Mindroid Workshop Aufgabenstellung mit Lösungen

TECHNISCHE UNIVERSITÄT DARMSTADT

NeXT Generation on Campus
TU Darmstadt

LÖSUNGEN

Aufgabe 1 Wand-Ping-Pong

Nutze deine Kenntnisse, um den Roboter wie einen Ping-Pong-Ball in gerader Linie zwischen zwei Wänden/Gegenständen/... hin und her fahren zu lassen. Anweisungen:

1. Der Roboter soll solange geradeaus fahren, bis er eine Wand erkennt. Er soll dabei so nah wie möglich an die Wand heranfahren, ohne mit ihr zu kollidieren.
2. Dann soll er ein kleines Stück rückwärts fahren und sich um 180° drehen
3. Beginne bei 1.

Listing 1: `SingleWallPingPong.java`

```java
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;

public class SingleWallPingPong extends ImperativeWorkshopAPI {


    public SingleWallPingPong() {
        super("Single Wall Ping-Pong [sol]");
    }

    @Override
    public void run() {
        do {
            forward(500);
            while (getDistance() > 15f && !isInterrupted()) {
                delay(25);
            }
            stop();
            driveDistanceBackward(10);
            turnLeft(180);
        }while(!isInterrupted());
    }
}
```

Aufgabe 2 Koordiniertes Wand-Ping-Pong

Diese Aufgabe lehnt sich an die vorherige an. Allerdings sollen sich nun zwei Roboter abstimmen. Beide Roboter starten nebeneinander und blicken in die gleiche Richtung. Anweisungen:

1. Roboter A fährt solange, bis er eine Wand entdeckt. Er setzt zurück, dreht sich um und bleibt stehen.
2. Roboter A sendet Roboter B eine “Start”-Nachricht.
3. Daraufhin setzt sich Roboter B in Bewegung, bis er auf die Wand trifft. Daraufhin setzt Roboter B zurück, wendet und bleibt stehen.
4. Roboter B sendet Roboter A eine “Weiter”-Nachricht.
5. Beginne bei 1.

Listing 2: CoordWallPingPong.java

```java
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.common.messages.server.MindroidMessage;
import org.mindroid.impl.brick.Button;

public class CoordWallPingPong extends ImperativeWorkshopAPI {

    private final String PLAYER_1 = "Alice";
    private final String PLAYER_2 = "Bob";

    //Messages
    private final String LEADER_MSG = "I AM THE LEADER";
    private final String START_MSG = "START!";
    private final String CONTINUE_MSG = "WEITER!";

    public CoordWallPingPong() {
        super("Coord Wall Ping-Pong Dynamic Leader [sol]", 2);
    }

    @Override
    public void run() {
        String myID = getRobotID();
        String colleague;

        if(myID.equals(PLAYER_1)){
            colleague = PLAYER_2;
        }else{
            colleague = PLAYER_1;
        }

        sendLogMessage("I am " + myID);
        sendLogMessage("My Colleague is " + colleague);

        boolean leaderElectionFinished = false;

        while(!leaderElectionFinished && !isInterrupted()){
            if(isButtonClicked(Button.ENTER)){
                sendLogMessage("I am the leader!");
                //I am the Leader
                sendMessage(colleague, LEADER_MSG);
                leaderElectionFinished = true;

                //Start doing wall ping pong, start driving
                while(!isInterrupted()) {
                    driveToWallAndTurn();
                    sendMessage(colleague, START_MSG);
                    waitForMessage(CONTINUE_MSG);
                }
            }

            if(hasMessage()){
                MindroidMessage msg = getNextMessage();
```

```
                sendLogMessage("I received a message: "+msg.getSource().getValue()+":
                    ↪  \""+msg.getContent()+"\"");
                if(msg.getContent().equals(LEADER_MSG)){
                    //Colleague is the leader
                    leaderElectionFinished = true;
                    sendLogMessage("I am NOT the leader!");

                    // do wall-pingpong, start with waiting
                    while(!isInterrupted()){
                        waitForMessage(START_MSG);
                        driveToWallAndTurn();
                        sendMessage(colleague, CONTINUE_MSG);
                    }
                }
            }
            delay(50);
        }
    }


    private void driveToWallAndTurn(){
        forward(300);
        while (!isInterrupted() && getDistance() > 10f) {
            delay(10);
        }
        driveDistanceBackward(10);
        turnLeft(180);
    }

    private void waitForMessage(String message){
        sendLogMessage("Warte auf: \"" + message + "\"");
        while (!isInterrupted()) {
            if (hasMessage()) {
                if (getNextMessage().getContent().equals(message)){
                    return;
                }
            }
            delay(10);
        }
    }
}
```

