разными способами. Например, при запуске из командной строки можно указать их их через опцию `-classpath`. Например, при выполнении в среде J2EE-сервера widlfly (см. [9]), все необходимые библиотеки, кроме `stmq-api-1.0.jar`, уже определены сервером; последнюю библиотеку нужно включить в состав приложения или определить на уровне сервера и указать в зависимостях в файле MANIFEST.MF приложения.

Единственной библиотекой, которую в обязательном порядке необходимо брать из дистрибутива, является библиотека `stmq-api-1.0.jar`. Другие библиотеки могут уже находиться в составе среды исполнения, - например, все они уже содержатся в сервере widlfly версии 19 или старше.

5 НАСТРОЙКА ИНТЕРФЕЙСА

В зависимости от среды исполнения, настройка интерфейса выполняется двумя разными способами: для отдельно выполняемой программы и для приложения в составе J2EE-сервера.

5.1 Настройка для отдельно выполняемой программы

В этом случае настройки задаются в файле parmstmq_*ИмяАбонента*.xml, который должен быть доступен программе при исполнении.

Файл имеет следующий вид, значения параметров выделены жирным курсивом:

```
<?xml version="1.0" encoding="UTF-8" ?>
<java>
  <object class="ru.stmq.api.STMQConfig">
    <void method="setConnectionFactory">
      <object
class="org.apache.activemq.artemis.jms.client.ActiveMQJMSConnectionFactory">
        <string>tcp://АдресСервера:ПортСервера</string>
        <string>ИмяПользователя</string>
        <string>ПарольПользователя</string>
      </object>
    </void>
    <void method="setPutDestination">
      <object class="org.apache.activemq.artemis.jms.client.ActiveMQQueue">
        <string>jms.queue.ОчередьОтправкиСообщений</string>
      </object>
    </void>
    <void method="setGetDestination">
      <object class="org.apache.activemq.artemis.jms.client.ActiveMQQueue">
        <string>jms.queue.ОчередьПриемаСообщений</string>
      </object>
    </void>
    <void method="setAnsDestination">
      <object class="org.apache.activemq.artemis.jms.client.ActiveMQQueue">
        <string>jms.queue.ОчередьОтправкиОтветов</string>
      </object>
    </void>
    <void method="setKeepConnect">
      <boolean>true</boolean>
    </void>
    <void method="setCCSID">
      <int>КодоваяСтраницаАбонента</int>
    </void>
  </object>
</java>
```

где параметры выдаются абоненту администратором узла.

Назначение параметров ***ОчередьОтправкиСообщений, ОчередьПриемаСообщений, ОчередьОтправкиОтветов, КодоваяСтраницаАбонента*** определено в документе «Правила именования и соглашения по обмену» в составе настоящей документации.

5.2 Настройка для приложения в составе J2EE-сервера

В этом случае параметры настройки передаются через Java-контекст.

6 ПОРЯДОК РАБОТЫ С ИНТЕРФЕЙСОМ

Все операции отправки и получения сообщений должны выполняться под управлением внешнего менеджера транзакций. Фактическое выполнение действий будет происходить при успешном завершении транзакции (commit).

В случае работы в режиме без транзакций выполнение действий (чтение и отправка сообщений) будет происходить сразу по завершении соответствующего метода, с автоматическим подтверждением (commit).

6.1 Отправка сообщения

Для отправки сообщения предназначен метод put(message). Интерфейс выполняет следующие действия при отправке:

- Формирует адрес создателя сообщения, указанный в конструкторе;
- Формирует дату создания сообщения.
- Формирует двоичное сообщение JMS (javax.jms.BytesMessage);
- Соединяется с очередью отправки, указанную в файле параметров интерфейса;
- Выполняет отправку сообщения;
- Завершает соединение.

6.1.1 Последовательность действий при отправке сообщения

Для отправки сообщения необходимо выполнить следующие действия:

- Создать объект класса STMQInterface с помощью одного из конструкторов класса. Произойдет создание и инициализация интерфейса АОС-2. Инициализация этого объекта выполняется по параметрами, описанным в разделе 5.
- Создать объект класса STMQMessage, вызовом метода createMessage(), передав в качестве параметра текст сообщения. Произойдет создание объекта, представляющего сообщение системы АОС-2. Обязательные свойства будут заполнены на основе значений, считанных из файла конфигурации.
- Заполнить нужные поля объекта-сообщения.
- (Опционально) Стартовать транзакцию, средствами менеджера транзакций. Транзакция может быть уже запущена ранее, тогда этот пункт можно пропустить;
- Выполнить операцию put() объекта STMQInterface, передав в качестве параметра созданный объект-сообщение.
- Завершить (подтвердить) транзакцию. Сообщение будет доступно для обработки другим приложениям только после успешного завершения транзакции.

