厦門大學


# 软件学院

## 《中间件技术》大作业报告（二）

题　　目 为ActiveMQ添加一个自定义的Message

姓　　名 陈澄

学　　号 32420212202930

班　　级 软工三班

实验时间 2024/05/09

**2024** 年 **05** 月 **09** 日

# 1 实验题目

为 ActiveMQ 添加一个自定义的 Message 并支持其收发等主要功能

# 2 本次实验目的

分析 ActiveMQ 中的 Message 结构，并且添加一个自定义的 Message（ActiveMQFileMessage）

# 3 实验步骤

1.ActiveMQMessage 结构

ActiveMQMessage 包含五个属性：

其中第一个为消息的数据结构类型，用于消息处理过程中标识消息类型。

第二个为 DLQ 死信队列，即传输失败的属性名称。

第三个为 Broker 路径属性。

第四个为消息携带的各种属性信息，即 Properties。

最后一个为回调的确认参数。

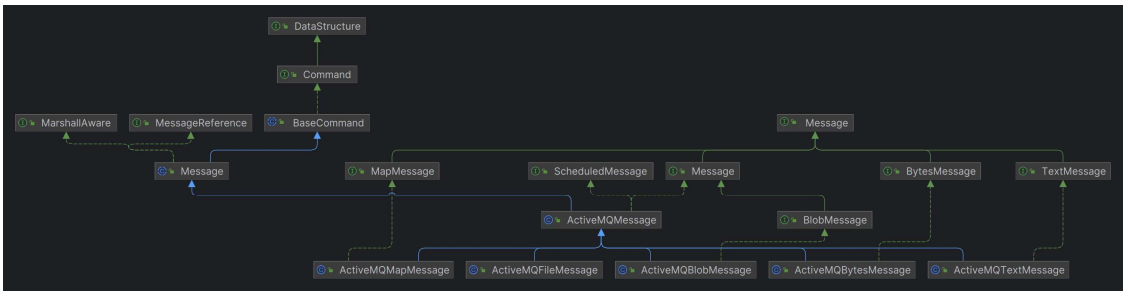
```
public static final byte DATA_STRUCTURE_TYPE = CommandTypes.ACTIVEMQ_MESSAGE;
0 个用法
public static final String DLQ_DELIVERY_FAILURE_CAUSE_PROPERTY = "dlqDeliveryFailureCause";
0 个用法
public static final String BROKER_PATH_PROPERTY = "JMSActiveMQBrokerPath";

13 个用法
private static final Map<String, PropertySetter> JMS_PROPERTY_SETERS = new HashMap<~>();

6 个用法
protected transient Callback acknowledgeCallback;
```

2.ActiveMQMessage 总体架构

新增的新类型 Message 需要继承自 ActiceMQ 风格的 ActiveMQMessage 也要实现 jms 规范的 Message 接口。




3.编写 ActiceMQFileMessage 的主要数据结构

第一个字段 DATA_STRUCTURE_TYPE 为 ActiceMQ 中定义的数据结构类型，会用于接收 Message 时取得其类型。

第二个字段即想要存储的信息，这里以文件为例。

```
public class ActiveMQFileMessage extends ActiveMQMessage{
    public static final byte DATA_STRUCTURE_TYPE = CommandTypes.ACTIVEMQ_FILE_MESSAGE;

    protected File file;
```

4.Copy()函数

用于将 jms 风格的 Message 复制转化为 ActiveMQ 风格的 Message。

```
@Override
public Message copy() {
    ActiveMQFileMessage copy = new ActiveMQFileMessage();
    copy(copy);
    return copy;
}

1 个用法
private void copy(ActiveMQFileMessage copy) {
    super.copy(copy);
    copy.file = file;
}
```

5.字段获取方法 getFile()区别与一般的字段获取方法，此处的 get 方法是为了从 Message 中的 content 中的二进制数据流取得 file 字段并存入 file。

```java
public File getFile() throws JMSException {
    ByteSequence content = getContent();

    if (file == null && content != null) {
        file = decodeContent(content);
        setContent(null);
        setCompressed(false);
    }
    return file;
}
```

6.字段写入方法 setFile()，与上个方法的过程相反

```java
public void setFile(File file) throws MessageNotWriteableException {
    checkReadOnlyBody();
    this.file = file;
    setContent(null);
}
```

7.storeContent()方法，即 setFile()主要逻辑的具体实现，将 File 对象转化为数据流。

```java
@Override
public void storeContent() {
    try {
        ByteSequence content = getContent();
        File file = this.file; // 假设有一个File对象引用
        if (content == null && file != null) {
            ByteArrayOutputStream bytesOut = new ByteArrayOutputStream();
            OutputStream os = bytesOut;
            ActiveMQConnection connection = getConnection();
            if (connection != null && connection.isUseCompression()) {
                compressed = true;
                os = new DeflaterOutputStream(os);
            }
            FileInputStream fis = new FileInputStream(file);
            byte[] buffer = new byte[1024]; // 读取文件的缓冲区
            int bytesRead;
            while ((bytesRead = fis.read(buffer)) != -1) {
                os.write(buffer, off: 0, bytesRead); // 将文件内容写入输出流
            }
            fis.close(); // 关闭文件输入流
            os.close(); // 关闭输出流
            setContent(bytesOut.toByteSequence()); // 将输出流的内容设置为消息的内容
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

8.与上个方法相反的过程。

```
private File decodeContent(ByteSequence bodyAsBytes) throws JMSException {
    File file = null;
    if (bodyAsBytes != null) {
        FileOutputStream fos = null;
        try {
            file = new File( pathname: "");
            fos = new FileOutputStream(file);
            fos.write(bodyAsBytes.getData(), bodyAsBytes.getOffset(), bodyAsBytes.getLength());
        } catch (IOException ioe) {
            throw JMSExceptionSupport.create(ioe);
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    // ignore
                }
            }
        }
    }
    return file;
}
```

9.beforeMarshall()在序列化之前会调用的方法，会将 file 存入 content 并且清空 file。

```
@Override
public void beforeMarshall(WireFormat wireFormat) throws IOException {
    super.beforeMarshall(wireFormat);
    storeContentAndClear();
}

2 个用法
@Override
public void storeContentAndClear() {
    storeContent();
    file=null;
}
```

10.清除未序列化状态（成功序列化之后调用）

```
6 个用法
@Override
public void clearUnMarshalledState() throws JMSException {
    super.clearUnMarshalledState();
    this.file = null;
}
```

11.是否成功序列化

```
@Override
public boolean isContentMarshalled() { return content != null || file == null; }
```

12.获得消息大小

加上所有非空属性字段的大小和文件大小

```
@Override
public int getSize() {
    File file = this.file;
    if (size == 0 && content == null && file != null) {
        size = getMinimumMessageSize();
        if (marshalledProperties != null) {
            size += marshalledProperties.getLength();
        }
        size += file.length();
    }
    return super.getSize();
}
```

13.toString()方法

```
@Override
public String toString() {
    try {
        String fileDetails = this.file.getAbsolutePath();  // 假设存在一个获取文件路径的方法
        if (fileDetails != null) {
            HashMap<String, Object> overrideFields = new HashMap<~>();
            overrideFields.put("fileDetails", fileDetails);
            return super.toString(overrideFields);
        }
    } catch (Exception e) {
    }
    return super.toString();
}
```

14.屏蔽强制转化错误，并且复制信息转为对应类型。

消息发送和接收时存在非 ActiveMQ 类型与 ActiveMQ 类型的双向转化。