Listing 3: CoordWallPingPongA.java

```
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;

public class CoordWallPingPongA extends ImperativeWorkshopAPI{

    public CoordWallPingPongA(){
        super("Coord Wall Ping-Pong Alice [sol]", 2);
    }

    String colleague = "Bob";

```

```
    @Override
    public void run(){
        while(!isInterrupted()) {
            driveToWallAndTurn();
            sendMessage(colleague, "Start!");
            waitForMessage("Weiter!");
        }
    }

    private void driveToWallAndTurn(){
        forward(300);
        while (!isInterrupted() && getDistance() > 10f) {
            delay(10);
        }
        driveDistanceBackward(10);
        turnLeft(180);
    }

    private void waitForMessage(String message){
        while (!isInterrupted()) {
            if (hasMessage()) {
                if (getNextMessage().getContent().equals(message)){
                    return;
                }
            }
            delay(10);
        }
    }
}
```

Listing 4: CoordWallPingPongB.java

```
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;

public class CoordWallPingPongB extends ImperativeWorkshopAPI{

    public CoordWallPingPongB() {
        super("Coord Wall Ping-Pong Bob [sol]", 2);
    }

    String colleague = "Alice";

    @Override
    public void run(){
        while(!isInterrupted()){
            waitForMessage("Start!");
            driveToWallAndTurn();
            sendMessage(colleague, "Weiter!");
        }
    }

    private void driveToWallAndTurn(){
        forward(300);
        while (!isInterrupted() && getDistance() > 10f) {
```

```
            delay(10);
        }
        driveDistanceBackward(10);
        turnLeft(180);
    }

    private void waitForMessage(String message){
        while (!isInterrupted()) {
            if (hasMessage()) {
                if (getNextMessage().getContent().equals(message)){
                    return;
                }
            }
            delay(10);
        }
    }
}
```

Aufgabe 3 Mähroboter

Nutze deine Kenntnisse, um den Roboter in einem mit schwarzem (oder weißem) Klebeband abgesperrten Bereich herumfahren zu lassen (so ähnlich wie beispielsweise viele Mähroboter arbeiten).

1. Wie beim Wand Ping-Pong soll der Roboter erstmal geradeaus fahren.
2. Wenn er eine Grenze erkennt, soll er zurücksetzen und sich eine neue Richtung aussuchen.
3. Beginne bei 1.

Tipp: Überprüfe, welche Color-IDs auf dem Boden und den Begrenzungen erkannt werden, um festzustellen, wann der Roboter an die Umzäunung gelangt ist.

Listing 5: `LawnMower.java`

```
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.impl.statemachine.properties.Colors;

public class LawnMower extends ImperativeWorkshopAPI {

    private static Colors tapeColor = Colors.GREEN;
    private static float backDist = 10.0f;

    public LawnMower() {
        super("Lawn Mower [sol]");
    }

  @Override
    public void run() {
        while(!isInterrupted()){
            setMotorSpeed(200);
            forward();
            if(getLeftColor()== tapeColor && getRightColor() == tapeColor){
```

```
                driveDistanceBackward(backDist);
                turnRight(135);
            }else if(getLeftColor()== tapeColor){
                driveDistanceBackward(backDist);
                turnRight(90);
            }else if(getRightColor()== tapeColor){
                driveDistanceBackward(backDist);
                turnLeft(90);
            }
            delay(50);
        }
    }
}
```