6.2 Получение сообщения и отправка подтверждения

Для получения сообщения предназначен метод get().

Интерфейс выполняет следующие действия при получении сообщения:

- Соединяется с очередью отправки, указанной в файле параметров интерфейса;

- Инициирует чтение сообщения из очереди.
- Преобразует полученное сообщение в объект STMQMessage;
- Закрывает сессию чтения;
- Возвращает объект STMQMessage или null, если в очереди не было сообщений.

После прочтения и обработки сообщения нужно выполнить операцию отправки подтверждения узлу АОС-2.

Для отправки подтверждения предназначен метод confirm(STMQMessage message, boolean isValid) . В качестве параметров этому методу передаются:

- Исходное сообщение (объект STMQMessage), для которого требуется отправить подтверждение;
- Флаг корректности сообщения с точки зрения прикладной программы:
  - true - сообщение корректно для прикладной системы;
  - false - сообщение некорректно для прикладной системы.

Отправлять подтверждение рекомендуется после выполнения обработки сообщения в общей или в отдельной транзакции. В сообщении будет зафиксировано время обработки сообщения.

6.2.1 Последовательность действий для получения сообщения и отправки подтверждения

Для получения сообщения и отправки подтверждения необходимо выполнить следующие действия:

- Создать объект класса STMQInterface с помощью конструктора класса. Произойдет создание и инициализация интерфейса АОС-2. Инициализация этого объекта выполняется по параметрами, описанным в разделе 5.
- (Опционально) Стартовать транзакцию, средствами менеджера транзакций. Транзакция может быть уже запущена ранее, тогда этот пункт можно пропустить.
- Выполнить операцию get(<таймаут>) объекта STMQInterface и получить сообщение (объект STMQMessage);
- Выполнить прикладной код, обрабатывающий сообщение;
- Выполнить операцию confirm();
- (Опционально)Завершить (подтвердить) транзкцию.

6.3 Обработка сообщений АОС-2 с помощью MDB-компонент

Интерфейс может быть использован в составе компонента Message Driven Bean.

В этом случае чтением сообщений из очереди занимается сервера приложений, а компонент MDB получает на вход объект с интерфейсом javax.jms.Message.

- Создать объект класса STMQInterface с помощью одного из конструкторов класса. Произойдет создание и инициализация интерфейса АОС-2;
- Выполнить операцию parse объекта STMQInterface для преобразования сообщения JMS (javax.jms.Message) в объект STMQMessage;
- Выполнить прикладной код, обрабатывающий сообщение;
- Выполнить операцию confirm();

Работа с транзакцией в этом случае должна выполняться в соответствии с правилами, принятыми для MDB-компонент сервера приложений.

7 ОПИСАНИЕ КЛАССОВ ИНТЕРФЕЙСА

Основные классы интерфейса размещены в пакете `ru.stmq.api`.

Вспомогательные классы размещены в пакете `ru.stmq.api.util`.

7.1 STMQAddress

Класс содержит методы для выделения идентификатора узла и приложения из строкового представления и проверки адреса на соответствие шаблону в соответствии с документом «Правила именования и соглашения по обмену».

Публичные методы:

7.1.1 static String getUserId(String address)

Выделение подстроки, содержащей имя абонента из полного адреса.

Параметры:

- address - объект класса String, содержащий строковое представление адреса абонента в формате системы АОС-2.

Возвращает:

- объект класса String, содержащий строковое представление имени узла АОС-2, или пустую строку, в случае неправильно заданного адреса..

7.1.2 static String getNode(String address)

Выделение подстроки, содержащей имя абонента из полного адреса.

Параметры:

- address – объект класса String, содержащий строковое представление адрес абонента в формате системы АОС-2.

Возвращает:

- объект класса String, содержащий строковое представление имени абонента АОС-2, или пустую строку, в случае неправильно заданного адреса.

7.2 STMQInterface

Класс является основным для организации взаимодействия с системой АОС-2 через API JMS.

