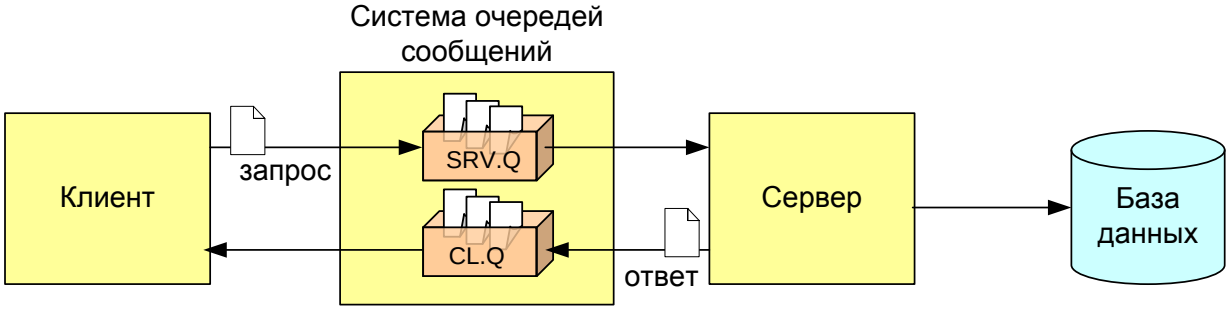
Лабораторная работа №5
Построение клиент-серверной архитектуры на базе технологии МОМ

Содержание

1. Лабораторная работа 5

1.1. Постановка задачи

В информационной системе, разработанной в рамках лабораторной работы №3, организовать асинхронное взаимодействие клиента и сервера через очереди сообщений.



1.2. Требования к работе

Взаимодействие клиента и сервера осуществляется по схеме «точка-точка» (поддержка одновременных соединений с несколькими клиентами не требуется) в асинхронном режиме:

- запускается клиент, отправляет серверу несколько запросов и завершает свою работу;
- запускается сервер, обрабатывает запросы клиента, отправляет клиенту ответы и завершает свою работу;
- вновь запускается клиент и получает ответы от сервера.

Взаимодействие между клиентом и сервером осуществляется через *систему очередей сообщений*. В системе очередей сообщений требуется создать две *очереди*:

- SRV.Q – серверная очередь, предназначенная для хранения запросов, поступивших от клиента
- CL.Q – клиентская очередь, предназначенная для хранения ответов на запросы клиента

Сервер запускается в фоновом режиме и не имеет пользовательского интерфейса. Сервер должен обеспечивать выполнение операций, представленных в вашем варианте задания.

На каждый запрос клиента сервер должен отправлять ответ. Ответы сервера должны различаться в зависимости от результата выполнения запроса клиента (положительный/отрицательный ответ).

Клиентское приложение отправляет запросы серверу и демонстрирует ответы пользователю. Клиентское приложение не требует создания пользовательского интерфейса. Тестирование работоспособности клиента осуществляется на основе сценариев, демонстрирующих возможности программы.

Формат информационных сообщений, которыми обмениваются клиент и сервер, определяется в соответствии с номером варианта.

Рекомендуемый язык программирования – Java.

Рекомендуемая система очередей сообщений – IBM WebSphere MQ v7.1.

1.3. Требования к оформлению отчета

Отчет должен содержать:

- титульный лист
- постановку задачи
- исходный код программы
- описание программы
- описание форматов информационных сообщений, которыми обмениваются клиент и сервер

1.4. Варианты заданий

Вариант 1	
Предметная область	Карта мира
Объекты	Страны, Города
Примечание	Карта мира содержит множество *стран*. Для каждой *страны* определено множество *городов*.
Требуемые операции	1. Добавление новой страны 2. Добавление нового города в заданную страну 3. Удаление города 4. Выдача полного списка городов с указанием названия страны
Формат сообщений	Строка с разделителем

Вариант 2	
Предметная область	Библиотека
Объекты	Авторы, Книги
Примечание	Книги в библиотеке сгруппированы по *авторам*. У каждого *автора* имеется множество *книг*.
Требуемые операции	1. Добавление нового автора 2. Добавление новой книги для автора 3. Удаление книги 4. Выдача полного списка книг с указанием ФИО автора
Формат сообщений	Двоичный

Вариант 3	
Предметная область	Отдел кадров
Объекты	Подразделения, Сотрудники
Примечание	Имеется множество *подразделений* предприятия. В каждом *подразделении* работает множество *сотрудников*.
Требуемые операции	1. Добавление нового подразделения 2. Прием на работу сотрудника в заданное подразделение 3. Увольнение сотрудника 4. Выдача списка сотрудников для заданного подразделения 5. Выдача списка подразделений
Формат сообщений	Строка с разделителем

Вариант 4	
Предметная область	Учебный отдел
Объекты	Группы, Студенты
Примечание	Имеется множество учебных *групп*. Каждая группа включает в себя множество *студентов*.
Требуемые операции	1. Добавление новой группы 2. Зачисление студента в заданную группу 3. Отчисление студента 4. Выдача полного списка студентов с указанием названия группы
Формат сообщений	Двоичный

Вариант 5	
Предметная область	Автосалон
Объекты	Производители автомобилей, Марки
Примечание	*Марки* автомобилей сгруппированы по производителям. У каждого *производителя* имеется множество *марок*.
Требуемые операции	1. Добавление производителя 2. Добавление новой марки заданного производителя 3. Удаление марки 4. Выдача полного списка марок с названием производителя
Формат сообщений	Строка с разделителем

Вариант 6	
Предметная область	Агентство новостей
Объекты	Категории новостей, Новости
Примечание	Новости сгруппированы по *категориям*. У каждой *категории* имеется множество *новостей*.
Требуемые операции	1. Добавление новой категории 2. Добавление новости заданной категории 3. Удаление новости 4. Выдача полного списка новостей 5. Выдача полного списка категорий
Формат сообщений	Двоичный

Вариант 7	
Предметная область	Продуктовый магазин
Объекты	Категория продукта, Продукт
Примечание	*Продукты* в магазине сгруппированы по *категориям*. Для каждой *категории* определено множество *продуктов*.
Требуемые операции	1. Добавление новой категории 2. Добавление продукта заданной категории 3. Удаление продукта 4. Выдача списка продуктов заданной категории 5. Выдача списка категорий
Формат сообщений	Строка с разделителем

Вариант 8	
Предметная область	Футбол
Объекты	Команды, Игроки
Примечание	Имеется множество футбольных *команд*. Для каждой *команды* определено множество *игроков*.
Требуемые операции	1. Добавление новой команды 2. Добавление нового игрока в команду 3. Удаление игрока 4. Выдача полного списка игроков с указанием названия команды
Формат сообщений	Двоичный

Вариант 9	
Предметная область	Музыкальный магазин
Объекты	Исполнители, Альбомы
Примечание	В музыкальном магазине *альбомы* сгруппированы по *исполнителям*. Для каждого *исполнителя* задано множество *альбомов*.
Требуемые операции	1. Добавление нового исполнителя 2. Добавление нового альбома заданного исполнителя 3. Удаление альбома 4. Выдача полного списка альбомов с указанием исполнителя
Формат сообщений	Строка с разделителем

Вариант 10	
Предметная область	Аэропорт
Объекты	Авиакомпании, Рейсы
Примечание	Имеется множество *авиакомпаний*. Для каждой *авиакомпании* определены ее *рейсы*.
Требуемые операции	1. Добавление новой компании 2. Добавление нового рейса 3. Отмена (удаление) рейса 4. Выдача полного списка рейсов с указанием названия авиакомпании
Формат сообщений	Двоичный

Вариант 11	
Предметная область	Файловая система
Объекты	Папки, Файлы
Примечание	Имеется множество *папок* (независимых друг от друга). Для каждой *папки* определено множество *файлов*.
Требуемые операции	1. Добавление новой папки 2. Добавление нового файла в заданную папку 3. Удаление файла 4. Выдача полного списка папок 5. Выдача списка файлов для заданной папки
Формат сообщений	Строка с разделителем