Aufgabe 4 Platooning

In dieser Aufgabe geht es darum, zwei Roboter hintereinander her fahren zu lassen, ohne dass es einen Auffahrunfall gibt. Aktuell forschen zahlreiche Unis und Unternehmen unter dem Schlagwort Platooning an genau dieser Problemstellung bei echten LKWs und PKWs: Die Fahrzeuge fahren dabei so nahe, dass sie den Windschatten des Vorausfahrenden ausnutzen können.
Roboter A und B werden hintereinander platziert, sodass sie in die gleiche Richtung blicken. Ziel ist es zunächst, dass Roboter B den Abstand zu Roboter A in einem bestimmen Toleranzbereich hält. Die Distanzangaben im Folgenden sind nur mögliche Werte - du bestimmst selbst, was geeignete Grenzwerte sind.

1. Roboter A fährt los. Sobald der Abstand zwischen Roboter A und Roboter B größer als 35cm wird, beginnt Roboter B aufzuschließen.
2. Wird der Abstand kleiner als 25cm, hält Roboter B die Geschwindigkeit von Roboter A .
3. Wird der Abstand kleiner als 15cm, lässt Roboter B sich zurückfallen (oder fährt sogar rückwärts).

Listing 6: `Platooning.java`

```
package org.mindroid.android.app.programs.workshop.solutions;

import java.util.Random;
import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.api.ev3.EV3StatusLightColor;
import org.mindroid.api.ev3.EV3StatusLightInterval;
import org.mindroid.common.messages.server.MindroidMessage;
import org.mindroid.impl.brick.Button;
import org.mindroid.impl.brick.Textsize;

public class Platooning extends ImperativeWorkshopAPI {

    public Platooning(){
        super("Platooning Dynamic Leader [sol]", 2);
    };
    enum State {
        FAST,
```

```
        MED,
        SLOW
    }
    State prevState;

    private final String player_1 = "Alice";
    private final String player_2 = "Bob";
    private final String myID = getRobotID();
    private String colleague;

    //Messages
    private final String leaderMsg = "I AM THE LEADER";

    @Override
    public void run() {

        // find out who i am, so i know who my colleague is
        if(myID.equals(player_1)){
            colleague = player_2;
        }else{
            colleague = player_1;
        }

        while(!isInterrupted()) {

            if (isButtonClicked(Button.ENTER)) {
                sendLogMessage("I am the leader!");
                sendMessage(colleague, leaderMsg);
                driveAsLeader();
            }
            if (hasMessage()) {
                MindroidMessage msg = getNextMessage();
                sendLogMessage("I received a message: " + msg.getSource().getValue()
                    ↪ + ": \"" + msg.getContent() + "\"");
                if (msg.getContent().equals(leaderMsg)) {
                    //Colleague is the leader
                    sendLogMessage("I am NOT the leader!");
                    driveAsFollower();
                }
            }
        }
    }

    private void driveAsLeader(){
        setMotorSpeed(200);
        forward();
        while (!isInterrupted()) {
            delay(50);
        }
        stop();
    }
    private void driveAsFollower(){
        while(!isInterrupted()) {
            clearDisplay();
            float distance = getDistance();
```

```
            drawString("Dist: " + distance);
            if (prevState != State.FAST && distance > 30f) {
                forward(300);
                prevState = State.FAST;
                setLED(LED_GREEN_ON);
            } else if (prevState != State.SLOW && distance < 20f) {
                forward(100);
                prevState = State.SLOW;
                setLED(LED_RED_ON);
            } else if (prevState != State.MED && distance > 20f && distance < 30f) {
                forward(200);
                prevState = State.MED;
                setLED(LED_YELLOW_ON);
            }
            delay(50);
        }
        stop();

    }
}
```

Listing 7: PlatooningLeader.java

```
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.api.ev3.EV3StatusLightColor;
import org.mindroid.api.ev3.EV3StatusLightInterval;
import org.mindroid.common.messages.server.MindroidMessage;
import org.mindroid.impl.brick.Button;
import org.mindroid.impl.brick.Textsize;

public class PlatooningLeader extends ImperativeWorkshopAPI {

    public PlatooningLeader() {
        super("Platooning Leader [sol]", 2);
    }

    @Override
    public void run() {
        while (!isInterrupted()) {
            if (isButtonClicked(Button.ENTER)) {
                setMotorSpeed(200);
                forward();
                while (!isInterrupted()) {
                    delay(50);
                }
                stop();
            }
        }
    }
}
```

