厦門大學

软件学院

《中间件技术》大作业报告（三）

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 2024/05/23

2024 年 **05** 月 **23** 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

完善自定义 Message 发送接收过程中需要的方法和定义等。

3 实验步骤

1. CommandType 中增加 ActiveMQFileMessage 的数据格式定义

```
4 个用法
byte ACTIVEMQ_MESSAGE = 23;
3 个用法
byte ACTIVEMQ_BYTES_MESSAGE = 24;
2 个用法
byte ACTIVEMQ_MAP_MESSAGE = 25;
4 个用法
byte ACTIVEMQ_OBJECT_MESSAGE = 26;
3 个用法
byte ACTIVEMQ_STREAM_MESSAGE = 27;
3 个用法
byte ACTIVEMQ_TEXT_MESSAGE = 28;
3 个用法
byte ACTIVEMQ_BLOB_MESSAGE = 29;

2 个用法
byte ACTIVEMQ_FILE_MESSAGE = 35;

// ////////////////////////////////////////////////
//
// Command Response messages
//
// ////////////////////////////////////////////////
1 个用法
```

（将 Responce 定义全体+1，将数据定义后移会导致链接无法正常建立，目前原因不明，这并不符合面向对象的设计原则？）

```
1 个用法
byte RESPONSE = 30;
1 个用法
byte EXCEPTION_RESPONSE = 31;
1 个用法
byte DATA_RESPONSE = 32;
1 个用法
byte DATA_ARRAY_RESPONSE = 33;
1 个用法
byte INTEGER_RESPONSE = 34;
```

2. 消息发送过程中需要调用 OpenWireFormat 类中的 Marshal 方法进行序列化

```
@Override
public synchronized ByteSequence marshal(Object command) throws IOException {

    if (cacheEnabled) {
        runMarshallCacheEvictionSweep();
    }

    ByteSequence sequence = null;
    int size = 1;
    if (command != null) {

        DataStructure c = (DataStructure)command;
        byte type = c.getDataStructureType();
        DataStreamMarshaller dsm = dataMarshallers[type & 0xFF];
        if (dsm == null) {
            throw new IOException("Unknown data type: " + type);
        }
        if (tightEncodingEnabled) {

            BooleanStream bs = new BooleanStream();
```

在这里该方法往往能够调用特定的类对特定的 Message 进行序列化

例如 TextMessage 能够调用 ActiveMQTextMessageMarshaller 类进行序列化，因此需要找到该数组的存储，为其添加数据类型对应的序列化类。

```
byte type = c.getDataStructureType();   type: 28
DataStreamMarshaller dsm = this.dataMarshallers[type & 255];   dsm: ActiveMQTextMessageMarshaller@1825
if (dsm == null) {
    throw new IOException("Unknown data type: " + type);   type: 28
}

if (this.tightEncodingEnabled) {
    BooleanStream bs = new BooleanStream();   bs: BooleanStream@1826
    size += dsm.tightMarshal1( openWireFormat: this, c, bs);   size: 1   bs: BooleanStream@1826   c: "Active
    size += bs.marshalledSize();
    if (this.maxFrameSizeEnabled && (long) size > this.maxFrameSize) {
```

3. 根据 2 中的分析找到 MarshallerFactory 中定义了该数组，长度为 256。可见此处使用了工厂方法的设计模式。

但是类的添加是无序的，为何能取得对应的序列化类？

```
public class MarshallerFactory {

    /**
     * Creates a Map of command type -> Marshallers
     */
    2 个用法
    static final private DataStreamMarshaller marshaller[] = new DataStreamMarshaller[256];
    static {

        add(new ActiveMQBlobMessageMarshaller());
        add(new ActiveMQBytesMessageMarshaller());
        add(new ActiveMQMapMessageMarshaller());
        add(new ActiveMQMessageMarshaller());
        add(new ActiveMQObjectMessageMarshaller());
        add(new ActiveMQQueueMarshaller());
        add(new ActiveMQStreamMessageMarshaller());
        add(new ActiveMQTempQueueMarshaller());
```

Add()方法是根据 DataStructureType 作为 key 存入的，因此存入顺序并不影响。

```
58 个用法
static private void add(DataStreamMarshaller dsm) { marshaller[dsm.getDataStructureType()] = dsm; }

static public DataStreamMarshaller[] createMarshallerMap(OpenWireFormat wireFormat) { return marshaller; }
```

4. 在末尾添加对应的类

```
        add(new SessionIdMarshaller());
        add(new SessionInfoMarshaller());
        add(new ShutdownInfoMarshaller());
        add(new SubscriptionInfoMarshaller());
        add(new TransactionInfoMarshaller());
        add(new WireFormatInfoMarshaller());
        add(new XATransactionIdMarshaller());
        add(new ActiveMQFileMessageMarshaller());
    }
```

5. 编写对应的序列化类

```
1 个用法
public class ActiveMQFileMessageMarshaller extends ActiveMQMessageMarshaller{

    public byte getDataStructureType() { return ActiveMQFileMessage.DATA_STRUCTURE_TYPE; }

    public DataStructure createObject() { return new ActiveMQFileMessage(); }

    public void tightUnmarshal(OpenWireFormat wireFormat, Object o, DataInput dataIn, BooleanStream bs) throws IOException {
        super.tightUnmarshal(wireFormat, o, dataIn, bs);
    }

    public int tightMarshal1(OpenWireFormat wireFormat, Object o, BooleanStream bs) throws IOException {

        int rc = super.tightMarshal1(wireFormat, o, bs);

        return rc + 0;
    }

    public void tightMarshal2(OpenWireFormat wireFormat, Object o, DataOutput dataOut, BooleanStream bs) throws IOException {
        super.tightMarshal2(wireFormat, o, dataOut, bs);

    }

    public void looseUnmarshal(OpenWireFormat wireFormat, Object o, DataInput dataIn) throws IOException {
        super.looseUnmarshal(wireFormat, o, dataIn);

    }

    public void looseMarshal(OpenWireFormat wireFormat, Object o, DataOutput dataOut) throws IOException {

        super.looseMarshal(wireFormat, o, dataOut);

    }
}
```

6. 在 JMSMappingOutboundTransformer 中添加对应的数据编码方案

```
} else if (messageType == CommandTypes.ACTIVEMQ_FILE_MESSAGE) {
    switch (orignalEncoding) {
        case AMQP_NULL:
        case AMQP_DATA:
        case AMQP_VALUE_STRING:
        case AMQP_UNKNOWN:
        default:
            body = new AmqpValue(((ActiveMQFileMessage) message).getFile());
            break;
    }
```