

Desenvolvendo Aplicações Java Utilizando WebSphere MQ

Otavio Rodolfo Piske

angusyoung@gmail.com

1. Introdução.....	3
2. Preparação do Ambiente	3
Requerimentos.....	3
Download e Instalação do WebSphere MQ.....	4
<i>Instalação de um Servidor WebSphere MQ.....</i>	<i>4</i>
<i>Instalação do Cliente WebSphere MQ</i>	<i>5</i>
3. Introdução ao WebSphere MQ	6
Preparando um Servidor MQ.....	6
<i>Criação das Filas.....</i>	<i>6</i>
4. Desenvolvimento	7
Planejando a Solução.....	7
Codificação.....	11
<i>Conecta no servidor.....</i>	<i>11</i>
<i>Instância a interface com o MQ</i>	<i>11</i>
<i>Desconecta do Servidor</i>	<i>12</i>
<i>Abre uma fila para escrita.....</i>	<i>12</i>
<i>Envia dados pela interface.....</i>	<i>13</i>
<i>Abre uma fila para leitura.....</i>	<i>14</i>
<i>Recebe Dados Pela Interface</i>	<i>15</i>
5. Testes.....	16
Compilação	16
Execução	17
6. Referência	18

1. Introdução

O WebSphere MQ, também conhecido como MQSeries, é um *middleware* orientado a mensagens desenvolvido pela IBM desde 1992. Ao longo dos anos tem sido utilizado, com sucesso, na implementação de sistemas distribuídos e como uma alternativa aos mecanismos de comunicação inter-processos nativos dos sistemas operacionais.

O objetivo deste artigo é fazer uma demonstrar à utilização do Java como plataforma de desenvolvimento de aplicações WebSphere MQ, bem como uma introdução do produto em si. Ao final deste artigo espera-se que o leitor seja capaz de desenvolver aplicações simples utilizando WebSphere MQ e tenha o conhecimento básico para alavancar o seu conhecimento, através da ampla documentação disponibilizada pela IBM.

2. Preparação do Ambiente

Requerimentos

Para o acompanhamento das atividades propostas nesse artigo recomenda-se o seguinte:

- 1 estação de trabalho rodando Windows, Linux ou Solaris.
 - Para elaboração deste artigo foram utilizados duas estações de trabalho rodando Fedora Linux 14, desta forma os detalhes de configuração do WebSphere MQ em ambiente Windows, distribuições Linux que não sejam da família do Red Hat Linux e Solaris Unix não constam neste artigo. Os detalhes de configuração do MQ nestes ambientes podem ser encontrados nos documentos de referência disponibilizados no final do artigo.
- Ambiente de desenvolvimento Java 1.5 ou superior.
- Eclipse Helios.

- WebSphere MQ 6.0.2.4.

Download e Instalação do WebSphere MQ

Uma versão de demonstração do WebSphere MQ pode ser baixado no site do produto, na página da IBM [Referência](#). Para fazer o baixar o arquivo será necessário criar um passaporte, sem qualquer custo, no site da IBM.

Nota: os procedimentos descritos nas próximas nas próximas duas seções foram executados em dois computadores diferentes. É perfeitamente seguro unificá-los caso a instalação do cliente e do servidor ocorra na mesma máquina.

Instalação de um Servidor WebSphere MQ

No Linux o WebSphere MQ Server é disponibilizado em formato RPM, deste modo a instalação é bastante simples.



```
root@mqserver-virt: /home/otavio/software/mq/6.0.0.0 — ssh ...
[root@mqserver-virt 6.0.0.0]# ls
copyright                               MQSeriesMsg_fr-6.0.0-0.i386.rpm
gsk7bas-7.0-3.15.i386.rpm                 MQSeriesMsg_it-6.0.0-0.i386.rpm
IBMJava2-SDK-1.4.2-0.0.i386.rpm           MQSeriesMsg_ja-6.0.0-0.i386.rpm
bin/                                       MQSeriesMsg_ko-6.0.0-0.i386.rpm
license/                                   MQSeriesMsg_pt-6.0.0-0.i386.rpm
mqlicense.sh*                             MQSeriesMsg_zh_CN-6.0.0-0.i386.rpm
MQSeriesClient-6.0.0-0.i386.rpm           MQSeriesMsg_zh_TW-6.0.0-0.i386.rpm
MQSeriesConfig-6.0.0-0.i386.rpm          MQSeriesRuntime-6.0.0-0.i386.rpm
MQSeriesFTA-6.0.0-0.i386.rpm             MQSeriesSamples-6.0.0-0.i386.rpm
MQSeriesIES30-6.0.0-0.i386.rpm           MQSeriesSDK-6.0.0-0.i386.rpm
MQSeriesJava-6.0.0-0.i386.rpm            MQSeriesServer-6.0.0-0.i386.rpm
MQSeriesKeyMan-6.0.0-0.i386.rpm          MQSeriesTCCClient-6.0.0-0.i386.rpm
MQSeriesMan-6.0.0-0.i386.rpm             README/
MQSeriesMsg_de-6.0.0-0.i386.rpm          readadd.txt
MQSeriesMsg_es-6.0.0-0.i386.rpm          README/
[root@mqserver-virt 6.0.0.0]#
```

