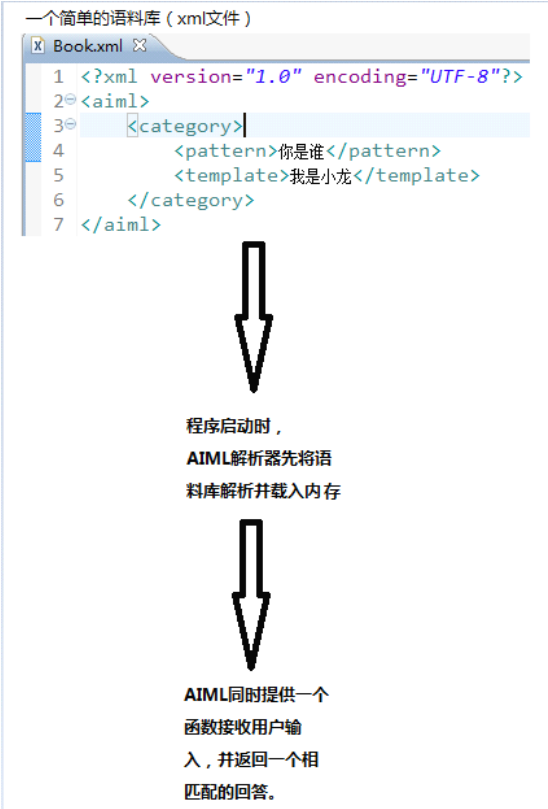
聊天机器人文档

一．聊天机器人基本原理



语料库中的 pattern 是模式的意思，可理解为问题，而相应的 template 可理解为回答（而这一对问答被包裹在了 category 标签里面）。假如你的语料库像上面的 xml 文件这样简单，那么当你输入“你是谁”，机器人就会在内存中去一个一个的匹配 pattern,最后匹配到了，就会回答“我是小龙”，而你输入其他任何语句，机器人就无从匹配了，程序会出现匹配不到的错误。那么怎样避免程序出错呢？我们修改语料库如下：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<aiml>
    <category>
        <pattern>你是谁</pattern>
        <template>我是小龙</template>
    </category>

    <category>
        <pattern>*</pattern>
        <template>
        对不起，主人还没有教我怎么回答这个问题呢？
        </template>
    </category>
</aiml>
```

上图中的*，是 AIML 中的通配符，它匹配任何你输入的语句。当你输入的语句成功匹配，那么返回相应的 template 后，就不会再去匹配其他的 category 了。假如程序没有任何相匹配的，那么*总是可以匹配你的输入，机器人会输出“对不起，主人还没有教我怎么回答这个问题呢？”

当然 AIML 解析器所支持的 xml 标签种类远不止这些，上述是最基本的。AIML 所支持的标签种类目测有 20 种。

二．为什么 AIML 解析器不支持中文

国外的一款做的很好的聊天机器人（通过了图灵测试），她叫“Alice”(你可以用英文和 Alice 聊天

http://www.pandorabots.com/pandora/talk?botid=f5d922d97e345aa1),它内部有很庞大的语料库，几乎所有用户可能说的话，语料库中都有，而且它用的就是 AIML 解析器。然而 AIML 程序中有一些地方会用正则表达式将除了英文字母和数字外的其他字符全部用空格替代，这就是 Aiml 解析器不支持中文的重要原因，这种正则表达式出现在 bitoflife.chatterbean.text.Transformations 类中，比如下面这个函数：

```java
public void normalization(Sentence sentence) {
    String input = breakWords(sentence.getOriginal());
    input = ' ' + input + ' ';
    input = chineseTranslate(input);
    input = input.replaceAll("\\s{2,}", " ");
    sentence.setOriginal(input);

    Mapper mapper = new Mapper(input);
    input = substitute(input, mapper);

    input = fit(input, mapper);
    sentence.setMappings(mapper.toArray());

    sentence.setSplittedOfSentence(input);
}
```

```
/**
 * Turns the entry to UPPERCASE, takes sequences of non-alphanumeric
 * characters out of it (replacing them with a single whitespace) and sees
 * that the entry is trimmed off leading and trailing whitespaces.
 */
private String fit(String input, Mapper mapper) {

    input = input.toUpperCase();