Вариант 12	
Предметная область	Расписание занятий
Объекты	Дни недели, Занятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *занятий*.
Требуемые операции	1. Добавление нового дня 2. Добавление нового занятия 3. Удаление занятия Выдача полного списка занятий с указанием дня
Формат сообщений	Двоичный

Вариант 13	
Предметная область	Записная книжка
Объекты	Календарные дни, Мероприятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *мероприятий*.
Требуемые операции	1. Добавление нового дня 2. Добавление нового мероприятия 3. Удаление мероприятия 4. Выдача полного списка мероприятий с указанием дня
Формат сообщений	Строка с разделителем

Вариант 14	
Предметная область	Видеомагазин
Объекты	Жанры, Фильмы
Примечание	Имеется множество *жаров*. Для каждого *жанра* определен перечень *фильмов*.
Требуемые операции	1. Добавление нового жанра 2. Добавление нового фильма 3. Удаление фильма 4. Выдача списка фильмов заданного жанра 5. Выдача полного списка жанров
Формат сообщений	Двоичный

Вариант 15	
Предметная область	Железная дорога
Объекты	Дороги, Станции
Примечание	Имеется множество *железных дорог*. В ведомстве каждой дороги находится множество *станций*.
Требуемые операции	1. Добавление новой дороги 2. Добавление новой станции 3. Удаление станции 4. Выдача полного списка дорог 5. Выдача списка станций для заданной дороги
Формат сообщений	Строка с разделителем

Вариант 16	
Предметная область	Склад
Объекты	Секции, Товары
Примечание	*Товары* на складе сгруппированы по *секциям*. Для каждой *секции* задано множество *товаров*.
Требуемые операции	1. Добавление новой секции 2. Добавление нового товара в заданную секцию 3. Удаление товара 4. Выдача полного списка секций 5. Выдача списка товаров для заданной секции
Формат сообщений	Двоичный

Вариант 17	
Предметная область	Кафедра университета
Объекты	Преподаватели, Дисциплины
Примечание	На кафедре имеется множество *преподавателей*. Для каждого *преподавателя* задано множество *дисциплин*.
Требуемые операции	1. Прием на работу (добавление) нового преподавателя 2. Добавление новой дисциплины 3. Удаление дисциплины 4. Выдача полного списка преподавателей 5. Выдача списка дисциплин для заданного преподавателя
Формат сообщений	Строка с разделителем

Вариант 18	
Предметная область	Программное обеспечение
Объекты	Производители, Программные продукты
Примечание	Программные *продукты* сгруппированы по *производителям*. Для каждого *производителя* задано множество *продуктов*.
Требуемые операции	1. Добавление нового производителя 2. Добавление нового продукта 3. Удаление продукта 4. Выдача полного списка производителей 5. Выдача списка продуктов заданного производителя
Формат сообщений	Двоичный

Вариант 19	
Предметная область	Геометрия
Объекты	Многоугольники, Вершины
Примечание	Имеется множество *многоугольников*. Каждый *многоугольник* состоит из произвольного числа *вершин*.
Требуемые операции	1. Добавление нового многоугольника 2. Удаление многоугольника 3. Добавление новой вершины 4. Выдача полного списка многоугольников 5. Выдача полного списка вершин заданного многоугольника
Формат сообщений	Строка с разделителем

Вариант 20	
Предметная область	Схема метро
Объекты	Линии, Станции
Примечание	Имеется множество *линий* метрополитена. Каждая *линия* состоит из последовательности *станций*.
Требуемые операции	1. Добавление новой линии 2. Добавление новой станции 3. Удаление станции 4. Выдача списка станций для заданной линии. 5. Выдача полного списка линий
Формат сообщений	Двоичный

2. Введение в системы очередей сообщений

2.1. Основные понятия

Message Oriented Middleware (MOM) **–** «*промежуточное программное обеспечение, ориентированное на сообщения*» – общепринятое название программных средств, обеспечивающих взаимодействие приложений в распределенной системе на основе механизма обмена сообщениями. Понятие *MOM* также широко применяется для обозначения технологии обмена данными в форме сообщений.

Сообщение (англ. ***message***) – структура данных, используемая для представления информации, передаваемой между приложениями. Сообщение состоит из *заголовка* и *прикладной части*. В заголовке содержится служебная информация о сообщении. Заголовок сообщения обычно имеет фиксированный размер и строго определенную структуру. Прикладная часть может содержать любую информацию. Размер, структура и формат прикладной части определяется приложением, создающим сообщение.

Очередь (англ. ***queue***) – структура данных, предназначенная для промежуточного хранения информационных сообщений, передаваемых между приложениями. Очередь организует сообщения в порядке ***FIFO*** (***first in – first out***, сообщения хранятся в очереди в порядке своего поступления), либо в порядке приоритетов сообщений. Прикладные программы могут записывать сообщения в очередь, и считывать сообщения из очереди. Можно сказать, что очередь выступает в качестве «почтового ящика» для информации, передаваемой между приложениями.

Система очередей сообщений (англ. ***message queuing system***) – программное обеспечение категории MOM, управляющее очередями сообщений. Система очередей сообщений обеспечивает доступ к очереди со стороны прикладных программ, управляет операциями чтения и записи сообщений, и отвечает за физическое хранение данных, содержащихся в очереди. На сегодняшний день на мировом рынке программного обеспечения существует большой выбор систем очередей сообщений: WebSphere MQ компании IBM, MSMQ компании Microsoft, Advanced Queuing компании Oracle, FioranoMQ компании Fiorano Software, Sonic MQ компании Progress Software, Active MQ сообщества Apache и др.

2.2. Модели взаимодействия приложений

В распределенных информационных системах, построенных на базе средств MOM, приложения могут обмениваться информацией различными способами.

Ниже рассматривается несколько типовых моделей взаимодействия приложений с использованием очередей сообщений.

2.2.1. Односторонняя передача данных

Один из самых простых способов взаимодействия – это ***односторонняя передача данных***. На рис. 1 показана односторонняя передача данных между двумя приложениями: A и B.

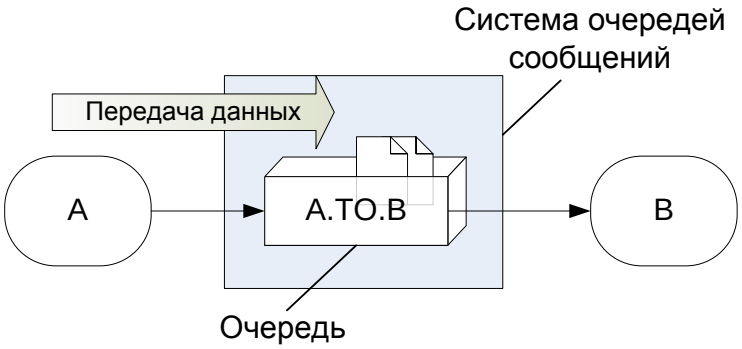


Рис. 1. Односторонняя передача данных

Приложение A помещает сообщения, содержащие некоторую информацию в очередь, которую обслуживает приложение B. Приложение B последовательно извлекает эти сообщения из очереди, и обрабатывает их.

Данную модель взаимодействия иногда называют моделью «отправил-забыл». Приложение A, отправляя сообщение, как будто забывает о нем. Оно не ждет ответного сообщения и не требует подтверждения доставки или обработки сообщения.

2.2.2. Модель «запрос-ответ»

Модель ***«запрос-ответ»*** предполагает двусторонний обмен информацией между приложениями. Приложение, отправляя сообщение, ожидает получения ответного сообщения.

На рис. 2 показано взаимодействие приложений A и B по принципу «запрос-ответ». Для каждой из программ в системе очередей сообщений создана очередь, которую они будут обслуживать.