Antes da instalação dos RPMs será necessário aceitar a licença. Para fazê-lo basta executar o script `mqlicense.sh`. Após a execução desses passos será possível prosseguir com a instalação do MQ.

Para iniciar a instalação execute o seguinte comando:

```
rpm -ivh MQSeriesMan-6.0.0-0.i386.rpm MQSeriesServer-6.0.0-0.i386.rpm MQSeriesRuntime-6.0.0-0.i386.rpm
```

Ao terminar a instalação, execute o `update` para a versão 6.0.2.4:

```
rpm -Uvh MQSeriesMan-U814336-6.0.2-4.i386.rpm
MQSeriesServer-U814336-6.0.2-4.i386.rpm MQSeriesRuntime-
U814336-6.0.2-4.i386.rpm
```

Por último modifique seu usuário, adicionando-o ao grupo *mqm*. Isto dará permissões administrativas para o usuário e o habilitará a administrar o servidor MQ recém-instalado.

```
usermod -a -G mqm <username>
```

Dica: recomenda-se adicionar o diretório bin da instalação do MQ – localizada em */opt/mqm/bin* - no *PATH* do seu usuário, deste modo a execução dos comandos fica mais fácil.

Execute um *logout* do seu usuário para efetivar as mudanças em seu perfil e valide a instalação através do comando *dspmqrver*.

A terminal window titled 'root@mqserver-virt:/home/otavio/software/mq/6.0.2.4 — ssh ...' shows the execution of the 'dspmqrver' command. The output displays the following information: Name: WebSphere MQ, Version: 6.0.2.4, CMVC level: p600-204-080509, and BuildType: IKAP - (Production). The prompt returns to the user after the command is executed.

```
[otavio@mqserver-virt] ~ 13:22 $ dspmqrver
Name:      WebSphere MQ
Version:   6.0.2.4
CMVC level: p600-204-080509
BuildType: IKAP - (Production)

[otavio@mqserver-virt] ~ 13:22 $
```

Instalação do Cliente WebSphere MQ

Assim como o servidor, o cliente MQ para Linux também é disponibilizado em formato RPM. Como pré-requisito para instalação do cliente será necessário aceitar a licença através da execução do comando *mqlicense.sh*, que é distribuído junto com o pacote do MQ.

Para iniciar a instalação do cliente MQ execute:

```
rpm -ivh MQSeriesClient-6.0.0-0.i386.rpm MQSeriesJava-
6.0.0-0.i386.rpm MQSeriesRuntime-6.0.0-0.i386.rpm
```

Ao terminar a instalação, atualize para a versão 6.0.2.4:

```
rpm -Uvh MQSeriesClient-U814336-6.0.2-4.i386.rpm  
MQSeriesJava-U814336-6.0.2-4.i386.rpm MQSeriesRuntime-  
U814336-6.0.2-4.i386.rpm
```

3. Introdução ao WebSphere MQ

O WebSphere MQ, conforme explicado anteriormente, é um *middleware* orientado a mensagens. Por ser orientado a mensagens, isso significa que a troca de dados entre os pares em uma comunicação MQ se dá de forma unitária. Ou seja, não existe um fluxo constante e ilimitado de dados.

Não se deixe enganar pela simplicidade do conceito – troca de mensagens entre aplicações. O WebSphere MQ se baseia em uma série de conceitos que devem ser compreendidos para que seja possível utilizá-lo adequadamente. Dentre esses conceitos, podemos explicar os principais, resumidamente, da seguinte forma:

- Gerente de Filas (*queue manager*): gerência um conjunto de filas.
- Canal (*channel*): um agrupamento de filas dentro de um mesmo *queue manager*.
- Fila (*queue*): é um *container* de mensagens. Serve como meio de transmissão utilizado pelo MQ para que as mensagens inseridas em uma ponta sejam recebidas em outra.