    Matcher matcher = fitting.matcher(input);
    StringBuffer buffer = new StringBuffer();
    while (!matcher.hitEnd() && matcher.find()) {
        mapper.prepare(input, matcher.group(), " ");
        mapper.update(matcher.start());
        matcher.appendReplacement(buffer, " ");
    }

    matcher.appendTail(buffer);
    return buffer.toString();
}
```

其中 fitting 为：

```
private final Pattern fitting = Pattern.compile("[^A-Z0-9\u4E00-\u9FA5]+"); // 修改正则表达式使其支持中文。
```

上面的 normalization 函数是用来对用户输入做规范化处理的，它做了一下工作：

1.在原始输入的内容两头加空格。

2.把句子中间的任何 2 个以上连续的空白字符都替换成一个空格。

3.并对 input 进行字符过滤

修改完正则表达式之后，算是成功了一半，那还需要做怎么处理呢？

我们知道 AIML 当初仅仅是针对 English 语言开发的，而 English 单词之间都是有空格的，所以在前期载入语料库阶段，解析器默认 xml 语料库中的词之间都有空格，然后通过空格将句子分成一个个单词，最后在内存构建一个匹配树，而且在处理用户输入的句子时也是将其进行了规范化处理，如下代码：

```
/**
 * 该方法的作用是规范化request。
 *
 * 规范化过程：1.在原始输入的内容两头加空格。 2.把句子中间的任何2个以上连续的空白字符都替换成一个空格。
 * 3.将输入内容按照约定好的句子结束符断句，放到Sentence[]中。 4.再将Sentence[i]进行规范化处理
 *
 * @param request
 */
public void normalization(Request request) {
    String original = chineseTranslate(request.getOriginal());
    original = ' ' + original + ' ';
    original = original.replaceAll("\\s{2,}", " ");
    String input[] = splitter.split(original);

    Sentence[] sentences = new Sentence[input.length];
    for (int i = 0, n = input.length; i < n; i++) {
        sentences[i] = new Sentence(input[i]);
        normalization(sentences[i]);
    }

    request.setOriginal(original);
    request.setSentences(sentences);
}
```

而为了让它支持中文，一个比较直观的方法是在对用户输入做规范化处理的时候，我们将输入的中文句子中加入空格，比如上述代码的 chineseTranslate 函数：

```java
/**
 * 该方法将中文字之间加上空格。
 *
 * @param input
 * @return
 */
public static String chineseTranslate(String input) {
    StringBuffer newStr = new StringBuffer("");
    java.util.regex.Pattern p = java.util.regex.Pattern
            .compile("[\u4E00-\u9FA5]");
    char[] chars = new char[1];
    Matcher m;
    for (int i = 0; i < input.length(); i++) {
        chars[0] = input.charAt(i);
        m = p.matcher(new String(chars));
        if (m.matches())
            newStr.append(" ").append(input.charAt(i)).append(" ");
        else
            newStr.append(input.charAt(i));
    }
    // java.lang.System.out.println("#" + newStr.toString());
    // strTemp = newStr.toString().replaceAll("\\s{2,}", " ");//
    // 把2个连续的空格替换成一个空格。
    // strTemp = strTemp.replaceAll("^\\s*|\\s*$", ""); // 把头和尾的空格去掉。
    return newStr.toString();
}
```

同理，在载入语料库的时候，同样需要写一个类似的函数，功能就是将语料库中的字之间加上空格。

三．关于标签的支持与使用

Aiml 标签使用的官方文档

http://www.alicebot.org/TR/2005/WD-aiml/

http://www.pandorabots.com/botmaster/en/tutorial?ch=4

根据上面的连接我们可以初步了解各种标签的作用和用法。

然而，就我现在的中文聊天机器人，有些标签还不能正常使用，这是需要改进的地方。

下面是一些我尝试过的一些标签用法：

最基本：

```
<category>
    <pattern>你好</pattern>
    <template>呵呵</template>
</category>
```

每一个问题和回答都被包裹在<category>标签中

随机返回功能：