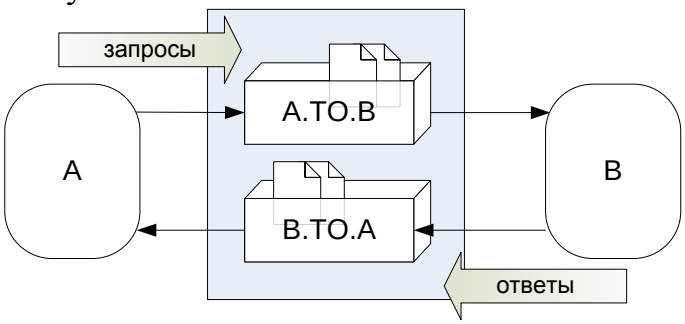


Рис. 2. Модель «запрос-ответ»

Приложение A, желая отправить некоторый запрос приложению B, помещает соответствующее сообщение в очередь приложения B (A.TO.B). Приложение B читает данное сообщение, обрабатывает запрос, содержащийся в сообщении, и помещает ответное сообщение в очередь приложения A (B.TO.A). Приложение A считывает этот ответ и узнает результат обработки своего запроса.

2.2.3. Синхронное и асинхронное взаимодействие

Взаимодействие по принципу «запрос-ответ» может осуществляться в синхронном или асинхронном режиме.

Синхронное взаимодействие, как следует из названия, предполагает последовательный обмен сообщениями между приложениями: одно приложение отправляет другому приложению запрос и ждет ответа на него; в том случае, если

в течение какого-то заданного времени ответ не получен, приложение-отправитель завершается ошибкой.

Синхронное взаимодействие имеет ряд ограничений. Во-первых, синхронный обмен данными возможен лишь при условии одновременной активности взаимодействующих приложений. Программы, выполняющиеся по различному временному регламенту, не могут общаться в синхронном режиме. Во-вторых, синхронному взаимодействию приложений может препятствовать отсутствие постоянной или качественной линии связи. В-третьих, приложение, участвующее в синхронном обмене информацией, после отправления запроса блокируется на время ожидания ответа и становится недоступным для других программ.

Асинхронное взаимодействие не предполагает блокировки приложений при ожидании ответов. Данный тип взаимодействия осуществляется следующим образом. Приложение А отправляет свой запрос приложению В и не ждет непременного ответа от него, а продолжает свое нормальное функционирование. После этого, приложение А будет периодически проверять, не пришло ли для него ответное сообщение. Параллельно, приложение А может отправить еще один запрос программе В, или участвовать во взаимодействии с другими программами. Приложение В в тот момент, когда сочтет нужным, обработает запрос от приложения А и отправит ему ответное сообщение.

В асинхронном взаимодействии могут принимать участие программы, которые выполняются в различное время. Кроме того, данный способ взаимодействия подходит и для обмена данными при наличии некачественной линии связи.

Следует отметить, что поддержка асинхронного взаимодействия является основным преимуществом технологии МОМ над технологиями взаимодействия приложений, в основе которых лежит синхронный механизм удаленного вызова процедур.

2.2.4. Модель «клиент-сервер»

Модель *«запрос-ответ»* является частным случаем общей модели ***«клиент-сервер»***. Модель *«клиент-сервер»* предполагает наличие централизованной программы – сервера, оказывающей некоторый набор услуг, а также множества программ – клиентов, запрашивающих эти услуги.

У сервера имеется централизованная очередь, в которую помещают свои запросы клиенты. У каждого клиента имеется своя персональная очередь, в которую сервер помещает ответы на запросы клиентов.

На рис. 3 показано взаимодействие трех программ: клиентов А и В и сервера С. Клиенты отправляют свои сообщения в очередь сервера TO.SERV. Сервер обрабатывает запросы клиентов, и возвращает персональный ответ клиенту в его очередь (TO.A или TO.B).

Клиент-серверное взаимодействие может осуществляться как в синхронном, так и в асинхронном режиме.

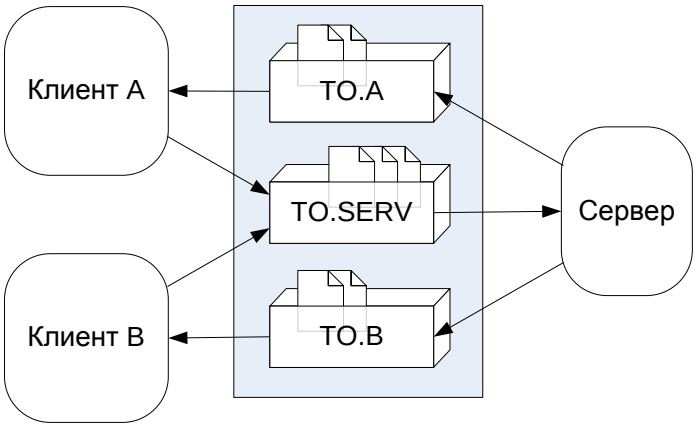


Рис. 3. Модель «клиент-сервер»

2.2.5. Модель «публикация-подписка»

«***Публикация-подписка***» – технология асинхронного взаимодействия, используемая для доставки сообщений внутри группы приложений.

Приложения, обменивающиеся информацией по технологии «публикация-подписка», подразделяются на две категории:

- ***публикаторы*** (англ. *publishers*) – приложения, которые распространяют информацию;
- ***подписчики*** (англ. *subscribers*) – приложения, которые заинтересованы в получении информации.

Взаимодействием *публикаторов* и *подписчиков* управляет специальная программа – *брокер сообщений (*англ. *message broker)*. Распределение сообщений между подписчиками осуществляется на основе *тем* сообщений. *Тема сообщения* (англ. *topic*) – специальный идентификатор, позволяющий отнести сообщение к той или иной логической категории

Приложения-подписчики регистрируются внутри брокера сообщений, указывая при этом, какие темы сообщений их интересуют. Публикаторы отправляют брокеру информационные сообщения, с указанием темы этих сообщений. Брокер для каждого полученного сообщения определяет подписчиков и высылает им копию этого сообщения.

На рис. 4 показан пример взаимодействия приложений по схеме «*публикация-подписка*». Имеются два приложения-публикатора (A и B), и два приложения-подписчика (C и D). При этом приложение C заинтересовано в сообщениях по теме *x*, а приложение D по темам *x и y*. Сведения о подписчиках хранятся в базе данных брокера сообщений.

Приложение A публикует в системе новое сообщение по теме *x*, помещая его в очередь брокера PUB. Аналогичным образом, приложение B публикует сообщение по теме *y*. Брокер получает два новых сообщения, и записывает копии этих сообщений в очереди соответствующих подписчиков (TO.C и TO.D).

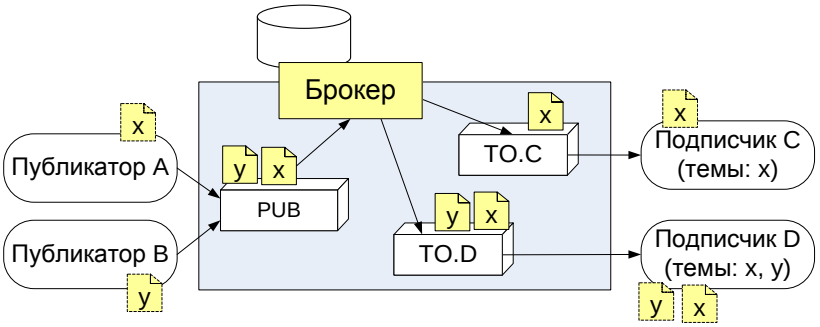


Рис. 4. Модель «публикация-подписка»

2.3. Система очередей сообщений WebSphere MQ

В данных методических указаниях в качестве средства управления сообщениями рассматривается программный продукт WebSphere MQ компании IBM. На сегодняшний день WebSphere MQ является одним из лидеров мирового IT-рынка в области технологий MOM.

WebSphere MQ - это система очередей сообщений, позволяющая строить надежные распределенные системы на основе разнородных приложений. WebSphere MQ может выполняться на большом количестве платформ (z/OS, OS/390, MVS, OS/400, Sun Solaris, Linux, AIX, HP-UX, VSE/ESA, OS/2, Windows 95/98, Windows NT/2000/XP и др.) и поддерживает большинство стандартных протоколов связи (в том числе TCP/IP, NetBIOS, LU62 и SPX).

