尊敬的杨老师：

　　您好，这是我这两周的学习情况： 这两周的主要工作都放在了软件开放上，还有就是对于上上周汇报的几种先进的数据流、复杂事件处理系统的跟进。软件开发情况：主要分两部分，一部份是 Drools 的学习，另一部分是结合复杂事件处理系统所做的测试内容

1.Drools 的学习（源码我放在 GitHub 上 https://github.com/Veronus/DroolsTest ）：

1.1 写一个 drl 的规则文件：

```
1  package droolscours
2
3  //list any import classes here.
4                (1)
5
6
7  //declare any global variables here
8
9             (2)
10
11
12 rule "Your First Rule" (3)
13
14     when
15         //conditions (4)
16     then
17         //actions (5)
18
19 end
```

（1）对于我们将利用每一个 Java 对象，我们需要导入类

（2）它可以定义全局变量

（3）唯一的规则名称

（4）它是在 RETE 算法的左侧被包括在的 drools

（5）在满足规则条件时触发规则操作，它可以使用纯 Java 代码。

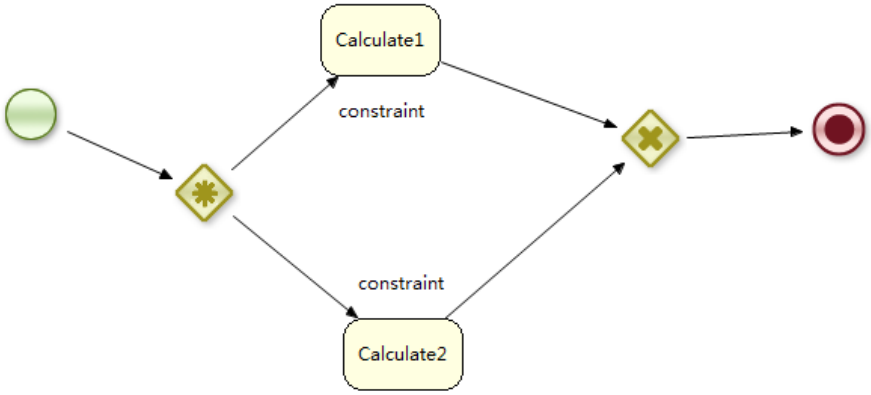
1.2 定义了一个名为 “ksession 规则 ”（即我们在测试中使用初始化会话）和在哪里可以找到规则文件在这里包 “esson1 ”会话。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule">
3
4      <kbase name="ruleDelay" packages ="eventRule">
5          <ksession name = "ksession-ruleDelay"/>
6      </kbase>
```

1.3 了解了一些基本的规则语言，包括 or 、and 、no 、exists 、forall 、from 、from collect、form accumulate 对应的规则 rule 文件和测试的 java 代码在下面的这两个 gitHub 的连接地址上 https://github.com/Veronus/DroolsTest/tree/master/src/test/rules
https://github.com/Veronus/DroolsTest/tree/master/src/test/java/droolscours

1.4 了解了规则流（如下图），将 bpmn 图（ps：这与小方在周六报告的 bpmn2 很好的联系了起来，所以在当时他报告的时候我就问他有没有将这个 bpmn 和规则结合起来，同时了解到他是通过网页画的 bpmn 图，而我这个是在 eclipse 中画的。）和规则进行绑定，能出很好的结果，直观形象的展示规则的流程（下面的学习案例是控制资金大与 1000 和小于 1000 的

两种不同的计算方式。）



2. 复杂事件处理系统所做的测试内容

测试主要内容：主要是比对多个事件发生时，传统 if 语句块与 drools 规则引擎的处理时间

样本大小（这里我就用的自己写的事件容器的所有事件）：17 个样本

测试方法：

公共代码部分：

项目整体按照 web 的标准框架进行构建，包涵 meta 包、dao 包、sevice 包、utils 包、webController 包，以及一些视图解析的文件，和一些前台展示的文件，包文件及主要 java 对象见：https://github.com/Veronus/webFrame/tree/master/src/main/java/com/mes

测试 If 语句的关键代码：