```
<category>
    <pattern>你好</pattern>  (或者在后面加一个*)
    <template>
        <random>
            <li>你好呀！</li>
            <li>嘿嘿</li>
            <li>我很好，你呢？</li>
```

```
        </random>
    </template>
</category>
```

是 library 的意思，不是 list

当你输入“你好”的时候，机器人会从 random 里面随机取出一句回答你。不过默认的都是先取第一句回答。

输入重定向功能（<srai>）：

```
<category>
    <pattern>你好</pattern>
    <template>
        <random>
            <li>你好呀！ </li>
            <li>嘿嘿</li>
            <li>我很好，你呢？ </li>
        </random>
    </template>
</category>

<category>
        <pattern>HELLO</pattern>
        <template><srai>你好</srai></template>
</category>
```

输入“hello”的时候，会匹配到第二个 category，而 srai 标签的功能是，将“你好”当成用户的输入，并重新到语料库里去匹配，最后就匹配到了第一个 category。换句话说，用户输入“hello”和输入“你好”的效果是一样的。但是在使用 srai 标签的时候有可能会形成死循环，所以请慎重。

另外需要注意的是，如果你想要在语料库里面写英文的语料库，那么英文单词都要是大写的，而用户输入的英文可以不用大写。如果你觉得用大写很不爽，那么你可以去修改源代码。

---

*，<think>,<set>,<get>,<star>的使用:

```
<category>
    <pattern>我叫*</pattern>
    <template>
        <think>
            <set name="myname"><star/></set>
        </think>
        hello, <get name="myname"/>.
    </template>
</category>
```

测试结果为：

you say>

我叫小龙

Alice>hello, 小龙.

标签解释：

set 和 get 里面的 myname 相当与参数名，首先在 set 标签中给 myname 赋值，然后用 get 标签得到相应参数的值，如果 myname 之前没有被赋值，那么就是空字符串。<star/>指的是 pattern 标签中*所匹配的内容。它还能指定 index，举个例子：

```
<pattern>我叫*呵呵*</pattern>
    <template>
        <think>
            <set name="myname"><star index=”2”/></set>
        </think>
        hello, <get name="myname"/>.
</template>
```

那么这时 star 标签就会被 pattern 中第2个*号所匹配的内容替代。而<star/>其实相当与<star index=”1”/>

<think>标签可以理解为机器人在思考，它只会在“脑子”里默默的记住一些事情，或者完成一些不会被用户看到的工作。

Condition 标签使用：

```
<category>
    <pattern>我叫*</pattern>
    <template>
        <think>
        <set name="myname"><star/></set>
        </think>hello, <get name="myname"/>.
    </template>
</category>
<category>
    <pattern>你好*</pattern>
    <template>
        你好啊！
        <condition name="myname" value="jack">怎么又是你？</condition>
    </template>
</category>
```

测试结果：

you say>

我叫 jack

Alice>hello, jack.

you say>

你好啊

Alice>你好啊！ 怎么又是你？

you say>

我叫 job

Alice>hello, job.

you say>

你好啊

Alice>你好啊！

标签解释：

<condition>标签中的 myname 是在 set 中被赋值的。然后在匹配到“你好*”后，就要判断是不是“jack”

---

input 标签的用法：

```
<category>
    <pattern>我叫*</pattern>
    <template>
        <think><set name="name"><star/></set></think>
        你好啊，<get name="name"/>
    </template>
</category>
<category>
    <pattern>嘿嘿</pattern>
    <template>
        你刚才说：“<input index="2"/>”？
    </template>
</category>
```

测试结果：

you say>

我叫 jack

Alice>你好啊，jack

you say>

嘿嘿

Alice>你刚才说：“我 叫 jack”？

标签解释：

<input>标签指的是用户之前的输入，加上一个 index，那就是说，用户倒数第几句输入，注意是“倒数”！index=”1”,就是用户倒数第一句输入的内容，以此类推，当然 index 是会出现越界错误的。

---

date 标签的使用：