WebSphere MQ обеспечивает хранение и транспортировку сообщений, гарантируя при этом доставку критически важной информации. Среди основных возможностей данной системы следует выделить наличие механизмов транзакционной обработки данных (WebSphere MQ может выступать в качестве менеджера распределенных транзакций), поддержку стандартных соглашений защиты каналов данных (в т.ч. SSL), а также наличие программных интерфейсов и библиотек для большинства современных языков и сред программирования.

2.4. Основные объекты WebSphere MQ

2.4.1. Менеджер очередей

Менеджер очередей (англ. ***queue manager***) – главный серверный компонент WebSphere MQ – служба, управляющая очередями сообщений и обработкой сообщений и принимающая вызовы от прикладных программ. Менеджер очередей также иногда называют *администратором очередей*.

В рамках одной системы очередей сообщений может быть создано несколько менеджеров очередей. У каждого менеджера имеется свое уникальное имя.

Очереди создаются в рамках менеджера очередей. Приложение, которому требуется обратиться к очереди для записи или чтения сообщений, должно предварительно установить соединение с соответствующим менеджером очередей WebSphere MQ.

Поддержку удаленных соединений с менеджером очередей обеспечивает специальная служба ***Listener*** («*получатель запросов*»). Listener выполняется для заданного протокола связи и определяет параметры соединения по этому протоколу. Для протокола TCP/IP Listener по умолчанию прослушивает порт 1414.

2.4.2. Очереди

Как уже говорилось выше, ***очереди*** предназначены для промежуточного хранения сообщений.

Физически очередь может быть представлена различными способами: очередь может храниться в оперативной памяти компьютера или на диске. В любом случае, управление очередями выполняется менеджером очередей и скрыто от приложений. Каждая очередь характеризуется названием, и имеет набор атрибутов, отвечающих за хранение и обработку сообщений.

Приложения получают доступ к очередям при помощи программных интерфейсов, в основе которых лежат вызовы ***MQI***[1], и могут выполнять следующие операции с очередью:

- открывать очередь для записи или чтения сообщений;
- помещать сообщение в очередь;
- извлекать сообщение из очереди;
- просматривать сообщения, находящиеся в очереди, без их извлечения;
- запрашивать и устанавливать атрибуты очереди;
- закрывать очередь.

Для того чтобы поместить сообщение в очередь или извлечь его из очереди, приложение должно предварительно открыть данную очередь на запись или на чтение сообщений, а после завершения работы с очередью - закрыть ее.

WebSphere MQ поддерживает различные типы очередей:

- *локальные очереди* (*local queues*) – обеспечивают хранение сообщений;
- *удаленные очереди* (*remote queues*) – используется для передачи сообщений на другие менеджеры очередей;
- *псевдоочереди* (*alias queues*) – выступают в качестве альтернативного имени для другой очереди;
- *модельные очереди* (*model queues*) – используются в качестве шаблона при создании других очередей.

При выполнении лабораторной работы, представленной в разделе 6, следует использовать локальные очереди.

2.4.3. Каналы

Канал (англ. ***channel***) – логическая линия связи, соединяющая удаленное приложение WebSphere MQ с менеджером очередей, а также два менеджера очередей между собой.

Каналы делятся на две категории:

- *Каналы MQI* (*MQI Channels*);
- *Каналы сообщений* (*Message Channels*).

Канал MQI обеспечивает взаимодействие между *клиентом WebSphere MQ* и менеджером очередей. Он предназначен для передачи вызовов MQI от клиентского приложения менеджеру очередей и ответов на эти вызовы от менеджера очередей клиенту (рис. 5). Другими словами, канал MQI используется приложением при помещении и извлечении сообщений из очереди.

---

[1] *Message Queue Interface (MQI)* – базовый низкоуровневый интерфейс WebSphere MQ, предназначенный для общения прикладной программы с менеджером очередей

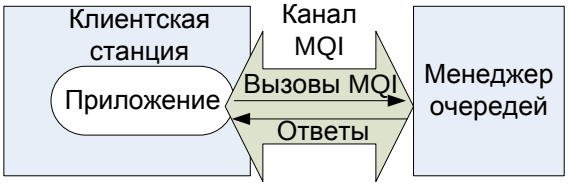


Рис. 5. Канал MQI

Канал сообщений используется для передачи сообщений между двумя менеджерами очередей. При этом канал сообщений, в отличие от канала MQI, является однонаправленным, т.е. позволяет передавать сообщения в строго определенном направлении (рис. 6).

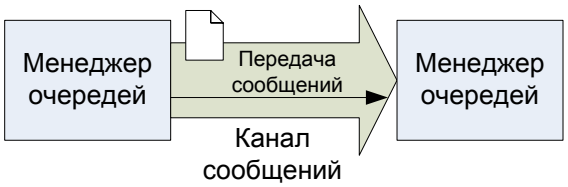


Рис. 6. Канал сообщений

2.5. Установка WebSphere MQ

Установка продукта IBM WebSphere MQ осуществляется из графического приложения *WebSphere MQ Installation LaunchPad* (*Панель установки WebSphere MQ*). Установку следует выполнять от имени пользователя, **обладающего правами локального администратора ОС**. Обратите внимание, что имя пользователя **не должно содержать кириллических символов**.

Для запуска *LaunchPad* необходимо выполнить файл *Setup.exe* из папки дистрибутива WebSphere MQ.

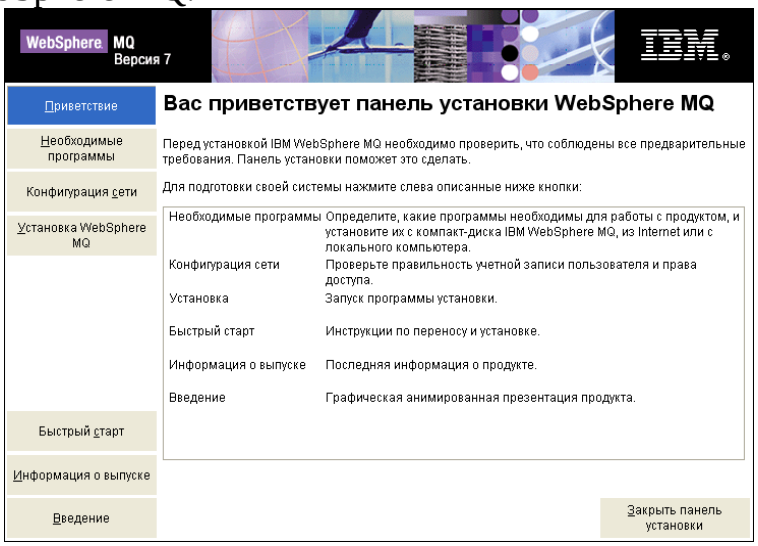


Рис. 7. WebSphere MQ LaunchPad

На закладке *Необходимые программы* (*Software Requirements*) предлагается установить дополнительное программное обеспечение, необходимое для WebSphere MQ (рис. 8): *Service Pack* для операционной системы (в зависимости от типа ОС Windows) и графическую платформу *Eclipse*. При этом *LaunchPad* автоматически определяет недостающее ПО и помечает его в списке необходимых программ надписью "*Не установлен*". Для установки *Eclipse* нажмите кнопку «*Компакт-диск*».

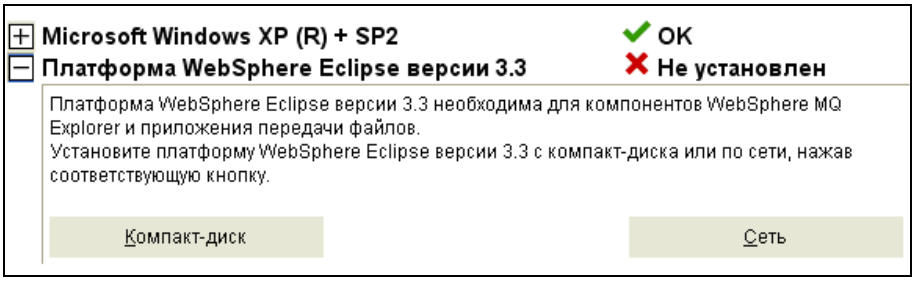


Рис. 8. Установка дополнительных компонентов

На закладке *Конфигурация сети* (*Network Configuration*) необходимо определить, будет ли сервер WebSphere MQ выполняться в домене Microsoft Windows или нет. На данном шаге рекомендуется выбрать опцию «*Нет*» (рис. 9).