Публичные методы:

7.2.1 Конструктор STMQInterface(String userId)

Конструктор класса создает экземпляр класса на основе файла свойств.

Имя файла свойств имеет вид parmstmq_*userId*.xml. Формат файла описан в разделе 5.1

Конструктор предназначен для программ, работающих вне среды J2EE.

Параметры:

- Формирует исключение:
- STMQInterfaceException – при ошибках чтения ресурсов и инициализации интерфейса.

7.2.2 Конструктор STMQInterface(Context ctx, String userId, boolean useCorrId)

Конструктор класса создает экземпляр класса на основе Java-контекста и файла свойств.

Конструктор предназначен для программ, работающих в среде J2EE.

Файл свойств имеет имя:

- parmstmq_*userId*_ru_RU.properties, если useCorrId=False
- parmstmq_ru_RU.properties, если useCorrId=True

Файл свойств имеет формат Java properties. Он может содержать определения перечисленных ниже свойств; жирным выделены обязательные свойства, курсивом — примеры значений:

```
QMGR=connectionFactory // Имя JNDI соединения с узлом
QUEPUT=ISPD // Очередь для приема сообщений
QUEGET=CSQ1 // Очередь для отправки сообщений
QUEANS=CSQ1.ANS // Очередь для отправки ответов
CODE=1251 // CCSID кодовой страницы абонента
ADDR=RU.17.3 // Адрес узла
ADDRTO=RU.17:DODRV // Получатель по умолчанию
AOCNAME=AOCADM // Имя служебного абонента - администратора узла
```

Параметры конструктора:

- ctx – объект класса javax.naming.Context, каталог ресурсов, содержащий ссылки на фабрику соединений.
- userId - имя абонента.
- useCorrId - будет ли использоваться идентификатор корреляции при чтении из очереди приема.

Формирует исключение:

- javax.naming.NamingException - при ошибке получения ссылок на JNDI-ресурсы.
- STMQInterfaceException – при ошибках чтения ресурсов и инициализации интерфейса.

7.2.3 void clear()

Удаление всех сообщений из очереди приема (с учетом фильтра-селектора).

Производит чтение из очереди всех сообщений, согласно фильтру-селектору, без анализа содержимого сообщения.

Можно использовать в начале работы клиента, для предотвращения чтения «старых» сообщений.

Формирует исключение:

- STMQInterfaceException при JMS-ошибках в процессе удаления сообщений.

7.2.4 void close()

Завершение работы интерфейса. Производится очистка внутренних структур.

7.2.5 void confirm(STMQmessage, boolean isValid)

Отправка подтверждения в систему АОС-2.

Параметры:

- message – объект класса STMQMessage, содержащий сообщение, для которого требуется отправка подтверждения;
- isValid –логическое значение (boolean), признак правильности сообщения с точки зрения прикладной программы.

Формирует исключение:

- STMQInterfaceException при ошибке отправки подтверждения.

7.2.6 String createSelector(String userId)

Генерация фильтра (селектора) для выборки сообщений.

Фильтр-селектор используется в операциях get() и clear().

Параметры:

- userId – объект класса String, имя абонента АОС-2.

Возвращает:

- Объект класса String, строка-селектор вида:

```
"JMSCorrelationID='ID:<идентификатор корреляционной выборки>'"
```

<идентификатор корреляционной выборки> - 48 шестнадцатеричных символов (24 байт), представляющих собой имя абонента в кодировке абонента.

В случае ошибок идентификатор будет содержать нули.

7.2.7 STMQMessage get(long timeout)

Чтение сообщения АОС-2 из очереди чтения.

Параметры:

- timeout – Значение типа long, интервал ожидания сообщения.

  Допустимые значения:

  -1 – чтение без ожидания;

  0 – чтение с бесконечным ожиданием;

  положительное число – интервал ожидания в миллисекундах.

Возвращает:

- Объект класса STMQMessage, содержащий прочитанное сообщение АОС-2.
- null, если в очереди сообщений не было подходящих сообщений.

Формирует исключения:

- STMQInterfaceException – при неустранимых ошибках во время чтения сообщения;
- STMQMessageException – при неустранимых ошибках при преобразовании сообщения в объект АОС-2.

7.2.8 String getSelector()

Получение значения фильтра для операции чтения сообщения get().

Возвращает:

- Значение фильтра-селектора для чтения сообщений.

7.2.9 STMQMessage parse(javax.jms.Message jmsMessage)

Извлечение сообщения АОС-2 из двоичного JMS-сообщения (BytesMessage).

