الـ Behaviours في JADE

1. مقدمة:

نحتاج لدى برمجة العملاء باستعمال JADE إضافة مجموعة من المهمات التي يجب على العميل القيام بها، كتابة هذه المهام يجب أن يخضع لبعض الشروط و المقاييس، سنتطرق من خلال هذا المقال إلى أسس كتابة المهمات Behaviour لعملاء JADE

2. نظرة عامة:

لكي يستطيع عميل مبرمج بالـ jade تنفيذ مهمة يجب علينا أولا إنشاء هذه المهمة. المهمات في Jade تسمى بالـ Behaviours و هي عبارة عن Instances من الـ class jade.core.behaviours

لكي يقوم العميل بتنفيذ المهمة تجب إضافتها إليه، وذلك عن طريق الدالة addBehaviour(Behaviour b) المنتمية إلى jade.core.Agent

يجب على كل Behaviour أن يقوم بتعريف على الأقل الدالتين التاليتين:

- Action() : و التي تحوي مجموعة الأوامر التي سيقوم العميل بتنفيذها
- Done(): و التي تخبر ما إذا كان العميل قد أنهى أم لا تنفيذ الـ behaviour

كما أنه توجد دالتان لا تقلان أهمية عن سابقتيهما لكن تبقيان غير ضروريتين للسير الحسن للـ behaviour و هما:

- onStart() : دالة يتم استدعاؤها قبيل استدعاء الدالة action()
- onEnd() : دالة يتم استدعاؤها بعيد إرجاع الدلة done() للقيمة true

أحيانا لدى كتابة behaviour نحتاج إلى استخراج هوية العميل صاحب هذا الـ behaviour و يتم ذلك عن طريق الـ myAgent Member

تجدر الإشارة إلى أن jade ينشئ thread واحد لكل عميل مما يعني أن كل الـ behaviours الخاصة بعميل ما سيتم تنفيذها باستعمال هذا الـ thread و عليه إذا كانت هناك أكثر من behaviour يجب أن تشتغل بطريقة متوازية يجب على المبرمج اختيار الطريقة المثلى لجعلها تشتغل بطريقة متوازية,

3. أنواع الـ behaviours

3.1. الـ behaviours البسيطة

يوفر jade ثلاث أنواع من الـ behaviours البسيطة و هي كالتالي:

3.1.1. One-shot behaviour

هذا النوع من الـ behaviours عبارة عن instance من الـ class المسمى jade.core.behaviours.OneShotBehaviour

يتميز هذا النوع بكونه يقوم بتنفيذ محتوى المهمة مرة واحدة و وحيدة فقط و من ثم ينتهي

الدالة done() المتعلق بهذا الـ class تم تعريفها لكي ترجع دائما القيمة true

3.1.2. Cyclic behaviour

الـ cyclic behaviour عبارة عن instance من الـ class المسمى jade.core.behaviours.CyclicBehaviour

مثلما يظهر من تسميته، فإنه يقوم بتنفيذ المهمة التي يقوم بها بطريقة دورية، حيث أن الدالة done() الخاصة به معدة سلفا لترجع دائما القيمة false

3.1.3. Generic behaviour

هذا النوع من الـ behaviours يأتي متوسطا النوعين السابقين، بحيث أن الدالة done() الخاصة به غير معرفة سلفا بل يجب على المبرمج القيام بتعريفها و عليه فإن ذلك يمكنه من عمل مهمات "أعقد" من التي يمكن عملها بالنوعين السابقين فقط

المثال التالي يوضح كيفية استعمال الـ Behaviours البسيطة

```
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
public class SimpleAgent extends Agent {
        protected void setup() {
// إضافة OneShotBehaviour يقوم بإظهار  //HelloWorld
                addBehaviour(new OneShotBehaviour(this){
                        public void action(){
                                System.out.println("Salam DWord I am
"+getLocalName());
                        }
                });
// إضافة CyclicBehaviour يقوم بإظهار cyclic... كل مرة //
                addBehaviour(new CyclicBehaviour(this) {
                        public void action() {
                                System.out.println("cyclic... ");
                        }
                });
                // إضافة generic behaviour 7 يتوقف لدى إرجاع القيمة//
                addBehaviour(new RandomBehaviour());
        }
        /**
        * Inner class RandomBehaviour
        */
        private class RandomBehaviour extends Behaviour {
                private int RANDOM ;
                public void action() {
                        RANDOM = (int) (Math.random()*10);
                        System.out.println("RANDOM ="+ RANDOM);
                }
                public boolean done() {
                        return RANDOM == 7;
                }
                public int onEnd() {
                        myAgent.doDelete();
                        return super.onEnd();
                }
        }
}
```