На закладке *Установка WebSphere MQ* (*WebSphere MQ Installation*) отображается информация о готовности системы к установке продукта. В том случае, если вы установили необходимое дополнительное программное обеспечение и определили параметры конфигурации сети, выберите язык установки и нажмите кнопку "*Запустить программу установки IBM WebSphere MQ*".

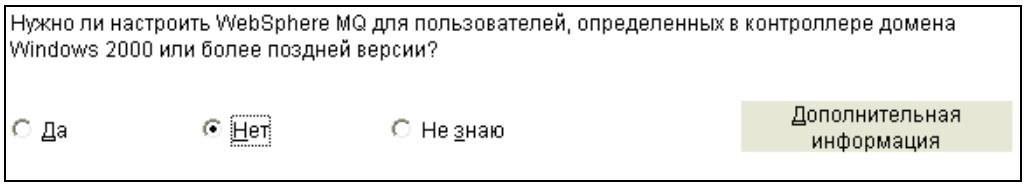


Рис. 9. Настройка конфигурации сети в LaunchPad

На экране появится *мастер установки WebSphere MQ*. Оставив значения всех параметров по умолчанию, дойдите до последней страницы мастера и нажмите кнопку "*Установить*". После этого будет запущен процесс установки системы на ваш компьютер.

После установки WebSphere MQ автоматически будет запущен *мастер подготовки WebSphere MQ* (*Prepare MQ Wizard*). Этот мастер позволяет настроить параметры работы сервера WebSphere MQ в сети, запустить основные сервисы WebSphere MQ, а также создать конфигурацию системы по умолчанию.

На определенном шаге мастер может задать вам вопрос о наличии контроллера доменов в вашей сети. Рекомендуется ответить на данный вопрос отрицательно, как показано на рис. 10.

2.6. Создание менеджера очередей

В случае если менеджер очередей не был создан автоматически при установке WebSphere MQ, его потребуется создать вручную.

Для создания нового менеджера очередей следует использовать утилиту *WebSphere MQ Explorer* (*Обозреватель WebSphere MQ*). *WebSphere MQ Explorer* – это графическое средство администрирования WebSphere MQ, разработанное на базе платформы Eclipse. Данная утилита устанавливается на компьютер вместе с продуктом WebSphere MQ.

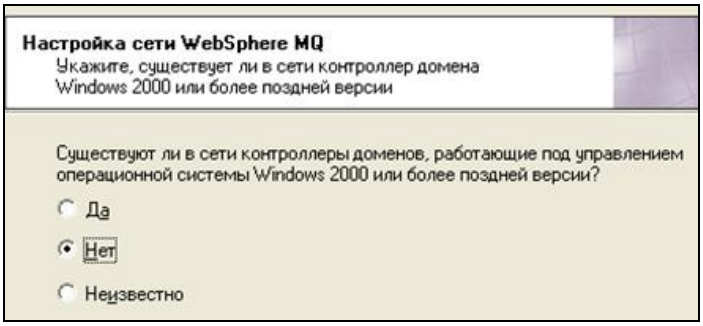


Рис. 10. Настройка конфигурации сети в «мастере подготовки»

Список менеджеров очередей представлен в утилите в окне *Навигатор* в папке *Администраторы очередей*. Для создания нового менеджера очередей вызовите контекстное меню на данной папке и вызовите команду *Создать* (рис.11). В появившемся мастере укажите название менеджера очередей. Для остальных параметров оставьте значения по умолчанию.

На последнем шаге мастера (шаг 4) обратите внимание на параметр *Рабочий порт*. Данный параметр задает порт TCP, который будет прослушивать служба *Listener* для создаваемого менеджера очередей. Данный номер порта потребуется указать в клиентской программе при подключении к менеджеру очередей.

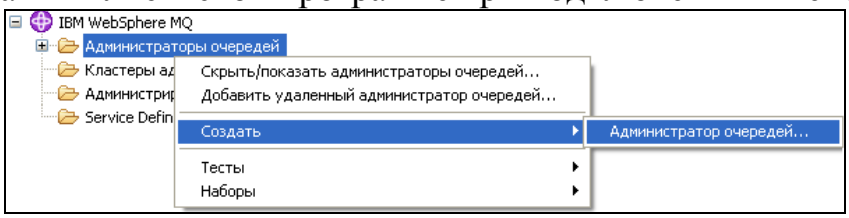


Рис. 11. Создание менеджера очередей

2.7. Создание очереди

Чтобы создать новую *локальную очередь*, откройте утилиту WebSphere MQ Explorer, выберите в дереве *Навигатора* внутри нужного менеджера очередей папку *Очереди* и вызовите для нее контекстное меню. В меню выберите команду *Создать → Локальная очередь* (рис. 12).

На экране появится диалоговое окно создания очереди. Укажите название очереди и нажмите кнопку *Готово*.

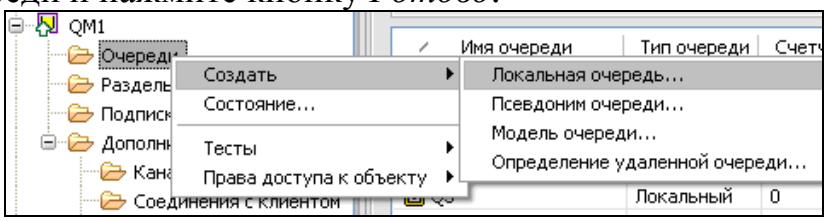


Рис. 12. Создание очереди

3. Разработка приложений для WebSphere MQ

Для разработки приложений, работающих с системой WebSphere MQ, может использоваться ряд стандартных программных интерфейсов, в частности:

- *Message Queue Interface (MQI)* – базовый низкоуровневый интерфейс WebSphere MQ для языков программирования C/C++ и COBOL;
- *Base Java MQ* – базовый интерфейс WebSphere MQ для языка Java;

- *Java Message Service* (*JMS*) – универсальный интерфейс работы с сообщениями (спецификация J2EE);
- *WebSphere MQ classes for .NET* – интерфейс WebSphere MQ для платформы Microsoft .NET.

Перечисленные выше интерфейсы входят в состав продукта WebSphere MQ.

В данных методических указаниях рассматривается интерфейс *Base Java MQ*. Пример программы, использующей *Base Java MQ*, вы можете найти в каталоге установки WebSphere MQ:

\Program Files\IBM\WebSphere MQ\tools\wmqjava\samples\MQSample.java

3.1. Подготовка Java-проекта для Base Java MQ

Классы Base Java MQ содержатся в JAR-файлах *com.ibm.mq.jar* и *com.ibm.mq.jmqi.jar*, размещающихся в директории

\Program Files\IBM\WebSphere MQ\java\lib .

Подключите данные JAR-файлы к вашему проекту Java, чтобы работать с системой WebSphere MQ (рис. 13).