Listing 8: PlatooningFollower.java

```
package org.mindroid.android.app.programs.workshop.solutions;
```

```
2
3  import org.mindroid.api.ImperativeWorkshopAPI;
4  import org.mindroid.api.ev3.EV3StatusLightColor;
5  import org.mindroid.api.ev3.EV3StatusLightInterval;
6  import org.mindroid.common.messages.server.MindroidMessage;
7  import org.mindroid.impl.brick.Button;
8  import org.mindroid.impl.brick.Textsize;
9
10 public class PlatooningFollower extends ImperativeWorkshopAPI {
11
12     public PlatooningFollower(){
13         super("Platooning Follower [sol]", 2);
14     }
15
16     enum State {
17         FAST,
18         MED,
19         SLOW
20     }
21     State prevState;
22
23     @Override
24     public void run() {
25       while(!isInterrupted()) {
26             clearDisplay();
27             float distance = getDistance();
28             drawString("Dist: " + distance);
29             if (prevState != State.FAST && distance > 30f) {
30                 forward(300);
31                 prevState = State.FAST;
32                 setLED(LED_GREEN_ON);
33             } else if (prevState != State.SLOW && distance < 20f) {
34                 forward(100);
35                 prevState = State.SLOW;
36                 setLED(LED_RED_ON);
37             } else if (prevState != State.MED && distance > 20f && distance < 30f) {
38                 forward(200);
39                 prevState = State.MED;
40                 setLED(LED_YELLOW_ON);
41             }
42             delay(50);
43         }
44         stop();
45     }
46 }
```

Aufgabe 5 Dancing Robots

Beim Cha-Cha-Cha gibt es die Tanzfigur “Verfolgung”. Dabei verfolgt jeweils ein Tanzpartner den anderen, bis beide sich umdrehen und die Rollen wechseln. Diese Figur ist tatsächlich nicht sehr weit vom Platooning-Beispiel aus der vorherigen Aufgabe entfernt. Der Ablauf soll dieses Mal wie folgt aussehen:

1. Roboter A übernimmt zunächst das Kommando und fährt voraus, während Roboter B einen möglichst gleichbleibenden Abstand hält.

2. Roboter A beschließt nach einer gewissen Zeit, dass nun die Drehung folgt. Er stoppt und sendet eine “Drehen”-Nachricht an Roboter B.