Параметры:

- jmsMessage - объект javax.jms.BytesMessage, содержащий сообщение узла АОС-2.

Возвращает:

- объект класса STMQMessage - при успешном преобразовании;
- null, если в качестве параметра был передан null.

Формирует исключение:

- STMQMessageException - при неустранимой ошибке преобразования.

7.2.10 void put(STMQMessage STMQMessage)

Отправка сообщения узлу АОС-2.

Параметры:

- STMQMessage – объект класса STMQMessage. Сообщение, подлежащее отправке.

Формирует исключения:

- STMQInterfaceException – при неустранимых ошибках во время отправки сообщения;
- STMQMessageException – при неустранимых ошибках при преобразовании сообщения в двоичное сообщение JMS.

7.2.11 void setTransacted(boolean isTransacted)

Переключение режима работы с транзакциями.

Параметры:

- isTransacted – объект класса boolean;

  Допустимые значения:

  – true – транзакционный режим. Работа под управлением внешнего менеджера транзакций; Режим по умолчанию.

  – fasle – режим без транзакций; Автоматический commit сразу после выполнения операции чтения или записи сообщения.

7.2.12 void setSelector(.String jmsSelector)

Установка фильтра (JMS selector) для операции чтения сообщения get и очистки clear.

Параметры:

- jmsSelector – объект класса String, текст фильтра;
- null - чтение без фильтра.

7.2.13 STMQMessage createMessage(String messageText)

Генерация объекта STMQMessage на основе переданного текста и параметров интерфейса.

Параметры:

- messageText – объект класса String, текст сообщения.

7.3 STMQMessage

Публичные методы:

7.3.1 Конструктор STMQMessage()

Конструктор класса создает сообщение, содержащее пустую строку.

7.3.2 Конструктор STMQMessage(String messageText)

Конструктор класса создает сообщение, используя в качестве информационного содержимого переданную строку.

Параметры:

- messageText - содержимое передаваемого сообщения.

7.3.3 String getOwner()

Чтение значения адреса зарождения сообщения.

Возвращает:

- адрес зарождения сообщения.

7.3.4 String getFrom()

Чтение значения адреса отправителя сообщения.

Возвращает:

- объект класса String, текущее значение адреса отправителя сообщения.

7.3.5 void setFrom(String fromAddr)

Установка значения адреса отправителя сообщения.

- fromAddr объект класса String, значение адреса отправителя сообщения.

7.3.6 String getTo()

Чтение адреса получателя сообщения.

Возвращает:

- объект класса String, текущее значение адреса отправителя сообщения.

7.3.7 void setTo(String toAddr)

Установка значения адреса получателя сообщения.

- toAddr объект класса String, значение адреса получателя сообщения.

7.3.8 int getCode()

Чтение текущего значения кодовой страницы.

Возвращает:

- Значение типа int, номер кодовой страницы представления текста сообщения.

7.3.9 void setCode(int code)

Установка текущего значения кодовой страницы.

Параметры:

- code - Значение типа int, номер кодовой страницы.

Поддерживаемые значения: 1025(По умолчанию) и 1251.

7.3.10 String getHex()

Чтение признака наличия (Y) или отсутствия(N) двоичной информации в тексте сообщения.

Возвращает:

- объект класса String:
  - Y - в теле сообщения присутствует двоичная информация.

- N – в теле сообщения отсутствует двоичная информация.

7.3.11 void setHex(String isHex)

Установка признака наличия двоичной информации в тексте сообщения.

Параметры:

- isHex - объект класса String:
  - Y - в теле сообщения присутствует двоичная информация.
  - N – в теле сообщения отсутствует двоичная информация.

7.3.12 int getPrty()

Получить текущее значение приоритета сообщения.

Возвращает:

- целое число (int), текущее значение приоритета.

7.3.13 void setPrty(int prty)

Установка значения приоритета для сообщения.

Параметры:

- prty - целое число в диапазоне от 0 до 9.
  Если число больше 9-ти, оно приравнивается 9-ти.

7.3.14 String getTyp()

Получение типа сообщения.

Допустимые значения (перечислены через запятую):

Mes,Req,Ans,Spr,SprA,SMes,SReq,SAns

Возвращает:

- объект класса String, содержащий текущее значение типа сообщения.

7.3.15 void setTyp(String)

Установка типа сообщения.