Preparando um Servidor MQ

Para a execução dos procedimentos descritos nesse artigo será necessário a configuração de um servidor MQ contendo um *queue manager*, um canal e uma fila. Esse é um processo que, facilmente, pode ficar bastante complicado.

Criação das Filas

Para a criação das filas crie um arquivo chamado *create-mq.mqsc* contendo o seguinte:

```
define qlocal (TUTO001.REQUEST)
define listener (TUTO001.LIS) trptype(TCP) ipaddr('0.0.0.0') port(1414)
start listener
define channel(TUTO001.CHANNEL) chltype(SVRCONN) trptype(TCP)
start channel (TUTO001.CHANNEL)
```

Apos salvar o arquivo rode os seguintes comandos para criar o *queue manager*, iniciá-lo e configurar as filas:

```
crtmqmq -q TUTO001
```

```
strmqm TUTO001
```

```
runmqsc < create-mq.mqsc
```

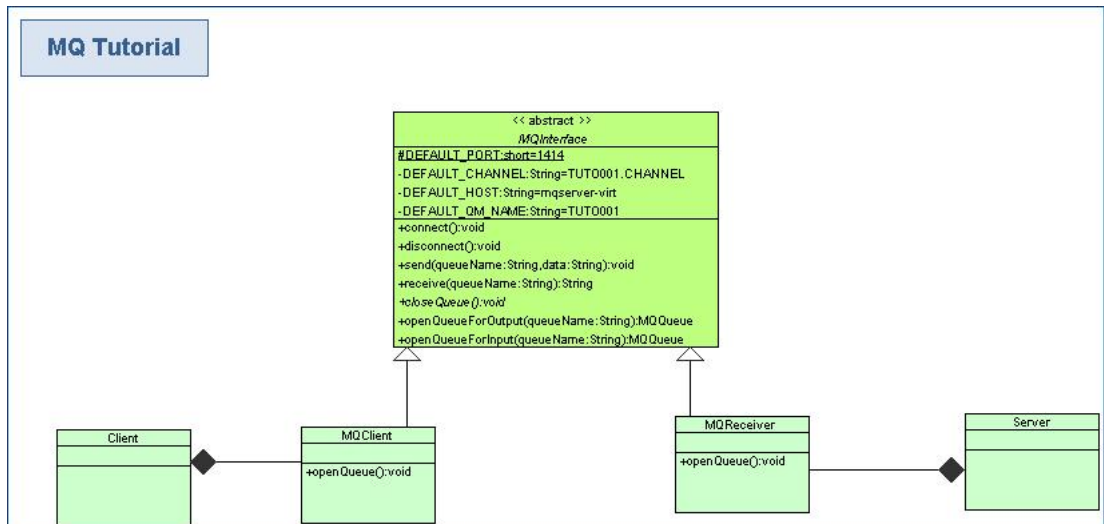
Uma vez que esses passos tenham sido executados com sucesso, um gerente de filas estará ativo e ouvindo na porta TCP 1414, a porta padrão, utilizada pelo WebSphere MQ.

4. Desenvolvimento

Planejando a Solução

Devido à ampla gama de opções, conceitos e funcionalidades, desenvolver para MQ pode ser uma tarefa complicada para desenvolvedores inexperientes. Desta forma trabalharemos em um design simples procurando abstrair a complexidade do MQ.