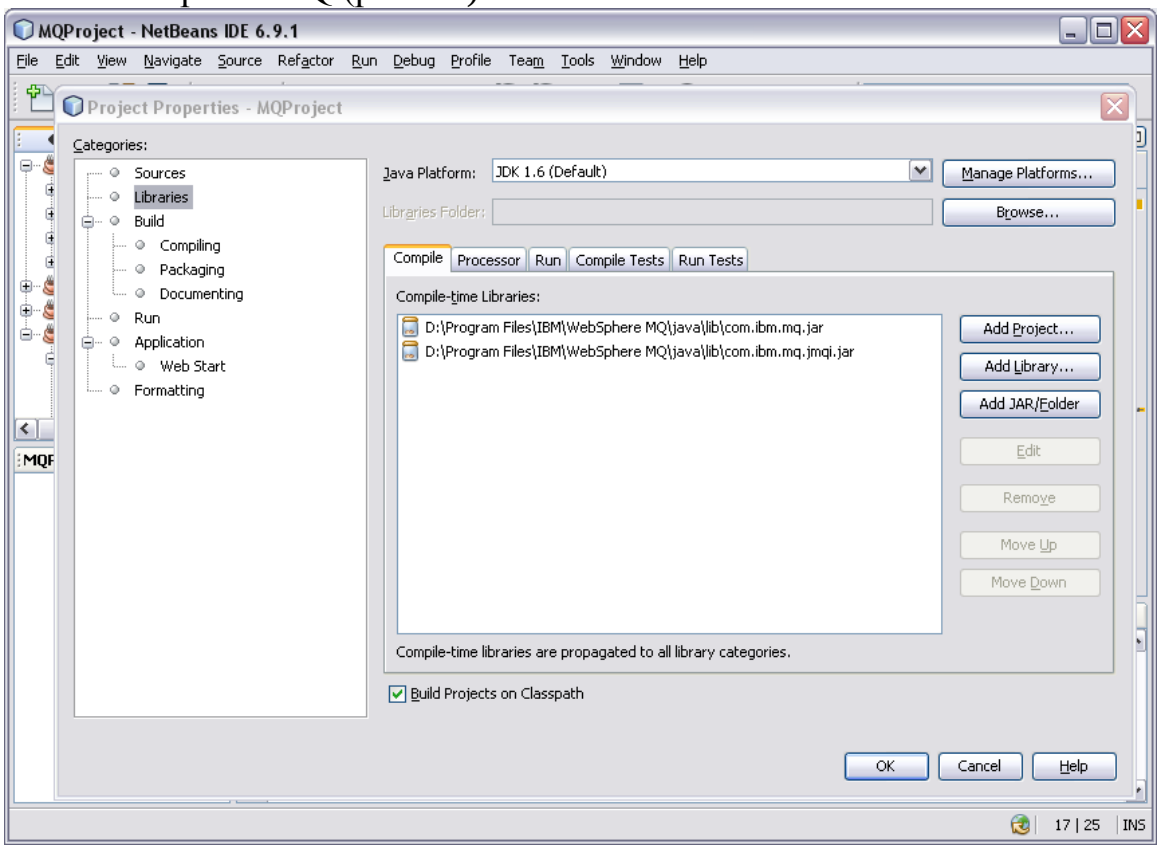


Рис. 13. Подключение модулей MQ к проекту в NetBeans IDE

К программным модулям подключите библиотеку com.ibm.mq:

```
import com.ibm.mq.*;
```

3.2. Основные классы Base Java MQ

Библиотека com.ibm.mq содержит следующие основные классы, обеспечивающие работу с системой WebSphere MQ:

- *MQEnvironment* – используется для настройки параметров соединения с менеджером очередей;
- *MQQueueManager* – класс «менеджер очередей» – позволяет установить соединение с менеджером очередей и обращаться к очередям;
- *MQQueue* – класс «очередь» – позволяет отправлять и принимать сообщения;
- *MQMessage* – класс «сообщение» – позволяет формировать и считывать содержимое прикладной части и заголовка сообщения;
- *MQGetMessageOptions* – класс «опции чтения» – задает дополнительные параметры для операции чтения сообщения из очереди;
- *MQPutMessageOptions* – класс «опции записи» – задает дополнительные параметры для операции записи сообщения в очередь;
- *MQMD* – класс, описывающий заголовок сообщения;
- *MQC* – содержит константы WebSphere MQ;
- *MQException* – исключение WebSphere MQ.

3.3. Настройка параметров соединения

Для настройки параметров соединения с менеджером очередей используются статические поля класса *MQEnvironment*:

- *hostname* – имя хоста или IP-адрес;
- *port* – номер порта TCP/IP;
- *channel* – название канала MQI;
- *CCSID* – кодовая страница, используемая для представления данных.

Если разрабатываемое приложение планируется запускать на том же компьютере, где выполняется менеджер очередей, установите для параметра *hostname* значение *null*. В этом случае значения параметров *port*, *channel* и *CCSID* будут игнорироваться, а соединение между приложением и менеджером очередей будет устанавливаться напрямую, без использования сетевых протоколов.

```
MQEnvironment.hostname = null;
```

Для взаимодействия с менеджером очередей по *каналу MQI* (при подключении к удаленному менеджеру очередей) следует задать ненулевое значение для параметра *hostname*, а также указать название канала MQI в параметре *channel* (используйте канал по умолчанию с названием *SYSTEM.DEF.SVRCONN*). В том случае, если порт TCP/IP, прослушиваемый программой *Listener* на менеджере очередей, отличен от порта по умолчанию (1414), следует указать номер этого порта в параметре *port*. При необходимости укажите также кодовую страницу клиента (*CCSID*).

```
MQEnvironment.hostname = "localhost";
MQEnvironment.port = 1414;
MQEnvironment.channel = "SYSTEM.DEF.SVRCONN";
MQEnvironment.CCSID = 866;
```

3.4. Обработка ошибок

Для обработки ошибок, возникающих при работе с WebSphere MQ, используется класс *MQException*. Класс *MQException* унаследован от класса *java.lang.Exception* и используется в конструкции *try – catch.*

Класс имеет два основных поля:

- *completionCode* – код завершения операции
- *reasonCode* – код ошибки

Поле *completionCode* может принимать значения:

- *MQException.MQCC_WARNING* – операция выполнилась с предупреждениями
- *MQException.MQCC_FAILED* – операция завершилась ошибкой

Поле *reasonCode* используется для уточнения причины возникновения ошибки.

Класс также содержит метод *getMessage*(), возвращающий сообщение об ошибке.

```
try
{
   // операции с объектами WebSphere MQ
   ...
}
catch(MQException e)
{  // Если возникла ошибка
   System.out.println(e.getMessage());
   if (e.reasonCode == ...)
   {
      ...
   }
   else if (e.reasonCode = ...) ...
   else ...
}
```

3.5 Установка соединения с менеджером очередей

Соединением с менеджером очередей управляет класс *MQQueueManager*.

Для установки соединения с менеджером очередей следует создать объект класса *MQQueueManager*. При этом в конструктор класса передается название менеджера очередей.

```
public MQQueueManager(String queueManagerName)
                              throws MQException
```

Параметры соединения определяются при создании экземпляра класса *MQQueueManager* на основе значений полей класса *MQEnvironment*.

Для разрыва соединения используется метод класса *disconnect*

```
public void disconnect() throws MQException
```

Пример:

```
// Установка соединения с менеджером QM1
MQQueueManager QM = new MQQueueManager("QM1");
... // работа с менеджером очередей
...
// Завершение соединения
QM.disconnect();
```

3.6. Открытие очереди

Чтобы отправлять и читать сообщения, необходимо предварительно открыть нужную очередь. Для открытия очереди используется метод *accessQueue* экземпляра класса *MQQueueManager*.

```
public MQQueue accessQueue(String queueName,
       int openOptions)  throws  MQException
```

В метод *accessQueue* передается название очереди *queueName* и опции открытия очереди *openOptions*.

Для определения опций открытия очереди используются константы *MQOO_** класса *MQC*:

- *MQC.MQOO_OUTPUT* – открытие очереди для записи сообщений;
- *MQC.MQOO_BROWSE* – открытие очереди для просмотра сообщений;
- *MQC.MQOO_INPUT_EXCLUSIVE* – открытие очереди для извлечения сообщений (эксклюзивный режим, только один процесс может читать сообщения из очереди);
- *MQC.MQOO_INPUT_SHARED* – извлечение сообщений (режим общего доступа);
- *MQC.MQOO_INPUT_AS_Q_DEF* – извлечение сообщений (режим по умолчанию).

На основе представленных констант необходимо сформировать целочисленное значение *openOptions*, используя побитовую логическую операцию *«или»*.

Например, если мы хотим открыть очередь для чтения и просмотра сообщений, в качестве параметра *openOptions* можно указать следующее значение: `MQC.MQOO_INPUT_EXCLUSIVE | MQC.MQOO_BROWSE`.

В случае успешного выполнения метод *accessQueue* возвращает экземпляр класса *MQQueue*.