3.2. الـ Behaviours الموقوتة

Jade يوفر إمكانية جعل المهمات موقوتة أي أن تنفيذها يكون بعد مضي زمن معين أو تتكرر حسب نظام زمني معين حيث أنه يوفر الصنفين التاليين:

3.2.1. WakerBehaviour

تمت تعريف الصنف wakerBehaviour بطريقة تجعله ينادي الدالة onWake() بعد مرور مدة معينة (ميليثانية)يتم تمريرها كـ parameter للـ consctructor الخاص بالصنف.

يتم الانتهاء من المهمة بمجرد الانتهاء من تنفيذ الدالة onWake()

3.2.2. TickerBehaviour

تمت تعريف الصنف TickerBehaviour بطريقة تجعله ينادي الدالة onTick() بصيغة دورية حيث أن زمن الدورة (ميليثانية)يتم تمريرها كـ parameter للـ consctructor الخاص بالصنف.

المثال التالي يوضح طريقة عمل عداد زمني عكسي

تقوم المهمة الأولى بإظهار الوقت المتبقي

أما المهمة الثانية فتوقف العداد و تنهي عمل العميل

```
import jade.core.Agent;
import jade.core.behaviours.WakerBehaviour;
import jade.core.behaviours.TickerBehaviour;
public class CompteaRebours extends Agent {

  protected void setup() {



final int  nbSeconds = (int)(Math.random()*15);
System.out.println("countdown"+nbSeconds);

                                        //يقوم هذا الـ behaviour بإظهار الوقت المتبقي

    addBehaviour(new TickerBehaviour(this, 1000) {
      protected void onTick() {
        System.out.println("..."+(nbSeconds-getTickCount())+" seconds");
      }
    });


                                        //يقوم هذا الـ behaviour بإنهاء العميل

    addBehaviour(new WakerBehaviour(this, 1000*nbSeconds) {
      protected void handleElapsedTimeout() {
        System.out.println("Over");
        myAgent.doDelete();
      }
    });
  }
}
```

3.3. الـ behaviours المركبة:

إن كنت تقول في قرارة نفسك أن النوعين السابقين من الـ behaviours لا يفيان بجميع أغراض المبرمج فأنت محق 100% ، لكن لا تقلق فـ JADE قد وفر نوعا ثالثا من الـ behaviours الذي يعتبر النوع الأعقد و الأهم .

ترث الـ behaviours من هذا النوع من الصنف **jade.core.behaviours.CompositeBehaviour**

كل instance من هذا الصنف عبارة عن behaviour طبعا و لكنه يحتوي على sub behaviours

الدالة action() الخاصة بالـ compositebehaviour معدة سلفا لكي تقوم باستدعاء الدالة action() الخاصة بأحد الـ sub behaviours

لا تقلق عن ترتيب تنفيذ الـ sub behaviours فذلك لا يقع على عاتق المبرمج مباشرة و إنما يتم عن طريق الدالتين **scheduleFirst()**و **scheduleNext()** و اللتان يجب على كل صنف يرث من composite behaviour أن يقوم بتعريفهما,

التعامل مع الصنف compositeBehaviour معقد لهذا وفرت JADE ثلاث أصناف ترث منها يسهل التعامل معها. هذه الأصناف هي:

3.3.1. SequentielBehaviour

مثلما هو ظاهر من اسمه ، الـ sequentielBehaviour يقوم بتنفيذ الـ sub behaviours الخاصة به تباعا ، الواحد تلو الآخر, بحيث أنه لا يقوم بالمرور إلى الـ behaviour ما حتى ينهي الـ behaviour الذي يسبقه عمله ، بعبارة أخرى يجب أن ترجع الدالة done() الخاصة بالـ behaviour القيمة true قبل المرور إلى الـ behaviour القادم.

تضاف الـ sub behaviours إلى الـ sequentielBehaviour بواسطة الدالة addSubBehaviour() . مع العلم أن ترتيب الإضافة هو الترتيب الذي يؤخذ به لتنفيذ الـ behaviours.