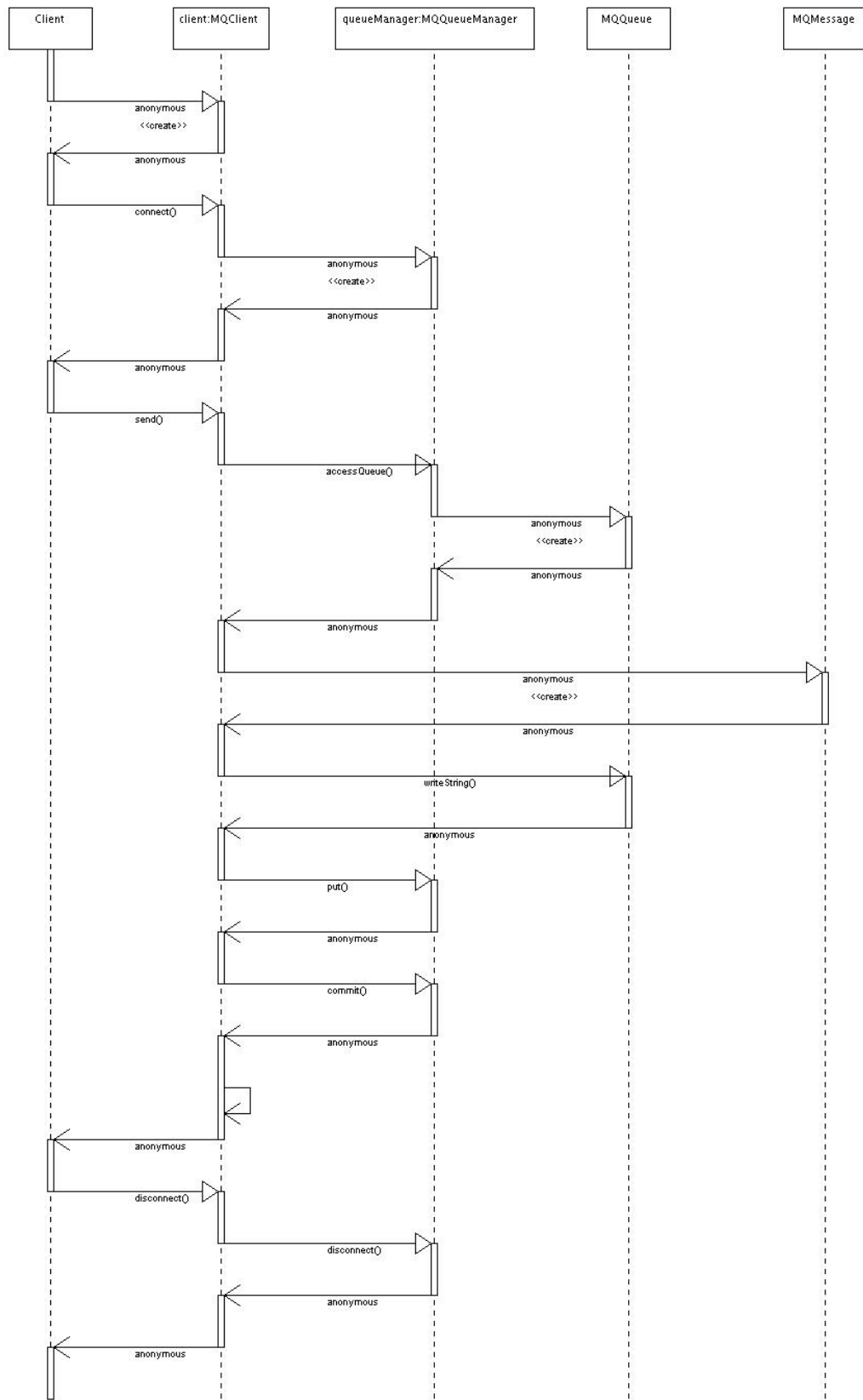
Nosso design será composto de classes “cliente” que irão se conectar ao servidor MQ, configurar a conexão, enviar e receber dados. O diagrama de classes abaixo mostra o relacionamento entre elas:



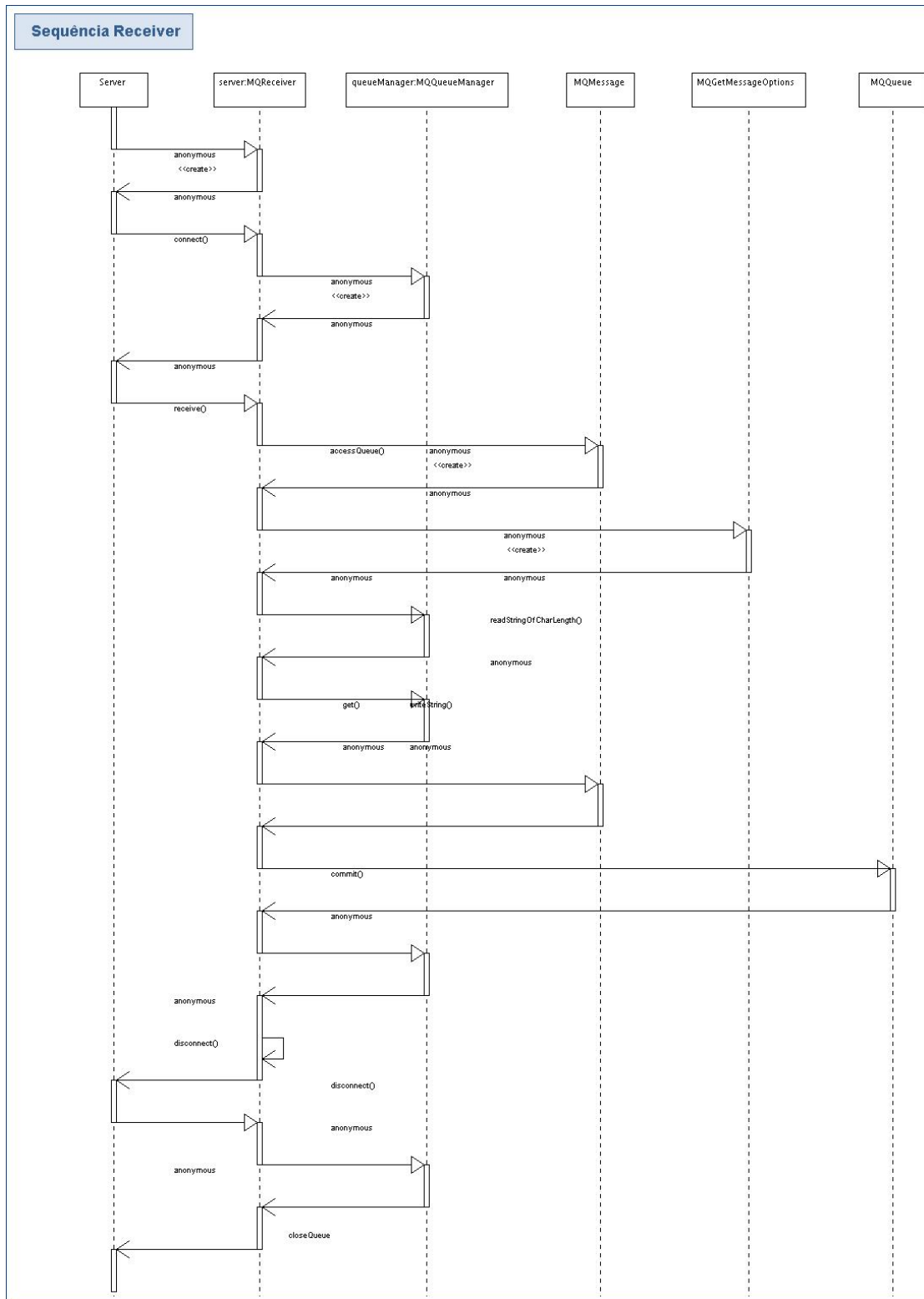
- *MQInterface*: classe base. Utilizada para gerenciar conectar no servidor, configurar a conexão e enviar e receber dados.
- *MQClient*: utilizada para abstrair a utilização MQ como um cliente. Não será amplamente utilizada nesse tutorial.
- *MQReceiver*: utilizada para abstrair a utilização do MQ como um receptor de mensagens (servidor).

O diagrama de sequência abaixo detalha, simplificado, o envio de uma mensagem via MQ.

Sequência Cliente



Aqui temos um segundo diagrama de sequência detalhando os passos utilizados pelo servidor para receber uma mensagem.



Codificação

Conecta no servidor

```
protected MQInterface(String hostname, int port, String channel) {
    // Seta o hostname do servidor MQ
    MQEnvironment.hostname = hostname;

    // Seta a porta do servidor MQ
    MQEnvironment.port = port;

    // Seta o canal padrao
    MQEnvironment.channel = channel;
}
```

Em nosso exemplo utilizamos o construtor da interface MQ para configurar alguns parâmetros básicos de conexão.

Instância a interface com o MQ

```
public void connect() throws MQException {
    disconnect();

    /*
     * Cria uma nova instância do objeto de acesso ao
     * queue manager, passando o nome da fila como
     * parametro
     */
    queueManager = new MQQueueManager(DEFAULT_QM_NAME);
}
```

O método *connect* cria uma instância de um objeto do tipo *MQQueueManager*. Esse objeto é utilizado para abstrair a manipulação e o gerenciamento dos objetos diretamente relacionados ao *queue manager*. Por exemplo, acesso às filas, *commit* e/ou *rollback* de transações, etc.

É importante notar que não é preciso chamar um método de conexão: quando o objeto *MQQueueManager* é instanciado ele se conecta no *queue manager*, utilizando os parâmetros previamente configurados pela *MQEnvironment*.

Desconecta do Servidor

```
public void disconnect() throws MQException {
    /* Caso exista uma conexao aberta anteriormente,
     * fecha-a para evitar desperdicios de recursos
     * no servidor MQ.
     */
    if (queueManager != null) {
        if (queueManager.isConnected()) {
            queueManager.disconnect();
        }
    }
}
```

A desconexão do servidor MQ é quase tão simples quanto à conexão ao mesmo: uma vez que o objeto é validado, verifica-se o estado da conexão utilizando o método *isConnected* e, estando conectado, a conexão é finalizada através do método *disconnect*.