По завершению работы с очередью ее следует закрыть при помощи метода *close*:

```
public void close() throws MQException
```

В следующем примере мы установим соединение с менеджером очередей *QM1* и откроем очередь *MY.QUEUE* для чтения и записи сообщений:

```
// Установка соединения с менеджером QM1
MQQueueManager QM = new MQQueueManager("QM1");
...
// Настройка опций открытия очереди
int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                  MQC.MQOO_OUTPUT;
// Открытие очереди
MQQueue Q =
       QM.accessQueue("MY.QUEUE", openOptions);
... // работа с очередью
...
// Завершение работы
Q.close();
QM.disconnect();
```

3.7. Работа с сообщением

Для работы с сообщениями WebSphere MQ используется класс *MQMessage*. Данный класс позволяет формировать прикладную часть сообщения и работать с его заголовком. Класс *MQMessage* используется при записи сообщений в очередь и при чтении сообщений из очереди.

Работа с прикладной частью сообщения осуществляется через механизм потокового ввода-вывода при помощи методов вида *readXXX* (чтение данных из прикладной части) и *writeXXX* (запись данных в прикладную часть). Эти методы определены в стандартных Java-интерфейсах *DataInput* и *DataOutput*.

Ниже приводятся прототипы некоторых методов класса *MQMessage*:

```
// Прочитать все сообщение в двоичном виде
public void readFully(byte[] b)
                      throws IOException;
// Прочитать целое число
public int readInt() throws IOException;
// Прочитать строку
public String readLine() throws IOException;
// Прочитать заданное количество символов
public String readString(int length)
                         throws IOException;
// Записать целое число
public void writeInt(int v) throws IOException;
// Записать строку
public void writeString(String s)
                        throws IOException;
// Очистить прикладные данные
public void clearMessage()
                        throws IOException;
// Позиционирование внутри сообщения
public void seek(int offset)
                 throws EOFException;
// Пропустить заданное количество байт
public int skipBytes(int n)
           throws IOException, EOFException;
// Возвращает оставшееся количество байт
public int getDataLength() throws IOException;
// Возвращает текущую позицию
public int getDataOffset()
                         throws IOException;
// Возвращает длину прикладной части сообщения
public int getMessageLength()
                        throws IOException;
```

Параметры заголовка сообщения описывается классом *MQMD*. Класс *MQMessage* унаследован от класса *MQMD* и содержит поля, позволяющие работать с заголовком сообщения, в частности:

- *messageID* – уникальный идентификатор сообщения (*byte*[]);
- *correlationID* – идентификатор связи, позволяет объединить несколько сообщений в группу (*byte*[]);
- *format* – формат прикладной части сообщения (*String*);
- *messageType* – тип сообщения (*int*);

- *persistence* – «постоянство» сообщения (*int*) – определяет, будет ли сообщение сохранено на жесткий диск при записи в очередь, или оно будет храниться в оперативной памяти;
- *priority* (*int*) – приоритет сообщения.

В следующем примере мы запишем в прикладную часть сообщения *m* приветственную строку и ее длину:

```
MQMessage m = new MQMessage();
String s = "Hello, world!";
m.writeInt(s.length());
m.writeString(s);
```

В следующем примере мы извлечем из сообщения *m* строку и выведем ее на экран:

```
int l = m.readInt();
String s = m.readString(l);
System.out.print(s);
```

3.8. Запись сообщения в очередь

Для записи сообщения в очередь используется метод *put* класса *MQQueue*. В метод передается ссылка на сообщение *MQMessage*, которое необходимо поместить в очередь:

```
public void put(MQMessage message)
                throws MQException;
```

В следующем примере мы запишем тестовое сообщение в очередь *MY.QUEUE*:

```
MQQueueManager QM = new MQQueueManager("QM1");
MQQueue Q =
    QM.accessQueue("MY.QUEUE", MQC.MQOO_OUTPUT);
MQMessage m = new MQMessage();
m.writeString("Hello, world!!!");
Q.put(m); //записываю сообщение в очередь
...
```

В WebSphere MQ существует возможность отправлять сообщения без явного открытия очереди непосредственно через объект менеджера очередей. У класса *MQQueueManager* имеется метод *put*, который фактически выполняет три операции: открывает очередь, записывает в очередь сообщение и закрывает очередь. В метод передается ссылка на сообщение *MQMessage* и название очереди, в которую это сообщение нужно записать.

```
public void put(String qName, MQMessage msg)
                              throws MQException;
```

В следующем примере мы запишем тестовое сообщение в очередь *MY.QUEUE*, используя объект менеджера очередей:

```
MQQueueManager QM = new MQQueueManager("QM1");
MQMessage m = new MQMessage();
m.writeString("Hello, world!!!");
// записываю сообщение в очередь
QM.put("MY.QUEUE",m);
...
```

Метод *put* менеджера очередей используют в том случае, когда планируется записать единственное сообщение в очередь. Если требуется записывать в процессе работы программы несколько сообщений в одну и ту же очередь, данный способ записи будет неэффективен, так как каждый раз при его вызове очередь будет снова открываться и закрываться, что потребует дополнительного времени.

3.9. Чтение сообщения из очереди

Для чтения сообщения из очереди используется метод *get* класса *MQQueue*. Метод извлекает сообщение из очереди и записывает его в заданный объект *MQMessage*:

```
public void get(MQMessage message)
                 throws MQException
```

В следующем примере мы прочитаем тестовое сообщение из очереди *MY.QUEUE*:

```
MQQueueManager QM = new MQQueueManager("QM1");
MQQueue q = QM.accessQueue("MY.QUEUE",
                       MQC.MQOO_INPUT_AS_Q_DEF);
MQMessage m = new MQMessage();
// читаю сообщение из очереди
q.get(m);
// обрабатываю сообщение и вывожу на экран
String s = m.readLine();
System.out.print(s);
...
```

В том случае, если очередь пуста, метод сгенерирует исключение *MQException* с кодом причины *MQException.MQRC_NO_MSG_AVAILABLE* (код *2033*):

```
try { q.get(); }
catch(MQException e){
   if (e.reasonCode ==
           MQException.MQRC_NO_MSG_AVAILABLE)
      System.out.println("очередь пустая");
}
```

При чтении сообщения из очереди можно указать дополнительные опции, используя класс *MQGetMessageOptions*:

```
public void get(MQMessage message,
         MQGetMessageOptions getMessageOptions)
                            throws MQException;
```

Данный класс позволяет:

- определить интервал ожидания сообщения в очереди;
- установить использование транзакций при чтении сообщения;
- определить критерии поиска сообщения в очереди;
- установить режим просмотра сообщений (сообщения будут прочитаны без удаления из очереди).

В следующем примере мы установим интервал ожидания поступления сообщения в очередь:

```
...
MQMessage m = new MQMessage();
MQGetMessageOptions gmo =
                    new MQGetMessageOptions();
gmo.options = MQC.MQGMO_WAIT;
gmo.waitInterval = 10000; // 10 секунд
q.get(m,gmo);
...
```

Поле *waitInterval* класса *MQGetMessageOptions* задает интервал ожидания поступления сообщения в очередь и используется вместе с опцией *MQC.MQGMO_WAIT*. В данном случае, метод *get* при наличии сообщения в очереди сразу же вернет его программе. Если очередь окажется пустой, метод в течение 10 секунд будет ждать поступления сообщения, и если сообщение за это время не придет, метод сгенерирует исключение *MQException*.

Для параметра *waitInterval* можно установить специальное значение *MQC.MQWI_UNLIMITED*. В таком случае метод *get* будет ждать поступления сообщения неограниченное время.

3.10. Пример взаимодействия через очереди

Разработаем клиент-серверную систему «Калькулятор».

Клиент передает серверу запросы на выполнение арифметических операций над двумя числами (рис. 14). Сервер выполняет вычисления и возвращает клиенту ответ. Взаимодействие осуществляется в асинхронном режиме через очереди сообщений по схеме «запрос-ответ» (сервер работает с единственным клиентом).