مثال:

```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import jade.core.behaviours.WakerBehaviour;
public class Seq extends Agent {
        protected void setup(){
                SequentialBehaviour SeqBehaviour = new
SequentialBehaviour();
                SeqBehaviour.addSubBehaviour(new OneShotBehaviour(){
                        @Override
                        public void action() {
                        System.out.println("the first sub behaviour");
                        }
                });
                SeqBehaviour.addSubBehaviour(new OneShotBehaviour(){
                        @Override
                        public void action() {
                                System.out.println("the second behaviour");
                        }
                });
                SeqBehaviour.addSubBehaviour(new OneShotBehaviour(){
                        @Override
                        public void action() {
                        System.out.println("the last behaviour");
                        myAgent.doDelete();
                        }
                });
                addBehaviour(SeqBehaviour);
        }
}
```

3.3.2. الـ FSM Behaviour

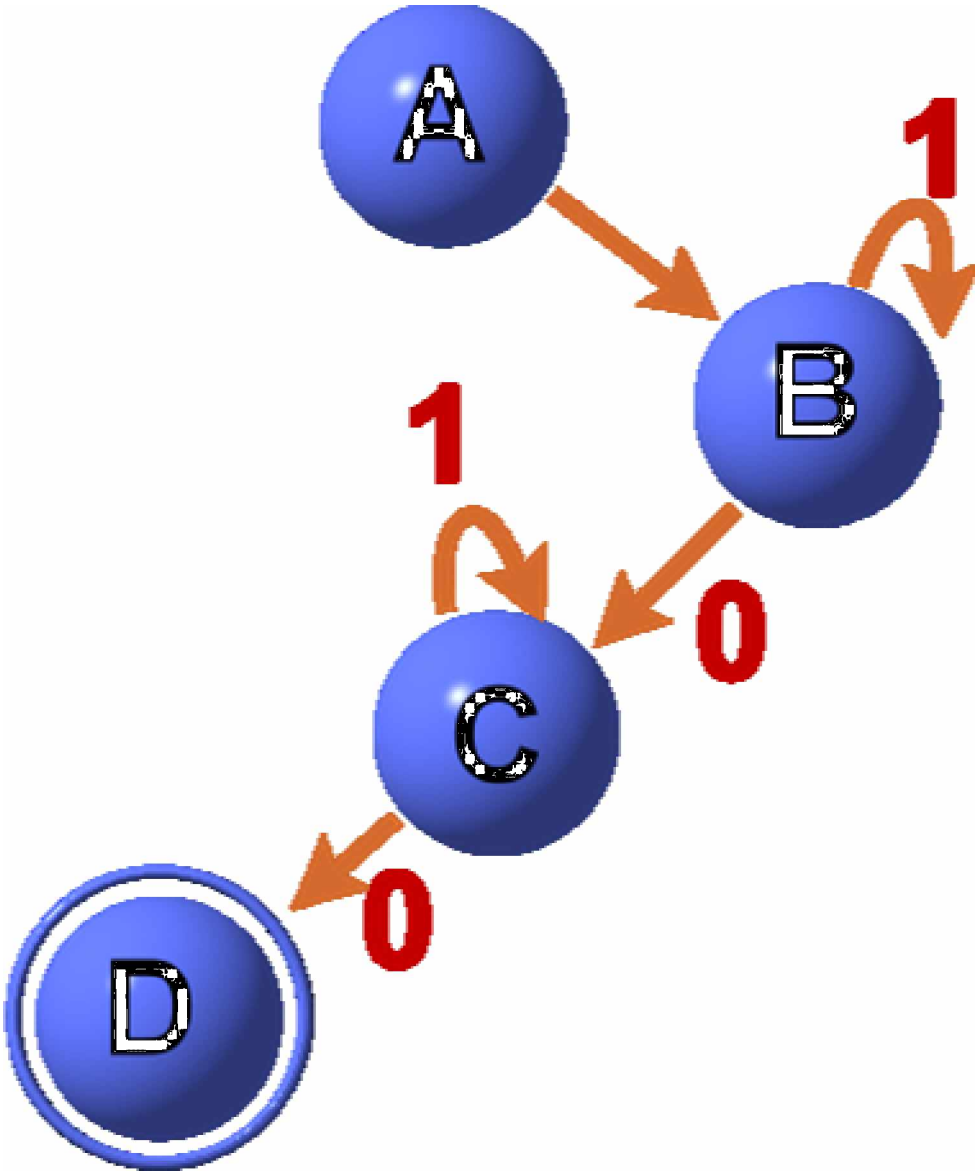
الـ FSM Behaviour عبارة عن behaviour يحوي Finite state machine بداخلها حيث أن كل state عبارة عن behaviour

إنشاء الـ behaviour يكون باستعمال التالي:

- إضافة state جديد يتم عن طريق الدالة registerState (Behaviour state, String name)
- إضافة الـ First state (لا وجود إلى لـ fisr state واحد و وحيد فقط) يتم عن طريق الدالة registerFirstState (Behaviour state, String name)
- إضافة Last State (يمكن لـ FSM أن يحتوي على أكثر من Last State) يتم عن طريق الدالة registerLastState ((Behaviour state, String name
- إضافة transition يتم عن طريق الدالة registerTransition(String s1, String s2, int event)
- إضافة Default transition (المقصود بـ Default transition هي الـ transition من state إلى آخر بغض النظر عن النتيجة التي سيخرج بها الـ state الأول) registerDefaultTransition(String s1, String s2, String[] toBeReset)

مثال:

لنعتبر الـ Finite state machine التالي:



و هذا الـكود الخاص بالـ FSM Behaviour الذي يمثله:

```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.FSMBehaviour;
public class Agent_FSM extends Agent{
        protected void setup(){
                FSMBehaviour fsm = new FSMBehaviour(this) {
                        public int onEnd() {
                                System.out.println("FSM behaviour ...Done");
                                myAgent.doDelete();
                                return super.onEnd();
                        }
                };
                //definiton des etats
                fsm.registerFirstState (new _1behaviour(),"A");
                fsm.registerState(new _1behaviour(),"B");
                fsm.registerState(new _1behaviour(),"C");
                fsm.registerLastState(new _1Otherbehaviour(),"D");
```

```
                //definition des transaction
                fsm.registerDefaultTransition("A","B");
                fsm.registerTransition("B","B",1);
                fsm.registerTransition("B","C",0);
                fsm.registerTransition("C","C",1);
                fsm.registerTransition("C","D",0);
                addBehaviour(fsm);
        }
        //Inner Class
        private class _1behaviour extends OneShotBehaviour{
                int RANDOM;
                @Override
                public void action() {
                        System.out.println("executing  "+
getBehaviourName());
                        RANDOM = (int)Math.random() * 1;
                }
                public int onEnd(){
                return RANDOM;
                }
        }
        private class unAutreComprtement extends OneShotBehaviour{
                @Override
                public void action() {
                        System.out.println("Finale State");
                }
        }
}
```

3.3.3. الـ ParallelBehaviour

يسمح الـ ParallelBehaviour بتنفيذ أكثر من behaviour في آن واحد، المقصود بـ "آن واحد" هنا هو أنه تتم مناداة الدالة action() للـ behaviour التالي دون انتظار نهاية الـ behaviour الحالي.

إضافة الـ behaviours إلى ParallelBehaviour تتم عن طريق الدالة **addSubBehaviour()** .

في حالة ما إذا أردنا أن ينتهي تنفيذ الـ ParallelBehaviour بمجرد انتهاء أحد الـ subBehaviour فإنه يتوجب علينا تمرير الـ parameter **WHEN_ANY** للـ constructor الخاص بالـ ParallelBehaviour ، أما إذا كان على الـ ParallelBehaviour انتظار جميع الـ subBehaviour فإنه في هذه الحالة يجب تمرير الـ parameter **WHEN_ALL**

```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ParallelBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.core.behaviours.WakerBehaviour;
public class agent_parallel extends Agent{
        protected void setup(){
                ParallelBehaviour ParBehaviour = new
ParallelBehaviour(ParallelBehaviour.WHEN_ANY);
                        ParBehaviour.addSubBehaviour(new WakerBehaviour(this,1000){
                        @Override
                        protected void handleElapsedTimeout() {
                                System.out.println("Time is over..");
                                myAgent.doDelete();
                        }
                });
                ParBehaviour.addSubBehaviour(new TickerBehaviour(this,50){
```

