软件复用课程设计

Client-Server 应用程序

复用文档

MessageUtils构件

小组： Team10

成员： 谢志杰 1352975

计鹏玥 1352914

王思尧 1352896

谢明玥 1352937

徐锦程 1353012

[1． 构件简介 1](#_Toc5431)

[2． 接口说明 1](#_Toc2229)

[2.1 Message.java 1](#_Toc5671)

[(1) Message(String msg, long ownerThread) 1](#_Toc14078)

[(2) getOwner() 2](#_Toc13858)

[(3) getValue(String key) 2](#_Toc7214)

[(4) setValue(String key, String value) 3](#_Toc26190)

[(5) setValue(String key, long value) 3](#_Toc24343)

[(6) toString() 4](#_Toc19313)

[(7) reset(String msg) 4](#_Toc11701)

[(8) clear() 5](#_Toc20126)

[(9) init(String queue\_name, String hostName) 5](#_Toc7093)

[(10) bindTo(String exchangeName, String routerKey) 6](#_Toc20525)

[(11) getQueueName() 6](#_Toc7677)

[(12) getChannel() 7](#_Toc30611)

[(13) publishToOne(String exchangeName, String routerKey) 7](#_Toc21428)

[(14) publishToAll(String exchangeName) 8](#_Toc11986)

[(15) publishToOthers(String exchangeName, String routerKey) 8](#_Toc4196)

[(16) terminate() 9](#_Toc3739)

[2.2 MessageDeparturer.java 9](#_Toc30747)

[(1) MessageConsumer(Message msg, PrintWriter out, Logger logger, String logKey) 9](#_Toc26717)

[(2) sendMessage(String msg) 10](#_Toc17857)

[(3) handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) 10](#_Toc17867)

[(4) MessageDeparturer(Message msg, PrintWriter out, Logger logger, String logKey) 11](#_Toc21757)

[3． Demo 12](#_Toc12710)

# 构件简介

|  |  |  |
| --- | --- | --- |
|  | **消息 － Message** | **消息发送 － MessageDeparturer** |
| 构件介绍 | 这一构件的主要功能是构件与读取客户端与服务端通信时传递的消息，本质上是对于JSON生成与解析的方法的封装，并添加了线程标识符，用以表示消息的所属关系。 | 这一构件的主要功能是对消息进行消耗，即发送消息。 |
| 功能说明 | 用户可以通过该构件实现对消息的构造、消息所属线程的指定、消息内容的添加以及消息内容的解析等操作。 | 用户可以使用该构件来设置消息的接收方和想要进行发送的消息。 |

# 接口说明

## 2.1 Message.java

注意：应先导入MessageUtils包。

### (1) Message(String msg, long ownerThread)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 消息的内容 |
| ownerThread | long | 消息所属线程的标识符 |

**作用描述：**

该方法实质为方法类的构造函数，通过该方法可以实例化Message对象，并且可以指定消息所属的线程以及消息的内容。

**使用方法：**

利用该方法实例化Message方法类，其中需要传递两个参数，第一个参数为string类型，为JSON格式的字符串，即消息的内容，第二个参数为long类型，为消息所属线程的标识符，形如：

Message msg = new Message("{}", this.getId());

**注意事项：**

该方法中指定的消息的内容可以为空的JSON，即形如”{}”的字符串，但该参数不可为空。

### (2) getOwner()

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 含义 |
| ownerThread | long | 消息所属线程的标识符 |
| 返回值 | 类型 | 含义 |

**作用描述：**

该方法用于获取消息所属消息的标识符，通过调用一个实例化的Message对象的该方法，即可获得这个实例化的Message对象的标识符。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法无需传递任何参数，返回类型为long型，即线程标识符，形如：

Long ownerThread = msg.getOwner();

**注意事项：**

该方法返回的是其所属消息的线程的标识符，该标识符在上述构造函数中指定。

### (3) getValue(String key)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| key | String | 键值 |

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 含义 |
| value | String | 键值对应的值 |

**作用描述：**

该方法用于获取消息内容，通过传入key值，可以获取到对应的value值。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法唯一的参数是string类型的key值，返回类型为string类型的value值，形如：

String sValue = msg.getValue(“target”);

**注意事项：**

尽管上述setValue方法实现了重载，value类型可以接受不同类型的参数，包括string类型与long类型，但是本方法返回的value类型仅为string类型。

### (4) setValue(String key, String value)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| key | String | 键值 |
| value | String | 键值对应的值 |

**作用描述：**

该方法用于添加消息内容，通过传递两个参数作为key和value值，可以将这个键值对加入到消息内容中，支持的value值为string类型。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个参数，第一个参数为string类型的key，第二个参数为string类型，调用该方法后，传入的键值对会添加到实例化的消息的内容中，形如：

msg.setValue("event", “login");