```
<category>
    <pattern>现在什么时间*</pattern>
    <template><date format="h:mm a"/>.</template>
</category>
```

测试结果：

you say>

现在什么时间啊

Alice>It is 9:49 下午.

Date 标签将获得当前的系统时间

<that>元素表示先前机器人说的话，例如：

```
<category>
    <pattern>聊什么好呢*</pattern>
    <template>一起聊聊电影好吗？</template>
</category>
<category>
    <pattern>好</pattern>
    <that>一起聊聊电影好吗？</that>
    <template>那你喜欢什么电影呢？</template>
</category>
<category>
    <pattern>不好</pattern>
    <that>一起聊聊电影好吗？</that>
    <template>那我也不知道聊什么了~</template>
</category>
```

测试结果：

you say>

聊什么好呢？

Alice>一起聊聊电影好吗？

you say>

好

Alice>那你喜欢什么电影呢？

you say>

聊什么好呢

Alice>一起聊聊电影好吗？

you say>

不好

Alice>那我也不知道聊什么了~

这个标签还能取前面任意机器人说的话，不过不太熟...没有试验过

如果要取前面的前面机器人的话，可以用：<that index=”nx,ny”>,例如：<that index=”2,1”

表示取机器人倒数第2句的话，<that index=”2,1”>也等于<justbeforethat/>

<thatstar>标签：

```
<category>
    <pattern>你好</pattern>
    <template>计算机的型号是什么</template>
</category>
<category>
    <pattern>*</pattern>
    <that>*的型号是什么</that>
    <template>
        <star/>
        这个型号是 <thatstar/> 里面
        <random>
            <li>很好的商品</li>
            <li>很流行的商品</li>
            <li>很华丽的商品</li>
        </random>。
    </template>
</category>
```

--》》这里的 star 标签匹配的是 pattern 中的*，但是奇怪的，如果把 index 改成2以后，却也不会出错。

测试结果：

you say>

你好

Alice>计算机的型号是什么

you say>

d

Alice>d 这个型号是 里面 很好的商品。

thatstar 是匹配 pattern-side that 标签里面的*号的，但是这里没匹配到。

我想这里也还需要修改源代码。

---

set 标签也有问题。

```
<category>
  <pattern>他做到了</pattern>
  <template>谁 ？ </template>
</category>
<category>
    <pattern>*</pattern>
    <that>谁 *</that>
    <template>
    Oh, why do you think <set name="他"><star/></set> did that? I wouldn't expect that
    kind of behavior from <get name="他"/>.
    </template>
</category>
<category>
    <pattern>*</pattern>
    <template>啊哦~</template>
</category>
```

测试结果：

you say>

他做到了

Alice>谁 ？

you say>

小龙

Alice>Oh, why do you think did that? I wouldn't expect that kind of behavior from .

假如这样写：<set name="he"><star/>

那么测试结果为：

you say>

他做到了

Alice>谁 ？

you say>

jack

Alice>Oh, why do you think jack did that? I wouldn't expect that kind of behavior from jack.

也就是说这个标签不支持中文。还是需要修改源代码。

---

template-side input 有问题：

```
<aiml:input
   index = (single-integer-index | comma-separated-integer-pair) />

<category>
<pattern>HELLO</pattern>
<template>吃饭了吗？</template>
</category>

<category>
<pattern>吃了</pattern>
<template>我也吃了</template>
</category>

<category>
<pattern>你好</pattern>
<template>计算机 的 型 号 是 什 么</template>
</category>