Допустимые значения (перечислены через запятую):

Mes,Req,Ans,Spr,SprA,SMes,SReq,SAns

Параметры:

- typ - объект класса String, содержащий один из допустимых типов.

7.3.16 int getMsgclass()

Получение текущего значения класса сообщения.

Возвращает:

- целое число, определяющее класс сообщения:
  - 0 – сообщение реального времени;
  - 1 – сеансовое сообщение.

7.3.17 void setMsgclass(int)

Установка текущего значения класса сообщения.

Параметры:

- msgclass – значение типа int.

  Допустимые значения:

  – 0 – сообщение реального времени;

  – 1 – сеансовое сообщение.

7.3.18 String getApo()

Получение значения автоответа терминала отправителя.

Возвращает:

- объект класса String, содержащий текущее значение автоответа терминала отправителя.

7.3.19 void setApo(String apo)

Установка значения автоответа терминала отправителя.

Параметры:

- apo - объект класса String, содержащий текущее значение автоответа терминала отправителя.

7.3.20 String getApto()

Получение значения типа терминала отправителя.

Возвращает:

- объект класса String, содержащий текущее значение типа терминала отправителя.

7.3.21 void setApto(String apto)

Установка значения типа терминала отправителя.

Параметры:

- apto - объект класса String, содержащий текущее значение типа терминала отправителя.

7.3.22 String getApi()

Получение значения автоответа терминала адресата.

Возвращает:

- объект класса String, содержащий текущее значение автоответа терминала адресата.

7.3.23 void setApi(String api)

Установка значения автоответа терминала адресата.

Параметры:

- api - объект класса String, содержащий текущее значение автоответа терминала адресата.

7.3.24 String getApti()

Получение значения типа терминала адресата.

Возвращает:

- объект класса String, содержащий текущее значение типа терминала адресата.

7.3.25 void setApti(String apti)

Установка значения типа терминала адресата.

Параметры:

- apti - объект класса String, содержащий текущее значение типа терминала адресата.

7.3.26 String getAtr()

Получение значения атрибута сообщения.

Возвращает:

- объект класса String, содержащий текущее значение типа атрибута сообщения.

7.3.27 void setAtr(String atr)

Установка значения атрибута сообщения.

Параметры:

- atr - объект класса String, содержащий текущее значение атрибута

  Формат: XX; X - может принимать значения 0-9, A-F.

  Символьное значение XX соответствует шестнадцатеричному значению 0xXX байта атрибута.

7.3.28 String getNumb()

Получение значения номера информационного сообщения, номера формы выходного документа или кода служебной команды.

Возвращает:

- объект класса String, содержащий текущее значение номера.

7.3.29 void setNumb(String numb)

Установка значения номера (информационного сообщения или формы выходного документа или служебной команды).

Параметры:

- numb - объект класса String, содержащий текущее значение номера.

7.3.30 String getSubj()

Получение значения комментария к сообщению.

Возвращает:

- объект класса String, содержащий комментарий к сообщению.
- Возвращает null, если значение не инициализировано.

7.3.31 void setSubj(String subj)

Установка значения комментария к сообщению.

Параметры:

- subj - объект класса String, содержащий комментарий.

В качестве комментария можно передавать текстовую строку не более 80 символов. В случае передачи более длинной строки, она будет урезана. Служебные символы (перевод строки, табуляция и т.п.) будут заменены точками.

7.3.32 java.util.Date getExpiry()

Получение значения времени жизни сообщения.

Возвращает:

- объект класса Date, содержащий время жизни сообщения.
- Возвращает Date(0), если значение не инициализировано.

7.3.33 void setExpiry(java.util.Date expiry)

Установка времени жизни сообщения.

Параметры:

- expiry - объект класса Date, указывающий время жизни сообщения (относительное).

Объект должен хранить не абсолютное значение (дату и время), а относительное (время или число миллисекунд).

7.3.34 void setExpiry(long expiry)

Установка времени жизни сообщения.

Параметры:

- expiry - длинное целое число (long), время жизни сообщения в миллисекундах.

7.3.35 String getImes()

Получение идентификатора сообщения.

Возвращает:

- объект класса String, содержащий идентификатор сообщения в формате GGGGXXZZHHMMSSDCAAAAAA.

GGGG – год

XX – месяц

ZZ – день месяца

HH – часы

MM – минуты

SS – секунды

DC – десятые и сотые доли секунды

AAAAAA – порядковый номер в пределах сотой доли секунды

7.3.36 String getMessageText()

Возврат текста сообщения.