### (5) setValue(String key, long value)

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 含义 |
| key | String | 键值 |
| value | long | 键值对应的值 |

**作用描述：**

该方法用于添加消息内容，通过传递两个参数作为key和value值，可以将这个键值对加入到消息内容中，支持的value值为long类型。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个参数，第一个参数为string类型的key，第二个参数为long类型的value，调用该方法后，传入的键值对会添加到实例化的消息的内容中，形如：

msg.setValue("event", “login");

### (6) toString()

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 含义 |
|  | String | JSON对象字符串内容 |

**作用描述：**

该方法用于将JSONObject对象转换为JSON格式的字符串。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法没有输入参数，返回类型为string类型，即JSON格式的字符串，形如：

String jsonMessage = msg.toString();

**注意事项：**

该方法会将实例对象中的内容转换为JSON格式的字符串，转换后若继续通过setValue方法添加内容，则需要在此调用本方法才能得到添加后的JSON格式的字符串。

### (7) reset(String msg)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | JSON对象的内容 |

**作用描述：**

实例化一个JSON对象，可以传入字符串类型的参数。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有一个string类型的输入参数，没有返回值，形如：

String msg = msg.rest(“Hello World!”);

**注意事项：**

传入参数必须是string类型。

### (8) clear()

**作用描述：**

实例化一个空的JSON字符串。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法没有输入参数和返回值，形如：

msg.clear()

**注意事项：**

必须要有一个实例化的Message方法类来调用clear()方法。

### (9) init(String queue\_name, String hostName)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| queue\_name | String | 消息队列的名称 |
| hostName | String | 建立连接的主机名 |

**作用描述：**

使用了消息中间件RabitMQ，来实例化一个factory，并设置hostName，利用factory创建connection，并为connection创建信道。这种设置可利用与但发送多接收的场景，保证消息发送的可靠性，不丢失消息，将消息持久化。还设置了当前的队列名，用于存放消息。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为消息队列的名称，第二个为建立连接的主机名，没有返回值，形如：

msg.init(“msgQueue”, “host1”)

**注意事项：**

A. 两个参数都是string类型。

B. RabbitMQ建议客户端线程之间不要共用Channel，至少要保证共用Channel的线程发送消息必须是串行的，但是建议尽量共用Connection。

### (10) bindTo(String exchangeName, String routerKey)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| exchangeName | String | 交换机的名称 |
| routerKey | String | 路由键的名称 |

**作用描述：**

现有的信道将转发器设置为direct类型， 将消息队列与一个特定的routerKey绑定，并要求二者完全匹配才可进行转发。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为交换机的名称，第二个是路由键的名称，没有返回值，形如：

msg.bindTo(“exchangeName”, “myRouter”)

**注意事项：**

两个输入参数均为string类型。

### (11) getQueueName()

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 含义 |
| queueName | String | 消息队列的名称 |

**作用描述：**

获取消息队列的名称。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法没有传入参数，返回值为当前Messge对象的消息队列的名称，形如：

String queueName = msg.getQueueName();

**注意事项：**

接收返回值的必须为string类型。

### (12) getChannel()

**作用描述：**

获取信道对象。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法没有传入参数，返回值为当前Meaage对象的消息队列的名称，形如：

Channel channel = msg.getChannel();

**注意事项：**

接收返回值的必须为channel类型。

### (13) publishToOne(String exchangeName, String routerKey)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| exchangeName | String | 交换机名称 |
| routerKey | String | 路由键 |

**作用描述：**

将信道中的消息只发送给匹配routerKey的特定接收端。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为交换机名称，第二个为路由键，形如：

msg.publishToOne(“myExchange”, “myRouter”);

**注意事项：**

A. 两个传入参数均为string类型。

B. 该方法的调用只在server端进行。

### (14) publishToAll(String exchangeName)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| exchangeName | String | 交换机名称 |

**作用描述：**

将信道中的消息发送给交换机对应的所有接收方。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有一个string类型的输入参数，代表交换机名称，形如：

msg.publishToAll(“myExchange”);

**注意事项：**

A. 传入参数为string类型。

B. 该方法的调用只在server端进行。

### (15) publishToOthers(String exchangeName, String routerKey)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| exchangeName | String | 交换机名称 |
| routerKey | String | 路由键 |

**作用描述：**

将信道中的消息发送给所有不匹配routerKey的接收端。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为交换机名称，第二个为路由键，形如：

msg.publishToOthers(“myExchange”, “myRouter”);

**注意事项：**

A. 两个传入参数均为string类型。

B. 该方法的调用只在server端进行。

### (16) terminate()

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 含义 |
| true/false | boolean | 判断获取的总消息数是否超过了max\_num\_message |