3. Roboter B stoppt, wendet und sendet Roboter A eine “Gedreht”-Nachricht.

4. Daraufhin dreht Roboter A ebenfalls um 180°.

5. Nun tauschen Roboter A und B die Rollen: Roboter B fährt voraus und gibt den Ton bis zur nächsten Drehung an.

Listing 9: `Follow.java`

```java
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.common.messages.server.MindroidMessage;
import org.mindroid.impl.brick.Button;

import java.util.Random;

public class Follow extends ImperativeWorkshopAPI {

    public Follow() {
        super("Follower Dynamic Leader [sol]",2);
    }
    enum PlatoonState {
        FAST,
        MED,
        SLOW
    }
    private PlatoonState prevState;

    final int LEADER = 0;
    final int FOLLOWER = 1;
    int role = -1;


    private String colleague;

    private String player_1 = "Alice";
    private String player_2 = "Bob";


    //Messages
    private final String leaderMsg = "I AM THE LEADER";
    private final String turnMsg = "TURN!";
    private final String turnedMsg = " TURNED";
    private String myTurnedMsg;
    private String otherTurnedMsg;

    @Override
    public void run() {

        String myID = getRobotID();
        //String colleague;
```

```
44
45          // find out who i am, so i know who my colleague is
46          if(myID.equals(player_1)){
47              colleague = player_2;
48          }else{
49              colleague = player_1;
50          }
51
52          myTurnedMsg = myID + turnedMsg;
53          otherTurnedMsg = colleague + turnedMsg;
54          boolean initDone = false;
55          // get Roles
56          while(!isInterrupted() && !initDone ) {
57              if (isButtonClicked(Button.ENTER)) {
58                  sendLogMessage("I am the leader!");
59                  sendMessage(colleague, leaderMsg);
60                  role = LEADER;
61                  initDone = true;
62              }
63              if (hasMessage()) {
64                  MindroidMessage msg = getNextMessage();
65                  sendLogMessage("I received a message: " + msg.getSource().getValue()
                        ↪ + ": \"" + msg.getContent() + "\"");
66                  if (msg.getContent().equals(leaderMsg)) {
67                      //Colleague is the leader
68                      sendLogMessage("I am NOT the leader!");
69                      role = FOLLOWER;
70                  }
71                  initDone = true;
72              }
73              delay(10);
74          }
75
76          // drive with changing roles
77          while(!isInterrupted()){
78              if (role == LEADER){
79                  driveAsLeader();
80              }
81              if (role == FOLLOWER){
82                  driveAsFollower();
83              }
84          }
85      }
86
87      private void driveAsLeader(){
88          sendLogMessage("Leading the way!");
89          setMotorSpeed(200);
90          forward();
91          delay(5000);
92          stop();
93
94          sendMessage(colleague, turnMsg);
95          waitForTurn();
96          turnRight(180, 100);
97          sendMessage(colleague, myTurnedMsg);
```

```
98          role = FOLLOWER;
99      }
100
101     private void driveAsFollower(){
102         sendLogMessage("Following!");
103         while(!isInterrupted()) {
104             keepDistance();
105
106             // check for Turn Message
107             if(hasMessage()){
108                 MindroidMessage msg = getNextMessage();
109                 sendLogMessage("I received: " + msg.getContent());
110                 if( msg.getContent().equals(turnMsg)){
111                     turnLeft(180, 100);
112                     sendMessage(colleague, myTurnedMsg);
113                     waitForTurn();
114                     role = LEADER;
115                     return;
116                 }
117             }
118             delay(50);
119         }
120         stop();
121     }
122
123     private void keepDistance(){
124         float distance = getDistance();
125         if (distance > 35f) {
126             forward(300);
127         } else if (distance < 25f) {
128             forward(100);
129         } else if (distance > 25f && distance < 35f) {
130             forward(200);
131         }
132     }
133
134     private void waitForTurn(){
135         boolean finished = false;
136         while (!isInterrupted() && !finished){
137             delay(50);
138             if(hasMessage()) {
139                 MindroidMessage msg = getNextMessage();
140                 if (msg.getContent().equals(otherTurnedMsg)) {
141                     sendLogMessage("I received: " + msg.getContent());
142                     finished = true;
143                 }
144             }
145         }
146     }
147 }
```

Listing 10: FollowA.java

```
1  package org.mindroid.android.app.programs.workshop.solutions;
2
3  import org.mindroid.api.ImperativeWorkshopAPI;
```

```
4  import org.mindroid.common.messages.server.MindroidMessage;
5
6  public class FollowA extends ImperativeWorkshopAPI {
7
8      public FollowA() {
9          super("Follower Alice [sol]", 2);
10     }
11
12     enum DistState {
13         FAST,
14         MED,
15         SLOW
16     }
17     private DistState prevState;
18
19     // Robot names
20     private String myID = "Alice";
21     private String colleague = "Bob";
22
23     // Messages
24     private final String turnMsg = "TURN!";
25     private final String turnedMsg = " TURNED";
26     private String myTurnedMsg = myID + turnedMsg;
27     private String otherTurnedMsg = colleague + turnedMsg;
28
29
30     @Override
31     public void run(){
32         while(!isInterrupted()) {
33             driveAsLeader();
34             driveAsFollower();
35         }
36     }
37
38
39     private void driveAsLeader(){
40         sendLogMessage("Leading the way!");
41         forward(200);
42         delay(3000);
43         stop();
44
45         sendMessage(colleague, turnMsg);
46         waitForTurn();
47         sendLogMessage("turning...");
48         turnRight(180, 100);
49         sendMessage(colleague, myTurnedMsg);
50     }
51
52     private void driveAsFollower(){
53         sendLogMessage("Following!");
54         while(!isInterrupted()) {
55             keepDistance();
56             // check for Turn Message
57             if(hasMessage()){
58                 MindroidMessage msg = getNextMessage();
```

```
                sendLogMessage("I received: " + msg.getContent());
                if( msg.getContent().equals(turnMsg)){
                    sendLogMessage("turning...");
                    turnLeft(180, 100);
                    sendMessage(colleague, myTurnedMsg);
                    waitForTurn();
                    return;
                }
            }
            delay(50);
        }
        stop();
    }

    private void keepDistance(){
        float distance = getDistance();
        if (prevState != DistState.FAST && distance > 35) {
            forward(300);
            prevState = DistState.FAST;
            setLED(LED_GREEN_ON);
        } else if (prevState != DistState.SLOW && distance < 25) {
            forward(100);
            prevState = DistState.SLOW;
            setLED(LED_RED_ON);
        } else if (prevState != DistState.MED && distance > 25 && distance < 35) {
            forward(200);
            prevState = DistState.MED;
            setLED(LED_YELLOW_ON);
        }
    }

    private void waitForTurn(){
        while (!hasMessage()) delay(50);
        if(hasMessage()){
            MindroidMessage msg = getNextMessage();
            if (msg.getContent().equals(otherTurnedMsg))
                sendLogMessage("I received: " + msg.getContent());
        }
    }
}
```