Возвращает:

- объект класса String, текст сообщения, хранящийся в объекте.

7.3.37 void setMessageText(String messageText)

Установка текста сообщения.

Параметры:

- messageText - объект класса String, текст сообщения.

8 ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

8.1 Примеры использования в каталоге examples.java

Дистрибутив интерфейса содержит примеры использования библиотеки в каталоге examples.java в формате Java-проекта Eclipse, см. [2].

Примеры находятся в пакете `ru.stmq.api.tests` и включают следующее**:**

Test.java — пример отправки и получения сообщения в среде J2EE;

TestSendFromFile.java — пример отправки и сообщения в среде J2EE с текстом из файла saimessage01.txt

TestStandalone.java — пример отправки и получения сообщения программой, запускаемой через java.exe.

8.2 Другие примеры использования

В этот раздел вошли исходные тексты примеров программ в среде J2EE, не включенные в проект examples.java, чтобы не нагружать его дополнительными зависимостями.

8.2.1 Сервлет для отправки сообщения

```
package ru.stmq.api.tests.web;

import java.io.IOException;
import java.io.PrintWriter;

import javax.annotation.Resource;
```

```
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import ru.ts.STMQ.api.j2ee.STMQInterface;
import ru.ts.STMQ.api.j2ee.STMQMessage;

/**
 * Servlet implementation class PutMessage
 */
@WebServlet("/putMessage")
public class PutMessage extends HttpServlet
{
     private static final long serialVersionUID = 1L;
     private static final Logger logger =
LoggerFactory.getLogger(PutMessage.class);
     @Resource (name="applAddress")   String applAddress   =
"RU.17:TESTJW";
     @Resource (name="targetAddress") String targetAddress =
"RU.17:DODRV";
     @Resource (name="messageText")   String messageText   =
"(:1111:)";
     @Resource (name="jms/qmgr")   ConnectionFactory  qmgr;
     @Resource (name="jms/qput") Destination qput;
     @Resource (name="jms/qans") Destination qans;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public PutMessage()
    {
        super();
    }

     /**
      * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
      */
     protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
     {
          request.setCharacterEncoding("UTF-8");
          response.setCharacterEncoding("UTF-8");
          response.setContentType("text/plain");
```

```
          PrintWriter out = response.getWriter();
          out.println("Отправитель: "+applAddress);
          out.println("Адресат: "+targetAddress);
          out.println("Текст сообщения: "+messageText);
     try
     {
          STMQInterface STMQInterface = new STMQInterface(qmgr,
null, qput, qans, applAddress);
          STMQMessage STMQMessage =
STMQInterface.createMessage(messageText);
          STMQMessage.setTo(targetAddress);
          STMQInterface.put(STMQMessage);
          STMQInterface.close();
          out.println("Сообщение успешно отправлено");
     }
     catch (Exception e)
     {
          logger.error("Ошибка при отправке сообщения АОС-2 !",e);
          out.println("Ошибка при отправке сообщения АОС-2 !");
          e.printStackTrace(out);
     }
     }
}
```

8.2.2 Сервлет для чтения сообщения

```
package ru.stmq.api.tests.web;

import java.io.IOException;
import java.io.PrintWriter;

import javax.annotation.Resource;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import ru.ts.STMQ.api.j2ee.STMQInterface;
import ru.ts.STMQ.api.j2ee.STMQMessage;

/**
 * Servlet implementation class GetMessage
 */
@WebServlet("/getMessage")
public class GetMessage extends HttpServlet
{
```

```
     private static final long serialVersionUID = 1L;
     private static final Logger logger =
LoggerFactory.getLogger(GetMessage.class);
     @Resource (name="applAddress")   String applAddress   =
"RU.17:TESTJW";
     @Resource (name="jms/qmgr")   ConnectionFactory  qmgr;
     @Resource (name="jms/qget") Destination qget;
     @Resource (name="jms/qput") Destination qput;
     @Resource (name="jms/qans") Destination qans;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public GetMessage()
    {
        super();
    }

     /**
      * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
      */
     protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
     {
          request.setCharacterEncoding("UTF-8");
          response.setCharacterEncoding("UTF-8");
          response.setContentType("text/plain");
          PrintWriter out = response.getWriter();
     try
     {
          STMQInterface STMQInterface = new STMQInterface(qmgr,
qget, qput, qans, applAddress);
          STMQMessage STMQMessage = STMQInterface.get(1000);
          if ( STMQMessage == null ) out.println("Не получено ни
одного сообщения.");
          else
          {
               out.println("Отправитель: "+STMQMessage.getFrom());
               out.println("Адресат: "+STMQMessage.getTo());
               out.println("Текст сообщения:
"+STMQMessage.getMessageText());
               STMQInterface.confirm(STMQMessage, true);
          }
          STMQInterface.close();
     }
     catch (Exception e)
     {
          logger.error("Ошибка при получении сообщения АОС-
2 !",e);
          out.println("Ошибка при получении сообщения АОС-2 !");
```

```
          e.printStackTrace(out);
     }
     }

}
```