```
long startTime=System.currentTimeMillis();    //获取开始时间
ApplicationContext context = new ClassPathXmlApplicationContext("application-context-dao.xml");
EventDao eventDao = context.getBean("eventDao",EventDao.class);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//设置日期格式
List<Event> events = eventDao.getEvents();
for (Event event : events) {
    try {
        Date d1 = df.parse(event.getStarttime());
        Date d2 = df.parse(event.DATE);
        System.out.println(d1.getTime()  +"/n this is d2:  "+ d2.getTime());
        if (d1.getTime() <= d2.getTime()) {
            System.out.println(event.toString()+"This task is delay");
        }
        System.out.println(event.toString()+event.DATE);
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
((ConfigurableApplicationContext) context).close();
long endTime=System.currentTimeMillis(); //获取结束时间
System.out.println("程序运行时间: "+(endTime-startTime)+"ms");
```

测试说明：获取当前系统事件，发生事件的事件做比较（这里简单的理解为与任务的完成时间做比较），当系统时间大于任务的完成事件，则视为任务拖期，则向控制台输出该任务的基本信息。

测试 Drools 规则引擎的具体代码：

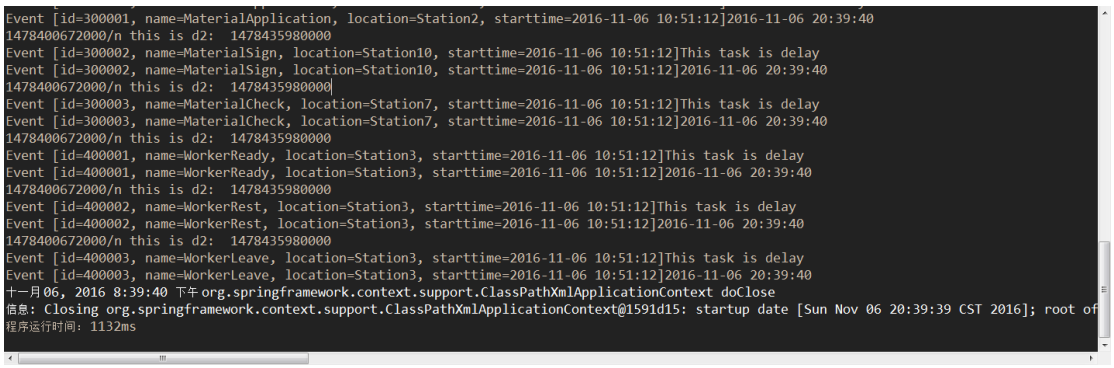
```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    try {
        // load up the knowledge base
        KieServices ks = KieServices.Factory.get();
        KieContainer kContainer = ks.getKieClasspathContainer();
        KieSession kSession = kContainer.newKieSession("ksession-ruleDelay");
        OutputDisplay outputDisplay = new OutputDisplay();
        // go !
        long startTime=System.currentTimeMillis();    //获取开始时间
        ApplicationContext context = new ClassPathXmlApplicationContext("application-context-dao.xml");
        EventDao eventDao = context.getBean("eventDao",EventDao.class);
        List<Event> events = eventDao.getEvents();
        for (Event event : events) {
            System.out.println(event.toString()+event.DATE);
            kSession.insert(event);
            kSession.fireAllRules();
        }
        ((ConfigurableApplicationContext) context).close();
        long endTime=System.currentTimeMillis(); //获取结束时间
        System.out.println("程序运行时间： "+(endTime-startTime)+"ms");