<category>
<pattern>*</pattern>
<that>* 的 型 号 是 什 么</that>
<template>
<input index="4,1"></input>    《《----
</template>
</category>
```

测试结果：

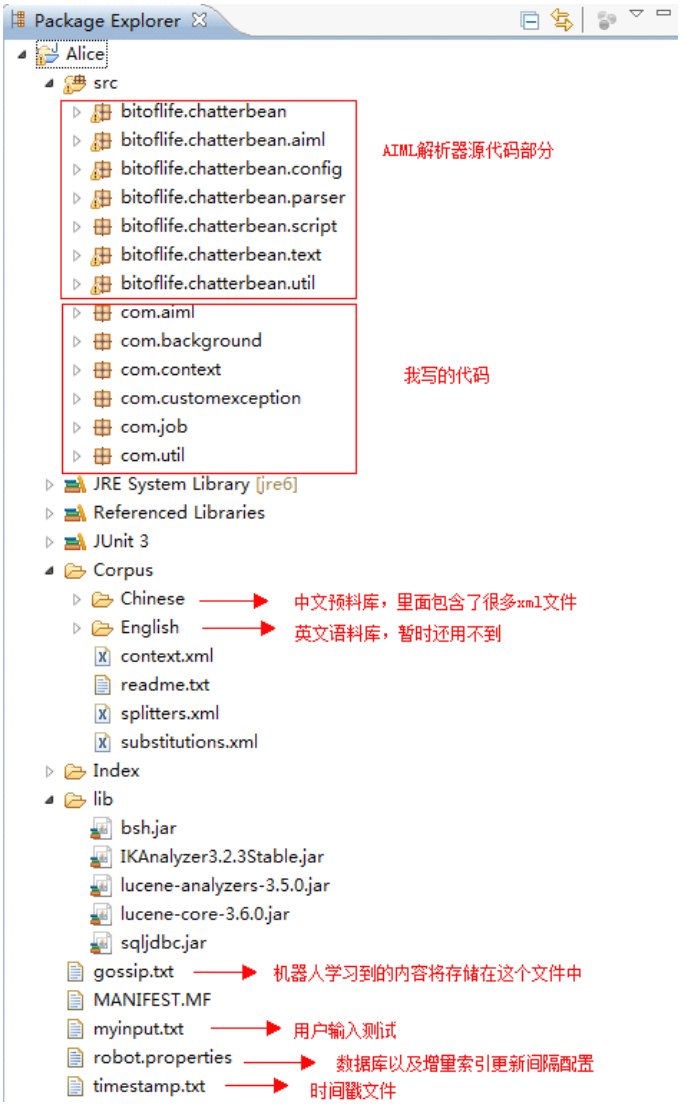
you say>
hello
Alice>吃饭了吗?
you say>
吃了
Alice>我也吃了
you say>
你好
Alice>计算机 的 型 号 是 什 么
you say>
345
Alice>hello

Input 标签中的 index 貌似当第一个参数是几,就返回倒数第几个用户的说的话,而第二个参数好像只能是1,其他的就会出现数组越界的错误。不知道为什么?

上面描述的标签部分还有问题，需要改进。

另外我想说的是，在写 xml 语料库的时候，一定要写一点，马上重启程序测试一次，看新加的预料是否工作正常，否则你写了一堆的预料后在去测试如果出错的话，就很难跟踪到错误的地方了。

四．项目结构总览



五．将数据库集成到聊天机器人中

一）为什么需要使用数据库

Xml 文件是 AIML 所支持的预料载体，而且凭借 AIML 提供的各种丰富的标签，作者可以设计并编写出很人性化的语料库。显然，通过这种方式写语料库的特点是灵活性很好，能很容易写出“唠嗑”类型的聊天内容。然而，当时对这个项目的定位是客服机器人，也就是说，语料库还应该包含具有业务针对的预料，这部分预料将随着业务的不同而不同。于是我想把这部分预料存储在数据库中形成动态的语料库（我把 xml 文件中的预料称为静态预料，也就是说这部分预料完善之后就不去频繁的修改），这样做的好处有一下几点：

1.客服不用去学习怎样写 xml 预料，降低门槛。

2.可以避免 xml 中预料越来越凌乱，到最后难以管理。

3.以后可以针对数据库在开发一个语料库管理系统，方便客服管理有业务针对的这部分预料。

二）数据库的表是怎样设计的

id	createTime	isDeleted	lastModifyTime	question	replay	label	copyfield
25	2012/9/17 11:51:40	False	2012/9/16 16:11:31	什么是淘宝hehe大学	逃吧大学是这样一所大学	听课 上课 学习	什么是淘宝hehe大学 逃吧大学是这样一所大学 听课 上课 学习
26	2012/9/17 20:12:45	False	2012/9/17 20:12:45	淘宝是什么	淘宝是苹果	支付宝 淘宝	淘宝是什么 淘宝是苹果 支付宝 淘宝
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

字段解释：

Id:预料的编号，自动生成（identify）

Createtime:该条数据创建时间，该条数据产生时自动生成（触发器）

Lastmodifytime:最近更新时间，该条数据的可填写字段被更新时自动修改（触发器）

Question:具体问题，自己填写。

Replay：具体回答，自己填写。

Label:标签字段（词语之间用空格隔开），里面填写的词语是要能体现 question 字段主题的，可以理解为一种补充说明，例：question:书是什么？那么书应该是被讨论的主题，lebel 就可以填写和书意义相近的词，比如课本，教科书，教材，book,有了这个字段可以从某种程度上增强匹配效果。

Copyfield:拷贝字段，这个字段会在你填写完（或者更新）question,replay,label 这 3 个字段后自动生成（触发器），其内容为上述 3 个字段的合体，期间用空格隔开。这个字段是要被索引的重点字段。

在以上的描述中，也许你会对某些字段存在的必要性产生疑惑，没有关系，在下面的叙述中也许能解决你的问题。

三）数据库里面的预料怎样使用到机器人当中

这就是基于 lucene 的处理。当程序启动的时候，程序会在载入 xml 语料库后,lucene 就开始对数据库进行全量索引（这其实也是一种载入语料库行为），并在项目的根目录下产生相应的索引文件 index 以及时间戳文件（timesTamp.txt：该文件记录了当前索引行为发生的时间。将在增量索引时用到）。索引文件会在后面响应用户输入的时候用到。

下面是索引操作的代码（在 `com.job` 包）：