Dica: evite utilizar a variável membro *isConnected* pois ela está marcada como depreciada (*deprecated*).

Abre uma fila para escrita

```
protected MQQueue openQueueForOutput(String queueName) throws MQException {
    /*
     * Define as opções de acesso a fila:
     * - MQOO_OUTPUT = abre a fila p/ inclusão de mensagens (saida, host -> mq)
     * - MQOO_FAIL_IF QUIESCING = falha se o servidor estiver sendo desligado
     */
    int openOptions = MQC.MQOO_OUTPUT | MQC.MQOO_FAIL_IF QUIESCING;

    return queueManager.accessQueue(queueName, openOptions);
}
```

O primeiro passo antes de enviar dados por uma fila MQ é requisitar ao *queue manager* acesso a fila. Isso é feito através do método *accessQueue*, do objeto *MQQueueManager*. Esse método recebe o nome da fila seguido de um inteiro, que serve como mapa de bits das opções de abertura da fila.

Envia dados pela interface

```
public void send(String queueName, String data) throws MQException,
IOException
{
    MQQueue queue = openQueueForOutput(queueName);
    MQMessage message = new MQMessage();

    /*
     * Seta o formato da mensagem. No nosso exemplo
     * estamos mandando uma mensagem de texto no MQ,
     * desta forma o formato é String.
     */
    message.format = MQC.MQFMT_STRING;

    /*
     * Tempo de vida da mensagem. Caso expirado, o
     * queue manager poderá descartar a mensagem se
     * necessário
     */
    message.expiry = 10000;

    // Serializa o objeto
    message.writeString(data);

    // Coloca a mensagem na fila
    queue.put(message);

    // Efetiva a inclusao da mensagem na fila
    queueManager.commit();

    closeQueue(queue);
}
```

Enviar dados pela interface MQ consiste, basicamente, 3 grupos de tarefas. Elas envolvem preparar os recursos, enviar a mensagem e tratar os erros e liberar os recursos utilizados. A partir do código utilizado como exemplo podemos categorizar as linhas de código acima da seguinte forma:

1. Preparação dos recursos utilizados na transação: mensagem e fila, neste exemplo:
 - a. Acesso a fila para escrita de dados, conforme detalhado no passo anterior.
 - b. Preparação dos parâmetros de envio da mensagem: formato e tempo de vida. É importante ressaltar que a opção *format*, da mensagem diz respeito ao formato do dado sendo enviado e não ao formato da transação. Desta forma, não importa se a transação é em texto puro, XML ou algum formato próprio baseado em caracteres: o formato do dado é *MQFMT_STRING*. Em contrapartida, existem outras opções igualmente adequadas para outros tipos de protocolos.

- c. Serialização do objeto a ser enviado: sempre feita através do método adequado ao tipo de dado enviado na transação, conforme especificado pelo formato no item anterior.
2. Envio da transação e tratamento de erros: através do método *put*, do objeto *MQQueue* que é recebido ao acessar a fila no passo 1a.
 3. Efetivação da transação e liberação de recursos.
 - a. Efetivação da transação: feita através de uma chamada ao método *commit*, existente no *MQQueueManager*. É possível configurar uma transação para que a efetivação da mesma seja automática, sem a necessidade do *commit*. Isso é feito através das opções de *sync-point* durante o acesso a fila. A configuração dessa política de transação não será abordada nesse artigo.
 - b. Liberação de recursos: feita através do método *close* que fecha o acesso a fila, liberando eventuais recursos alocados para a transação.

Note que o tratamento de erros do WebSphere MQ foi deixado de intencionalmente. O tratamento de erros do WebSphere MQ é um razoavelmente complicado, desta forma esse assunto será abordado posteriormente em um outro artigo.

Abre uma fila para leitura

```
protected MQQueue openQueueForInput(String queueName) throws MQException {
    /* Define as opções de acesso a fila:
     * - MQOO_INPUT_AS_Q_DEF = abre a fila p/ obtenção de mensagens
     *   (entrada, mq -> host) utilizando as definições padrão da fila
     * - MQOO_FAIL_IF QUIESCIENG = falha se o servidor estiver sendo
     *   desligado
     */
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
MQC.MQOO_FAIL_IF QUIESCIENG;
    return queueManager.accessQueue(queueName, openOptions);
}
```

Assim como no envio de dados através do MQ, é necessário abrir a fila com os parâmetros apropriados ao recebimento de mensagens. Assim como na abertura de fila para envio, é através de uma chamada ao método *accessQueue*, que nos recebemos um objeto *MQQueue* apropriado para gravação de mensagens.