Listing 11: FollowB.java

```
package org.mindroid.android.app.programs.workshop.solutions;

import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.common.messages.server.MindroidMessage;
import org.mindroid.impl.brick.Button;

import java.util.Random;

public class FollowB extends ImperativeWorkshopAPI {

    public FollowB() {
        super("Follower Bob [sol]", 2);
    }
```

```
14
15      enum DistState {
16          FAST,
17          MED,
18          SLOW
19      }
20      private DistState prevState;
21
22      // Robot names
23      private String myID = "Alice";
24      private String colleague = "Bob";
25
26      // Messages
27      private final String turnMsg = "TURN!";
28      private final String turnedMsg = " TURNED";
29      private String myTurnedMsg = myID + turnedMsg;
30      private String otherTurnedMsg = colleague + turnedMsg;
31
32
33      @Override
34      public void run(){
35          while(!isInterrupted()) {
36              driveAsFollower();
37              driveAsLeader();
38          }
39      }
40
41      private void driveAsLeader(){
42          sendLogMessage("Leading the way!");
43          forward(200);
44          delay(3000);
45          stop();
46
47          sendMessage(colleague, turnMsg);
48          waitForTurn();
49          sendLogMessage("turning...");
50          turnRight(180, 100);
51          sendMessage(colleague, myTurnedMsg);
52      }
53
54      private void driveAsFollower(){
55          sendLogMessage("Following!");
56          while(!isInterrupted()) {
57              keepDistance();
58              // check for Turn Message
59              if(hasMessage()){
60                  MindroidMessage msg = getNextMessage();
61                  sendLogMessage("I received: " + msg.getContent());
62                  if( msg.getContent().equals(turnMsg)){
63                      sendLogMessage("turning...");
64                      turnLeft(180, 100);
65                      sendMessage(colleague, myTurnedMsg);
66                      waitForTurn();
67                      return;
68                  }
```

```
            }
            delay(50);
        }
        stop();
    }

    private void keepDistance(){
        float distance = getDistance();
        if (prevState != DistState.FAST && distance > 35) {
            forward(300);
            prevState = DistState.FAST;
            setLED(LED_GREEN_ON);
        } else if (prevState != DistState.SLOW && distance < 25) {
            forward(100);
            prevState = DistState.SLOW;
            setLED(LED_RED_ON);
        } else if (prevState != DistState.MED && distance > 25 && distance < 35) {
            forward(200);
            prevState = DistState.MED;
            setLED(LED_YELLOW_ON);
        }
    }

    private void waitForTurn(){
        while (!hasMessage()) delay(50);
        if(hasMessage()){
            MindroidMessage msg = getNextMessage();
            if (msg.getContent().equals(otherTurnedMsg))
                sendLogMessage("I received: " + msg.getContent());
        }
    }
}
```