**Nlu引擎规则说明文档**

# 背景

本项目的目的是希望能够通过配置规则，即可满足基础的语音指令识别，来实现人机交互。

# 配置说明

配置文件存放在com.ainemo.nlu.config中，在android工程中是存放在assets/nlp中。

配置文件按前缀来分类，分为rules\_和domain\_两类

* rules\_ 用于配置一些通常的规则，filter, validator和result\_handler
* domain\_ 则用于配置与某一个垂类相关的规则

目前配置文件的元素主要包括domain, rule(s), filter(s), validator(s), result\_handler(s)五大类。

## <filters>

<filters>用于配置过滤器规则，用于对规则引擎的输入进行预处理。没有attribute属性，只有一个<filter>子标签

### 配置说明

#### <filter>

filter的相关属性如下定义：

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | value | default | note |
| name | String | Not null | filter必须要显式指定一个name的属性。 |
| class | Class path | false | class用于定义当前name所对应的filter的类路径，如com.ainemo.nlu.filter.NluExampleFilter，filter必须要实现NluFilter，NluFilter定义见下一条 |
| global | True/false | False | 为true的时候，说明这是一个全局的filter，会在全局域内对输入进行预处理。而为false的时候，只会在<rules>域内对输入进行预处理。  如：定义两个rule（rule1和rule2），其中rule1配置了filter1，而 rule2没有配置。则此时rule1的输入将会先经过filter1的预处理，而rule2不会  <rule name=”rule1”>  <rule></rule>  <filter name=”filter1”>  </rule>  <rule name=”rule2”>  <rule></rule>  </rule> |

### NluFilter

**package** com.ainemo.nlu.rule;

**import** com.ainemo.nlu.parser.NluContext;

**public** **interface** NluFilter {

**public** String filter(String input, NluContext context);

}

## <validators>

### 配置说明

#### <validator>

validators用于声明校验器，由子标签<validator>声明，具体属性如下：

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | value | default | note |
| name | String | Not null | validator必须要显式指定一个name的属性。 |
| class | Class path | false | class用于定义当前name所对应的validator的类路径，如com.ainemo.nlu.validator.NluExampleValidator，validator必须要实现NluValidator，NluValidator定义见下一条 |
| rule | String | 空 | 当rule没有定义的时候，校验器直接使用子类的实现来校验，而当rule有指定的时候，此时的name必须要指向一个已经定义好的<rules>的规则的名字。框架会把validator的rules初始化好。以便在子类中使用这一规则来做校验。  框架已经预定义好一个NluRulesValidator，只需要指定相应的rule就可以了。NluRulesValidator在rules匹配成功的时候返回true，否则返回false，具体定义表格下方 |

<rules name="negative\_rule">

<rule>不</rule>

<rule>没有</rule>

<rule>别</rule>

<rule>否</rule>

<rule>非</rule>

</rules>

<validators>

<validator name="negativeValidator" class="com.ainemo.nlu.rule.validator.NluRulesValidator" rule="negative\_rule"/>

</validators>

<domain name="phone">

<rules action="open" match="false">

<rule><![CDATA[ (?<who>.\*)打.\*电话 ]]></rule>

<rule><![CDATA[ (?<who>.\*)紧急呼叫 ]]></rule>

<validator group="who" name="negativeValidator" return="false"/>

</rules>

</domain>

定义一个negativeValidator，用于校验rules匹配的结果group组（这里是who），有没有否定词，有的话返回true，没有返回false。

因为这domain的rules定义的group（即<who>）里是要匹配不能有否定词，所以里面的validator配置的return=”false”。

更多domain的定义参见<domain>一栏

### NluValidator

**package** com.ainemo.nlu.rule;

**import** com.ainemo.nlu.parser.NluContext;

**public** **interface** NluValidator {

**public** **boolean** validate(String value, NluContext context);

/\*\*

\* 如果<validator>指定了rule的标签，setRules接口就会被调用，否则忽略

\*/

**public** **void** setRules(NluRules rules);

}

## <result\_handlers>

result\_handlers用于声明相应的NluResultHandler的子类对象，NluResultHandler用于对<rules>匹配的结果组进行进一步的处理。

result\_handlers只有一个<result\_handler>子元素

### 配置说明

#### <result\_handler>

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | value | default | note |
| name | String | Not null | 必须要显式指定一个name的属性。 |
| class | Class path | Not null | class用于定义当前name所对应的result\_handler的类路径，如com.ainemo.nlu.result.handler.NluExampleHandler，class对应的类必须要实现NluResultHandler，NluResultHandler定义见下一条 |