Recebe Dados Pela Interface

```
public String receive(String queueName) throws MQException, IOException {
    MQMessage message = new MQMessage();
    MQQueue queue = null;
    String ret = null;

    /* Instância o objeto utilizado para configurar os parâmetros
     * utilizados para obter a mensagem da fila
     */
    MQGetMessageOptions gmo = new MQGetMessageOptions();

    /* O MQ fornece diferentes formas de relacionar uma mensagem
     * previamente enviada com a sua resposta.
     */
    gmo.matchOptions = MQC.MQMO_NONE;

    // Habilita a espera no recebimento de mensagens
    gmo.options += MQC.MQGMO_WAIT;

    // Habilita a espera ilimitada
    gmo.waitInterval = MQC.MQEI_UNLIMITED;

    /*
     * Abre a queue para leitura
     */
    queue = openQueueForInput(queueName);

    /*
     * Obtém a mensagem da fila, utilizando as opções definidas no
     * objeto gmo
     */
    queue.get(message, gmo);

    try {
        ret = message.readStringOfCharLength(
            message.getMessageLength());
    }
    catch (EOFException e) {
        System.out.println(
            "A leitura da mensagem terminou abruptamente");
    }
    finally {
        closeQueue(queue);
        queueManager.commit();
    }

    return ret;
}
```

A obtenção das mensagens através do MQ também pode ser dividida em 3 grupos de tarefas: preparação dos recursos utilizados na transação, recebimento e tratamento de erros e efetivação transação.

Usando o código mostrado acima, é possível categorizar as instruções da seguinte forma:

1. Preparação dos recursos utilizados na transação:
 - a. Criação dos objetos utilizados na transação.
 - b. Configuração das regras de match, utilizadas para casar os identificadores de uma mensagem.
 - c. Abertura da fila, conforme descrito no passo anterior.
2. Recebimento da mensagem, pelo objeto *MQQueue* retornado como parte do passo 1c, através de uma chamada ao método *get*.
 - a. De-serialização da mensagem recebida, transformando-a em um objeto manipulável pela nossa aplicação. Neste exemplo isto é feito através de uma chamada ao método *readStringOfCharLength*. Este método recebe como entrada o tamanho de caracteres a serem de-serializados e, a partir desta informação, retorna um objeto do tipo *String*.
 - b. Tratamento de erros, tentando gerenciar as exceções lançadas.
3. Efetivação da transação, pelo fechamento da fila e *commit* da transação.

5. Testes

Verifique a seção Referencia para fazer download do código-fonte utilizado como exemplo neste artigo.

Compilação

O código-fonte pode ser compilado utilizando o *maven* – versão 3.0 ou superior. Para isto basta acessar o diretório do projeto e executar os seguintes passos:

Compilar a biblioteca comum:

```
cd mqbase && mvn install && cd ..
```

Compilar o cliente:

```
cd mqclient && mvn package
```


Compilar o servidor:

```
cd mqserver && mvn package
```

O arquivo gerado ao final da compilação ficará no diretório *target*.

Execução

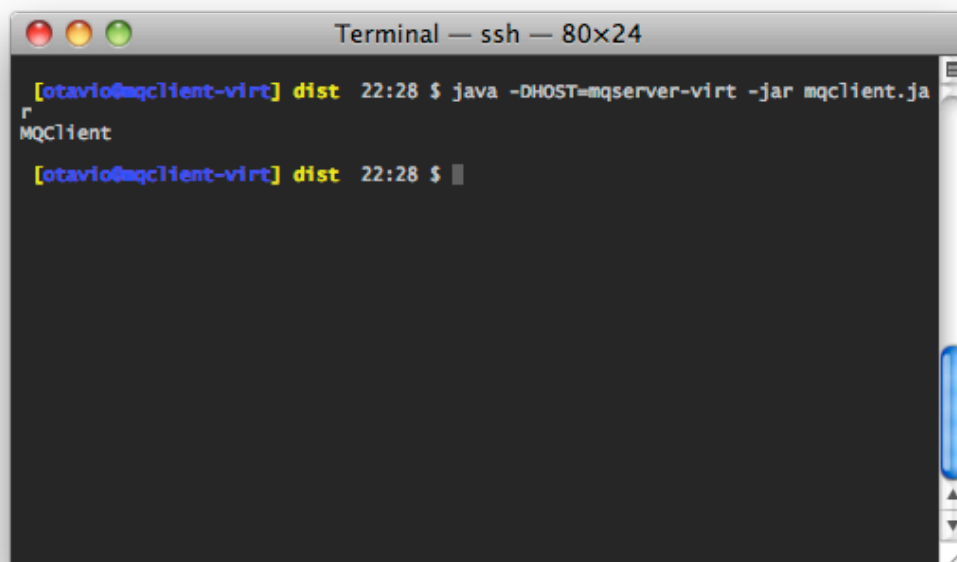
Para rodar as aplicações, inicie primeiramente o servidor e, obtendo sucesso, execute o cliente. Os comandos a serem executados são:

- Servidor: `java -Djava.ext.dirs=/opt/mqm/java/lib -DHOST=<hostname> -jar mqserver-1.0.0.jar`
- Cliente: `java -Djava.ext.dirs=/opt/mqm/java/lib -DHOST=<hostname> -jar mqclient-1.0.0.jar`

Lembre-se de substituir *<hostname>* pelo nome do host ou endereço IP utilizado pelo seu servidor MQ. Uma vez que o cliente tenha sido executado com sucesso, o servidor deverá mostrar a seguinte mensagem: *Hello World*.

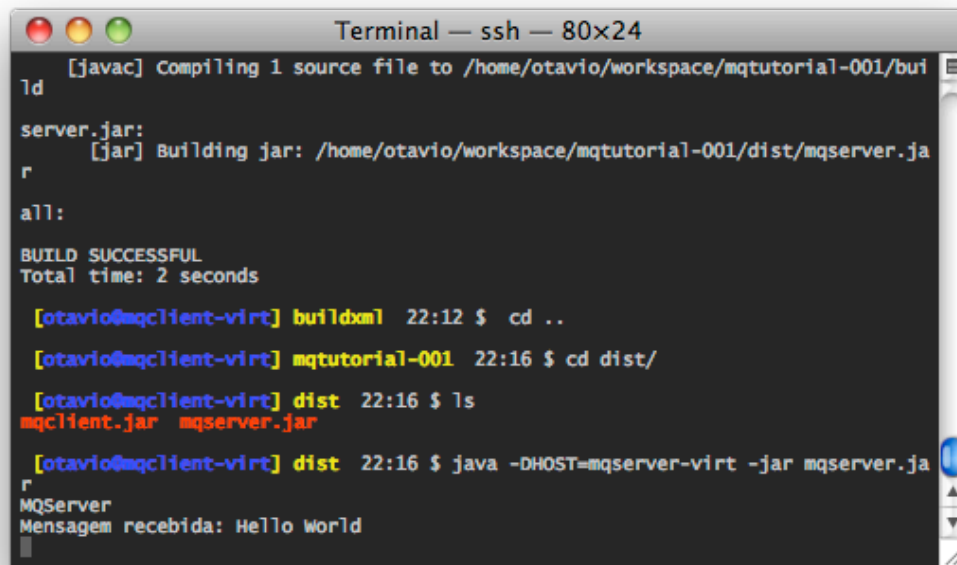
As imagens abaixo mostram o envio e o recebimento das mensagens através dos exemplos.

Cliente:



```
Terminal — ssh — 80x24
[otavio@mqclient-virt] dist 22:28 $ java -DHOST=mqserver-virt -jar mqclient.jar
MQClient
[otavio@mqclient-virt] dist 22:28 $
```

Servidor:



```
Terminal — ssh — 80x24
[javac] Compiling 1 source file to /home/otavio/workspace/mqtutorial-001/build
server.jar:
[jar] Building jar: /home/otavio/workspace/mqtutorial-001/dist/mqserver.jar
all:
BUILD SUCCESSFUL
Total time: 2 seconds

[otavio@mqclient-virt] build$ cd ..
[otavio@mqclient-virt] mqtutorial-001$ cd dist/
[otavio@mqclient-virt] dist$ ls
mqclient.jar  mqserver.jar
[otavio@mqclient-virt] dist$ java -DHOST=mqserver-virt -jar mqserver.jar
MQServer
Mensagem recebida: Hello World
```

6. Referência

1. Página do Produto WebSphere MQ: <http://www-01.ibm.com/software/integration/wmq/>
2. Download do código-fonte utilizado no artigo: <http://www.angusyoung.org/arquivos/artigos/websphere/mq/java/01/mqtutorial-001.tar.bz2>
3. Scripts para configuração do servidor MQ: <http://www.angusyoung.org/arquivos/artigos/websphere/mq/java/01/mq-scripts.zip>