**作用描述：**

关闭信道和连接。

**使用方法：**

对于一个实例化的Message对象，可以直接调用该方法，该方法既没有输入参数，也没有返回值，形如：

msg.terminate();

**注意事项：**

该方法只可以在server端进行调用，且在调用之后，所有与信道或连接相关的操作都无法继续进行。

## 2.2 MessageDeparturer.java

注意：应先导入MessageUtils包。

### (1) MessageConsumer(Message msg, PrintWriter out, IntervalLogger pm, String logKey)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 消息的内容 |
| out | PrintWriter | 输出流 |
| pm | IntervalLogger | 登录间隔计数器 |
| logKey | String | 登录关键字 |

**作用描述：**

这个方法获取消息信道，同时为多个登录成功的用户发送消息。

**使用方法：**

对于一个实例化的MessageConsumer对象，可以直接调用该方法，该方法输入四个参数，分别代表要发送的消息、输出流、登录间隔计数器、登录关键字，没有返回值，形如：

MessageConsumer msgConsumer = new MessageConsumer(“Hello World!”, printWriter, pm, “login\_success”);

**注意事项：**

在实例化MessageConsumer对象的时候，应传入四个参数。

### (2) sendMessage(String msg)

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 含义 |
| msg | String | 消息的内容 |

**作用描述：**

该方法通过输出流来发送消息。

**使用方法：**

对于一个实例化的MessageConsumer对象，可以直接调用该方法，该方法有一个string类型的输入参数，代表要发送的消息，没有返回值，形如：

msgConsumer.sendMessage(“Hello World!”);

**注意事项：**

传入的参数必须为string类型。

### (3) handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| consumerTag | String | 消费者标签 |
| envelope | Envelope | 信封 |
| properties | AMQP.BasicProperties | AMQP基本属性 |
| body | byte[] | 内容 |

**作用描述：**

当消费者接收到消息后会自动调用该方法，对消息数进行加一。

**使用方法：**

对于一个实例化的MessageConsumer对象，可以直接调用该方法，输入四个参数，返回值为空，形如：

msgConsumer.handleDelivery(consumerTag, envelop, properties, body);

**注意事项：**

消费者接受消息后会自动调用该方法，无需再特意进行。

### (4) MessageDeparturer(Message msg, PrintWriter out, Logger logger, String logKey)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | Message | 消息内容 |
| out | PrintWriter | 输出流 |
| logger | Logger | 登录计数器 |
| logKey | String | 登录关键字 |

**作用描述：**

当server给消息的consumer发送消息之后，将消息从队列中清除。

**使用方法：**

对于一个实例化的MessageDeparturer对象，可以直接调用该方法，输入四个参数，返回值为空，形如：

MessageDeparturer msgDeparturer = MessageDeparturer(“Hello World!”, printWriter, logger, forwarded\_message);

**注意事项：**

注意四个参数的输入格式。

# Demo

package server;  
  
import MessageUtils.Message;  
import MessageUtils.MessageDeparturer;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Created by shieh on 3/20/16.  
 \*/*class ServerThread extends Thread {  
 private Socket client;  
 private PrintWriter out;  
 private BufferedReader in;  
 public Message msg;  
 public MessageDeparturer messageDeparturer;  
 String exchangeName = "test";  
 String queueName;  
  
 public ServerThread(Socket s)throws IOException {  
 this.client = s;  
 out = new PrintWriter(client.getOutputStream(),true);  
 in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
 start();  
 }  
  
 @Override  
 public void run() {  
 try {  
  
 // 以线程ID标示queue。  
 queueName = String.valueOf(this.getId());  
 // 初始化。  
 msg = new Message("{}", this.getId());  
 // 设置字段。  
 msg.setValue("username", username);  
 msg.setValue("target", "others");  
 // 绑定指定频道。  
 msg.init(queueName, "localhost");  
 msg.bindTo(exchangeName, queueName);  
 // 为这个message对象建立departurer，信息会由其自动分发。  
 messageDeparturer = new MessageDeparturer(msg, out, logger, forwarded\_msg);  
 msg.setValue("event", "logedin");  
 // 将信息分发给同一exchange中的其她所有人。  
 msg.publishToAll(exchangeName);  
  
 String line = in.readLine();  
 msg.reset(line);  
 while(!"logout".equals(msg.getValue("event"))) {  
 //查看在线用户列表  
 if ("showuser".equals(msg.getValue("event"))) {  
 out.println(listOnlineUsers());  
 } else if ("message".equals(msg.getValue("event"))) {  
 msg.setValue("username", username);  
 msg.setValue("event", "message");  
 msg.setValue("target", "others");  
 msg.publishToAll(exchangeName);  
 }  
 line = in.readLine();  
 msg.reset(line);  
 }  
 msg.setValue("target", "all");  
 msg.setValue("event", "quit");  
 msg.publishToAll(exchangeName);  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 client.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}