```
/**
 * 全量索引
 */
public void fullIndexSetup() {
    IndexOperation(fullIndexSql, OpenMode.CREATE);
}
```

```
    /**
     * openMode会有2中模式： 1.全量索引模式 2.增量索引模式
     * @param sql
     * @param openMode
     */
    public void IndexOperation(String sql, OpenMode openMode) {
        indexResource.dbObject.linkDataBase(); // 连接数据库
        IndexWriter writer = null;
        ResultSet rs = indexResource.dbObject.getResultSet(sql);// 根据sql得到结果集
        if (!Util.INDEXFILE.exists()) {// 判断index索引文件是否存在
            Util.INDEXFILE.mkdir();
        }

        try {
            // 根据模式得到相应的写索引的对象
            writer = indexResource.getIndexWriterByMode(openMode);
            while (rs.next()) {
                Document doc = new Document();
                // 得到数据库中copyfield字段名称
                String columnName = indexResource.dbObject.getColumnName(7);
                // 得到copyfield字段的值
                String columnValue = indexResource.dbObject.getColumnValue(7);
                Field.Index p = Field.Index.ANALYZED;
                // 在document中加入该field
                doc.add(new Field(columnName, columnValue, Field.Store.YES, p));

                // 将该document（也就是数据库中一个数据项）写入索引中
                indexDoc(writer, doc, openMode);
            }
            //记录当前索引行为的时间
            Util.setTimestamp();
        } catch (SQLException e) {
            throw new AppException(e);
        } finally {
            if (indexResource.dbObject != null) {
                indexResource.dbObject.close();
            }
            if (writer != null) {
                try {
                    writer.close();
                } catch (CorruptIndexException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }

    public void indexDoc(IndexWriter writer, Document doc, OpenMode openMode) {
        try {
            // 更新索引，他的工作原理就是先将索引文件中该id的数据删除掉，然后在将数据库中该id的数据索引一遍
            if (openMode == OpenMode.APPEND) {
                writer.updateDocument(new Term("id", doc.get("id")), doc);

            } else if (openMode == OpenMode.CREATE) {// 创建索引
                writer.addDocument(doc);
            }
        } catch (CorruptIndexException e) {
            throw new AppException(e);
        } catch (IOException e) {
            throw new AppException(e);
        }
    }
}
```

其中 sql 语句是这样的：