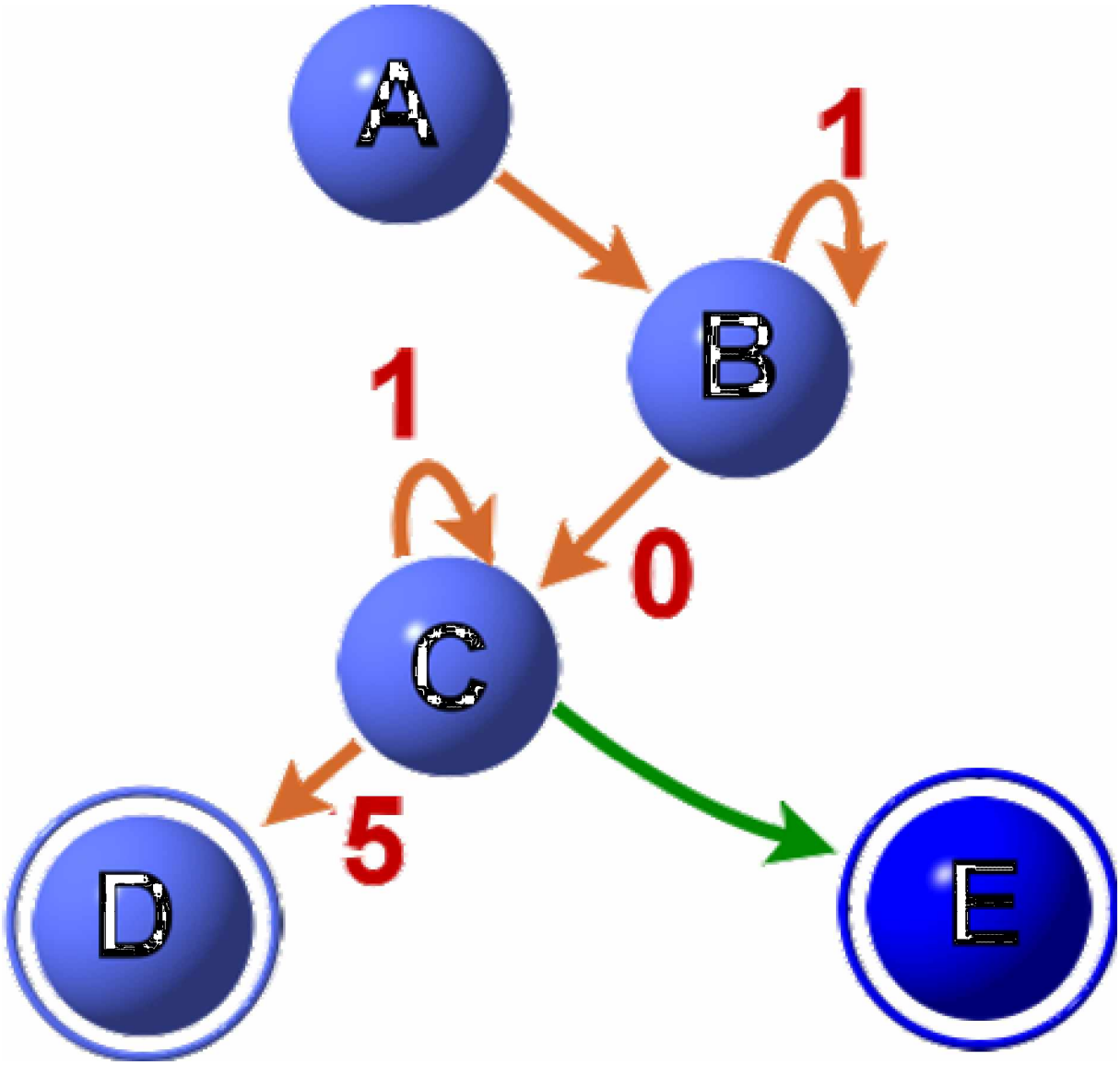
```
                int RANDOM;
                @Override
                protected void onTick() {
                        RANDOM=(int) (Math.random()*100);

                        if(RANDOM==5)
                        {
                                System.out.println("Bingo!");
                                myAgent.doDelete();
                        }
                }
        });
        addBehaviour(ParBehaviour);
    }
}
```

3.3.4. دمج أنواع مختلفة من الـ behaviour في behaviour واحد

يسمح JADE بدمج أكثر من نوع من الـ behaviour مع بعض و هذا لإنشاء Behaviours ذات تعقيد أعلى مما يعطي ذكاء اصطناعيا أعلى للعميل، مثلا يمكن دمج الـ FSMBehaviours و parallelBehaviour السابقين على النحو التالي:

تغيير الـ state C بحيث أنه يصبح ParallelBehaviour و عليه إن كان بمقدور الـ state C إرجاع القيمة 5 في أقل من ثانية فإننا نذهب إلى الـ state D و إلا فإننا نذهب إلى الـ state E



4. المصادر:

[Developing multi-agent systems with JADE edition wiley.](#)

JADE PROGRAMMER'S GUIDE

5.للاتصال:

يرجى منك عزيزي القارئ إذا وجدت أية أخطاء أو لديك أية ملاحظات أن ترسلها إلى djug على العنوان التالي:

djugprog@gmail.com