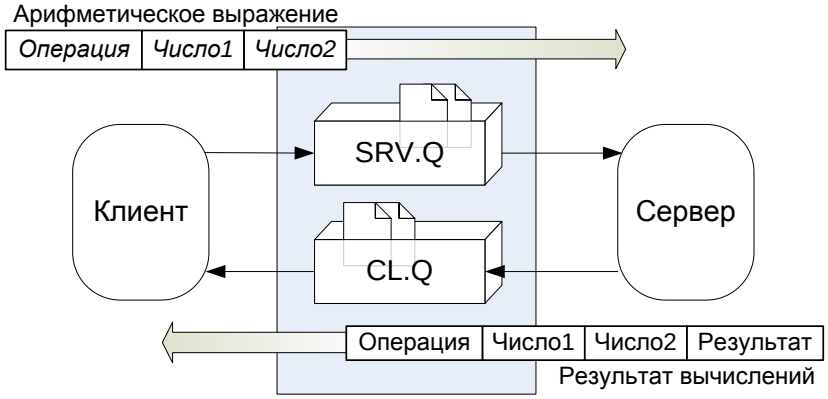


Рис. 14. Клиент-серверная система «Калькулятор»

Формат сообщений, которыми обмениваются клиент и сервер – двоичный. Структура запросов клиента представлена в таблице 1, структура ответов сервера – в таблице 2.

Обратите внимание, что ответное сообщение дублирует параметры, передаваемые в запросе. Это сделано для того, чтобы клиент смог понять, на какой именно запрос он получает ответ.

Таблица 1. Структура запроса клиента

Поле	Длина (байт)	Тип данных	Описание
Операция	4	int	Код арифметической операции: 0 – сложение 1 – вычитание
Число1	4	int	Первый операнд
Число2	4	int	Второй операнд

Таблица 2. Структура ответа сервера

Поле	Длина (байт)	Тип данных	Описание
Операция	4	int	Код выполненной арифметической операции: 0 – сложение 1 – вычитание
Число1	4	int	Первый операнд
Число2	4	int	Второй операнд
Результат	4	int	Результат сложения/вычитания

Взаимодействие клиента и сервера происходит по следующему сценарию:

1. Приложение-клиент запускается в первый раз (метод *Client.main1*) и отправляет серверу несколько запросов на выполнение арифметических вычислений, после чего завершает свою работу.
2. Запускается сервер. Он получает запросы клиента, производит необходимые вычисления, формирует и отправляет ответы, и завершает свою работу.
3. Клиент запускается вновь (метод *Client.main2*) и получает результаты вычислений от сервера.

Исходный код приложения–сервера

```
import com.ibm.mq.*;
public class Server
{
  private MQQueueManager QM = null; // Менеджер
                                    // очередей
  private MQQueue Q1 = null; // Очередь запросов
  private MQQueue Q2 = null; // Очередь ответов

  // Запуск сервера
  public void start(String QMName, String IP,
                int port, String channel,
                String Q1_name, String Q2_name)
                          throws MQException
  {
    // Настриваю среду
    MQEnvironment.hostname = IP;
    MQEnvironment.port = port;
    MQEnvironment.channel = channel;
    // Устанавливаю соединение с менеджером
    QM = new MQQueueManager(QMName);
    // Открываю очередь запросов на чтение
    Q1 = QM.accessQueue(Q1_name,
                  MQC.MQOO_INPUT_EXCLUSIVE);
```

```
    // Открываю очередь ответов на запись
    Q2 = QM.accessQueue(Q2_name,
                  MQC.MQOO_OUTPUT);
    // Обрабатываю все запросы
    int i=0;
    while (processQuery()) i++;
    // Завершаю работу
    Q1.close(); Q2.close(); QM.disconnect();
    System.out.println("Обработано "+i+
                       " запросов");
  }
  // Обработка сообщения-запроса
  public boolean processQuery()
  {
    try
    {
      // Настраиваю интервал ожидания 3 сек.
      MQGetMessageOptions gmo =
                   new MQGetMessageOptions();
      gmo.options = MQC.MQGMO_WAIT;
      gmo.waitInterval = 3000;
      // Читаю сообщение из очереди запросов
      MQMessage query = new MQMessage();
      Q1.get(query,gmo);
      // Обрабатываю сообщение
      int oper  = query.readInt(); // Операция
      int v1    = query.readInt(); // Число1
      int v2    = query.readInt(); // Число2
      // Считаю результат
      int result; // Результат операции
      result = (oper==0)? (v1+v2) : (v1-v2);
      // Формирую ответ
      MQMessage response = new MQMessage();
      response.writeInt(oper);
      response.writeInt(v1);
      response.writeInt(v2);
      response.writeInt(result);
      // Отправляю ответ
      Q2.put(response);
      return true;
    }
    catch (Exception e)
    {
        return false;
    }
  }
  public static void main(String[] args)
                              throws MQException
  {
    (new Server()).start("QM1","localhost",1414,
                         "SYSTEM.DEF.SVRCONN",
                         "SRV.Q","CL.Q");
  }
}
```

Исходный код приложения-клиента

```
import com.ibm.mq.*;
import java.io.IOException;

public class Client
{
  private MQQueueManager QM = null; // Менеджер
  private MQQueue Q1 = null; // Очередь запросов
  private MQQueue Q2 = null; // Очередь ответов

  // конструктор
  public Client(String QMName, String IP,
                int port, String channel,
                String Q1_name,  String Q2_name)
                throws IOException, MQException
  {
    // Настриваю среду
    MQEnvironment.hostname = IP;
    MQEnvironment.port = port;
    MQEnvironment.channel = channel;
    // Устанавливаю соединение с менеджером
    QM = new MQQueueManager(QMName);
    // Открываю очередь запросов на запись
    Q1 = QM.accessQueue(Q1_name,
                    MQC.MQOO_OUTPUT);
    // Открываю очередь ответов на чтение
    Q2 = QM.accessQueue(Q2_name,
                    MQC.MQOO_INPUT_EXCLUSIVE);
  }

  // отправить запрос серверу
  private void sendQuery(int operation,
                int value1, int value2)
                throws IOException, MQException
  {
    // Создаю сообщение-запрос
    MQMessage query = new MQMessage();
    query.writeInt(operation); // Операция
    query.writeInt(value1);    // Число1
    query.writeInt(value2);    // Число2
    // Записываю в очередь запросов
    Q1.put(query);
  }

  // посчитать сумму чисел
  public void sum(int value1, int value2)
                 throws IOException, MQException
  {
    sendQuery(0, value1, value2);
  }
```

```
// посчитать разность чисел
public void sub(int value1, int value2)
                throws IOException, MQException
{
  sendQuery(1, value1, value2);
}

// Получить ответ от сервера
public boolean printResult()
{
  try
  {
    // Читаю сообщение из очереди ответов
    MQMessage response = new MQMessage();
    Q2.get(response);
    int oper = response.readInt(); // Операция
    int v1   = response.readInt(); // Число1
    int v2   = response.readInt(); // Число2
    int res = response.readInt(); // Результат
    String s = (oper==0)? "+" : "-";
    System.out.println(v1 + s + v2 +
                       " = " + res);
    return true;
  }
  catch(Exception e)
  {
    return false;
  }
}
// отсоединиться
public void disconnect()
              throws IOException, MQException
{
  Q1.close(); Q2.close(); QM.disconnect();
}

// ПЕРВЫЙ ЗАПУСК КЛИЕНТА
public static void main1()
{
  try
  {
    Client client =
          new Client("QM1","localhost",1414,
                     "SYSTEM.DEF.SVRCONN",
                     "SRV.Q","CL.Q");
    client.sum(15,20);
    client.sub(30,38);
    client.sum(100,200);
    client.disconnect();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
```

```
  // ВТОРОЙ ЗАПУСК КЛИЕНТА
  public static void main2()
           throws IOException, MQException
  {
    Client client =
           new Client("QM1","localhost",1414,
                      "SYSTEM.DEF.SVRCONN",
                      "SRV.Q","CL.Q");
    while (client.printResult());
  }

  public static void main(String[] args)
            throws IOException, MQException
  {
        main1(); // или main2()
  }
}
```