```
  public void buildQuery() {
      // 全量索引的sql语句
      fullIndexSql = "select * from ResponseTable where isDeleted=0 and datalength(replay)!=0";
      // 增量索引的sql语句
      deltaIndexSql = "select * from ResponseTable where lastModifyTime >'"
              + Util.getTimestamp()
              + "' and isDeleted=0 and datalength(replay)!=0";
  }
```

四）在机器人处于运行状态时修改了数据库的预料，怎样做到与客户端的同步

这里将用到时间戳的概念。首先当机器人程序运行的时候，里面的一个 timmer 任务也

会同时运行，这个任务做的工作是定期（比如每隔 10 秒），进行一次增量索引---lucene 中的概念。增量索引所针对的数据等同与这样一条 sql 语句所返回的数据，该 sql 语句满足的逻辑是：查找出数据库中所有 Lastmodifytime 字段值大于 timesTamp.txt 中记录的时间。那么这样，每次客服对数据库做的预料修改，都会在隔一段时间后同步到客户端。

```
/**
 * 增量索引
 */
public void deltaIndexSetup() {
    Timer timerIndexMake = new Timer();
    timerIndexMake.schedule(
            new makeIndexTask(),
            Integer.parseInt(RobotProperty.getConfiguration().getProperty(
                    "delay")),
            Integer.parseInt(RobotProperty.getConfiguration().getProperty(
                    "period")));
}

public class makeIndexTask extends TimerTask {

    @Override
    public void run() {
        IndexOperation(deltaIndexSql, OpenMode.APPEND);
    }
}
```

六．机器人怎样响应用户的输入

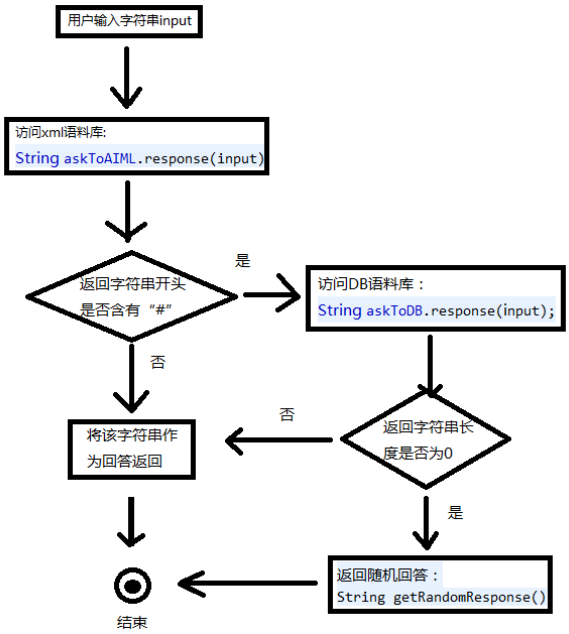
在没有引入数据库前，只要调用 Aiml 提供的聊天接口就能得到一个字符串返回了，但是现在加入了数据库，那么我的处理的思路是这样的：

我在 xml 语料库里面的*通配符所对应的 templete 做了标记，如下图：

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml>
    <category>
        <pattern>你是谁</pattern>
        <template>我是小龙</template>
    </category>

    <category>
        <pattern>*</pattern>
        <template>
        #对不起，主人还没有教我怎么回答这个问题呢?
        </template>
    </category>
</aiml>
```

注意到上面的红点了吗？我在这个回答最前面加上了“#”的标记。逻辑如下：



代码如下：

```
/**
 * 联合AIML和数据库这2个知识库。
 */
public String response(String input) {
    String responseFromAIML = askToAIML.response(input);
    responseFromAIML = responseFromAIML.replace(" ", "");
    return translate(input, responseFromAIML);
}

private String translate(String originInput, String aimlReplay) {
    String asDBInput = "";
    if (-1 != aimlReplay.indexOf(NULLSIGN)) {
        asDBInput = originInput;
    } else if (-1 != aimlReplay.indexOf(USEFULSIGN)) {
        asDBInput = aimlReplay.replaceAll(USEFULSIGN, "");
    } else
        return aimlReplay;
    String dbReplay = askToDB.response(asDBInput);
    if (0 == dbReplay.length()) {
        return getRandomResponse();
    } else
        return dbReplay;
}

private String getRandomResponse() {
    return NULLREPLAY[getRandomNum()];
}

private int getRandomNum() {
    return (int) (Math.random() * NULLREPLAY.length);
}

private final String[] NULLREPLAY = { "对不起，我还不能回答您的这个问题。",
        "唔，主人还没有教会我这个问题呢。", "我暂时还回答不了这个问题呢？" };
```