    } catch (Throwable t) {
        t.printStackTrace();
    }

}
```

Rule 规则：

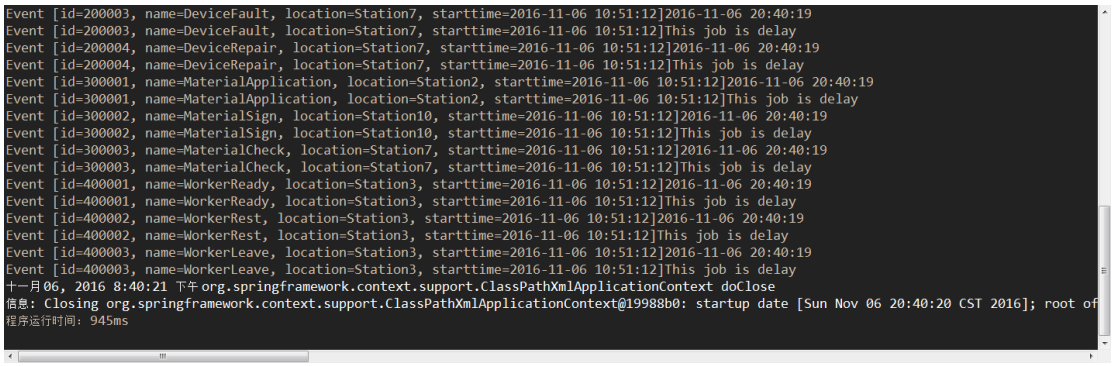
```
package com.mes.meta

//list any import classes here.
import com.mes.meta.Event;
import com.mes.utils.OutputDisplay;


//declare any global variables here
global OutputDisplay showResults;



rule "Delay"

    when
        e:Event( starttime <= Event.DATE )
    then
        //actions
         System.out.println( e.toString() + "This job is delay" );
end
```

规则说明：当系统时间大于任务的完成事件，则视为任务拖期，则向控制台输出该任务的基本信息。

测试的结果：



用 if 语句写的测试代码运行时间 1132ms



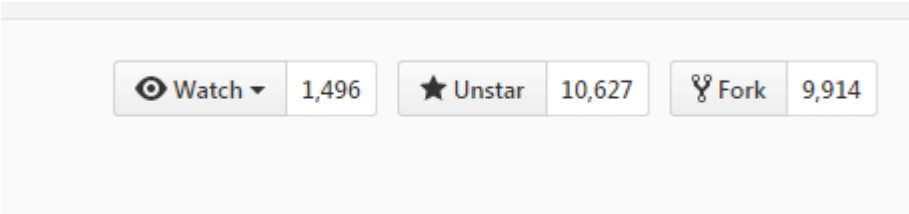
用 Drools 规则引擎测试出来的时间是 945ms

结论：通过以上测试说明 drools 的处理时间大约比 if 语句快 1/5~1/6，在处理大量事件的时候，能明显的加快处理效率。

3.热点问题的关注 Spark Streaming

开源地址：https://github.com/apache/spark

简介：Spark 是为大数据快速和常规集群计算系统。它提供了在 Scala 中，使用 Java，Python，和 R，以及支持对数据进行分析通用计算图形优化引擎的高级 API。它还支持一组丰富的更高级别的工具，包括 SQL 星火为 SQL 和 DataFrames，MLlib 机器学习，GraphX 进行图形处理，和 **Spark 流进行流处理。（我们所关注的主要是事件流的处理）**



从上面的关注度我们可以看出，本开源项目目前十分的火，有 1496 人时刻关注这此项目的代码更新情况，共有 10627 人关注此项目，有 9914 个项目分支。同时国内的华为也结合此项目做了 SparkMD，如下图。

streamDM: Data Mining for Spark Streaming

streamDM is a new open source software for mining big data streams using Spark Streaming, developed at Huawei Noah's Ark Lab. streamDM is licensed under Apache Software License v2.0.

HUAWEI

华为结合 SparkSreaming 做的 StreamMD

总结：本周基本掌握了 drools 的配置和编写，还有 bpmn 结合 drools 规则的思路，同时基本完成了从数据库里取数据，放入规则池中进行判断，到推送通知这一过程。

下周工作安排：由于开题临近，下周主要先完成开题报告和文献综述。

此致

敬礼

文波

2016/11/06