8.2.3 Message-Driven Bean для чтения сообщения и отправки ответа.

```
package ru.stmq.api.tests.ejb;

import javax.annotation.Resource;
import javax.ejb.EJBException;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.Message;
import javax.jms.MessageListener;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import ru.ts.STMQ.api.j2ee.STMQInterface;
import ru.ts.STMQ.api.j2ee.STMQMessage;

/**
 * Message-Driven Bean implementation class for: SampleMDB
 */
@MessageDriven
public class SampleMDB implements MessageListener,
MessageDrivenBean
{

     /**
      *
      */
     private static final long serialVersionUID =
2533549267441804920L;
     private static final Logger logger =
LoggerFactory.getLogger(SampleMDB.class);
     @Resource (name="applAddress") String applAddress =
"RU.17:TESTJW";
     @Resource (name="jms/qmgr")   ConnectionFactory  qmgr;
     @Resource (name="jms/qput") Destination qput;
     @Resource (name="jms/qans") Destination qans;

    public SampleMDB()
    {
     try
     {
     }
```

```
     catch (Exception e)
     {
          logger.error("Ошибка при инициализации интерфейса с АОС-
2 (STMQInterface)",e);
     }
    }

     /**
     * @see MessageListener#onMessage(Message)
     */
    public void onMessage(Message message)
    {
     try
     {
          STMQInterface STMQInterface = new STMQInterface(qmgr,
null, qput, qans, applAddress);
          STMQMessage STMQMessage = STMQInterface.parse(message);
          String messageText = STMQMessage.getMessageText();
          String fromAddr = STMQMessage.getFrom();
          logger.info("Получено сообщение от '{}'. Текст
сообщения: '{}'", fromAddr, messageText);
          STMQInterface.confirm(STMQMessage, true);
     }
     catch (Exception e)
     {
          logger.error("Ошибка при обработке сообщения АОС-2.",e);
     }
    }

     @Override
     public void ejbRemove() throws EJBException
     {
     }

     @Override
     public void setMessageDrivenContext(MessageDrivenContext mdx)
throws EJBException
     {
     }
}
```

9 ИСПОЛЬЗОВАННЫЕ ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ОС	Операционная система
API	Application programming interface (прикладной программный интерфейс)
JMS	Java Messaging Service - стандарт Java интерфейса обмена сообщениями

JRE	Java Runtime Environment - среда выполнения программ на Java
JTA	Java Transaction API — стандарт Java интерфейса с менеджером транзакций
SLF4J	Simple Logging Facade for Java - интерфейс журналирования

10 ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

[1] Apache ActiveMQ Artemis User Manual – Using JMS // http://activemq.apache.org/components/artemis/documentation/1.0.0/using-jms.html
[2] Eclipse Documentation // https://www.eclipse.org/documentation/
[3] Java Documentation // https://docs.oracle.com/en/java/
[4] Java Message Service (JMS) // https://www.oracle.com/java/technologies/java-message-service.html
[5] What is JRE? // https://www.javatpoint.com/java-jre
[6] Java Transaction API // https://www.oracle.com/java/technologies/jta.html
[7] Simple Logging Facade for Java (SLF4J) // http://www.slf4j.org/
[8] .TGZ File Extension // https://fileinfo.com/extension/tgz
[9] WildFly Documentation // https://docs.wildfly.org/

СОСТАВИЛИ

Наименование подразделения	Должность исполнителя	Фамилия, имя, отчество	Подпись	Дата
	Старший системный архитектор	Казаков В.В.		
	Главный системный архитектор	Мишустин М.Б.		
	Руководитель проекта	Сологуб А.И.		