### NluResultHandler

result\_handlers用于声明一系列handler

**package** com.ainemo.nlu.rule;

**import** com.ainemo.nlu.parser.NluContext;

**public** **interface** NluResultHandler {

**public** NluRuleMatchResult handle(NluRuleMatchResult result, String group, NluContext context);

}

## <rules>

<rules>主要用来配置匹配规则，由attribute和property两部分配置组成。

rules分成两类，一个是全局的rules，一个是在domain里面的 rules，domain里面的rules可以引用全局的rules，如

<rules name=”example\_rule”>

<rule>好的</rule>

</rules>

<domain name=”phone”>

<rules action=”input”>

<rule ref=”example\_rule”/>

</rules>

</domain>

### attribute

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | value | default | note |
| name | String |  | 全局的rules必须要显式指定一个name的属性。在domain里面嵌套的name由系统定义，通常为domainName.actionName |
| action | String | 空 | 全局的rules无需指定action属性，而在domain里面的rules，则需要指定一个action属性。 |
| match | true/false | false | 为true时，完全匹配整个字符串，false的时候只需匹配子串 |
| ~~confirm~~ | ~~True/false~~ | ~~False~~ | ~~用于标识是否是一个回复，实质上并不需要用到也可以~~ |
| return\_group | String | 空 | 指出在正则表达式中用于返回的组的名字，支持返回多个字段 |
| return\_key | String | 空 | 指出本规则返回上层的key，如果没有配置，则会使用return\_group做为返回上层的key，如果也没有配置return\_group，则不会返回任何key和value，只会返回是否匹配的一个布尔值（比较少用，可能会有bug） |
| replacement | String | 空 | replacement为空的时候，正则表达式用于匹配规则，当replacement不为空，即有配置这一属性（包括replacement=””）时，正则表达式的规则用于替换字符串，这时主要用于对规则校验之前的字符串进行预处理 |

### property

|  |  |  |  |
| --- | --- | --- | --- |
| Property Name | value | default | note |
| rule | String | NOT null | rule定义一个正则表达式的规则，可是一个纯字符串，也可以是一个正则表达式，使用正则表达式的时候，尽量采用CDATA标签 |
| filter | String | NOT null | 声明一个过滤器，对经过当前<rules>的所有规则的输入进行处理，filter标签的attribute如下定义：  name： 用于指定一个filter的name，声明是调用哪一个filter，具体的filter在<filters>标签会有预定义，引用一个不存的filter name，会抛出一个异常。  cascade: 是一个boolean的值，默认为false，为 true时，说明当前filter也会定义在<rule ref=”ref\_rule\_name/>标签中引用的ref\_rule\_name的规则中去。 |
| validator | String | NOT null | 声明一个校验器，对正则表达式处理完的结果，进行校验。相关的attribute如下定义：  name: 用于指定一个validator的name，声明是调用哪一个validator，具体的validator在<validators>中定义。引用一个不存在的validator，无法抛出一个异常。  return: 声明validator的返回值，即只有当validator返回return中声明的值的时候，检验通过。目前的类型是true/false  group: 正则表达式会把匹配的字符串归组成一个一个的组，而group则声明了需要校验哪一个组的值，当group配置的值为\*的时候，说明校验整个匹配的字符子串。  cascade: 是一个boolean的值，默认为false，为 true时，说明当前validator也会定义在<rule ref=”ref\_rule\_name/>标签中引用的ref\_rule\_name的规则中去。 |
| result\_handler | String | Not null | 声明一个结果处理器，结果处理器用于处理正则表达式的结果，即如果匹配的结果需要做一些特殊处理的时候，配置一个result\_handler即可  相关的result\_handler的attribute如下定义：  name: 用于指定一个result\_handler的name，声明是调用哪一个result handler，具体的handler在<result\_handlers>中定义。引用一个不存在的result\_handler，无法抛出一个异常。  group: 正则表达式会把匹配的字符串归组成一个一个的组，而group则声明了需要对哪一个组的值进行特珠处理后再返回，当group配置的值为\*的时候，说明校验整个匹配的字符子串。  cascade: 是一个boolean的值，默认为false，为 true时，说明当前result\_handler定义的handler也会用在在<rule ref=”ref\_rule\_name/>标签中引用的ref\_rule\_name的规则中去。 |

## <domain>

domain主要是由name attribute和rules elements组成

### Attribute

name属性用于指定domain所属的垂类

priority用于指定domain的优先级

### property

rules必须指定一个action属性，标记在domain name这个垂类里面的动作action。在domain里面定义的<rules>还有一个priority的属性，用于定义优先级。

具体的rules定义参见前面