七．聊天机器人学习功能实现

其实之前在介绍 Aiml 标签的时候，有 2 个很重要的标签还没有介绍到，那就是<system>和<gossip>标签。

在我的 xml 语料库中有一个文件叫 Study.xml，它的内容如下：

```
<category>
    <pattern>T</pattern>

    <template>
    <think>
        <system>
          String learn(String question,String answer)
          {
            return question+":"+answer;
          }
        </system>
      </think>
    您已经进入机器人训练模式，请输入问题和答案，格式如下：Q问题A答案。例如：Q你好吗？A我很好。或者你可以在问题中加入"*"符号，比如：Q你*好a谢谢，
    当你输入你真好或你很好时，机器人都会回答谢谢。
    </template>
</category>

<category>
    <pattern>*我教你*</pattern>
    <template><srai>T</srai></template>
</category>


<category>
    <pattern>Q*A*</pattern>
    <template>
        我学到啥！下次再来和我聊天我会变的不一样哦！
        <gossip>
            <system>learn("<star index="1"/>","<star index="2"/>")</system>
        </gossip>
    </template>
</category>
```

（ps:如果不懂这里面一些标签的功能，可以回顾之前的标签功能解释）

我们看看这样的测试结果是什么：

```
you say>你的主人帅吗?
Alice>对不起，我还不能回答您的这个问题。
you say>那我教你!
Alice>您已经进入机器人训练模式，请输入问题和答案，格式如下：Q问题A答案。例如：Q你好吗？A我很好。或者你可以在问题中加入"*"符号，比如：Q你*好a谢谢，当你输入你真好或你很好时，机器人都会回答谢谢。
you say>Q你的主人帅吗？A帅到爆棚!
Alice>我学到啥！下次再来和我聊天我会变的不一样哦！
```

等重启聊天机器人程序的时候，问同样的问题：

```
you say>你的主人帅吗?
Alice>帅到爆棚
```

这其中都做了哪些工作呢？下面解释：

system 标签的工作原理我还不是很清楚，但是我们可以看一下 AIML 解析器对应的 System.java 里面的 process 方法干了什么：

```
public String process(Match match) // 这个函数很重要
{
    try {
        AliceBot bot = match.getCallback();
        Context context = bot.getContext();
        Interpreter interpreter = (Interpreter) context
                .property("beanshell.interpreter");
        if (interpreter == null)
            return "";

        String script = super.process(match);
        interpreter.variable("result", null);
        interpreter.variable("match", match);

        Object evaluated = interpreter.evaluate(script);
        Object result = interpreter.variable("result");
        if (result == null)
            result = evaluated;

        interpreter.variable("match", null);

        return (result != null ? result.toString() : "");
    } catch (Exception e) {
        throw new RuntimeException("Evaluation error on <system> tag", e);
    }
}
```

这个函数我暂时解释不清楚，但我知道上面的语料库中 system 标签只是会被 learn 函数的返回值替代。

我重点介绍一下 gossip 标签的工作过程：

首先看 AIML 解析器中对应的 gossip.java 文件的 process 方法干了什么：

```
public String process(Match match)
{
  AliceBot bot = null;
  Context context = null;
  if (match != null) try
  {
    bot = match.getCallback();
    context = bot.getContext();
    context.print(super.process(match));
  }
  catch (Exception e)
  {
    throw new RuntimeException(e);
  }

  return "";
}
```

match 参数其实已经封装了上述语料库中 learn 函数的返回值“你的主人帅吗:帅到爆棚”。而 super.process(match)就是取出这个字符串。

然后我们在看 print 函数：

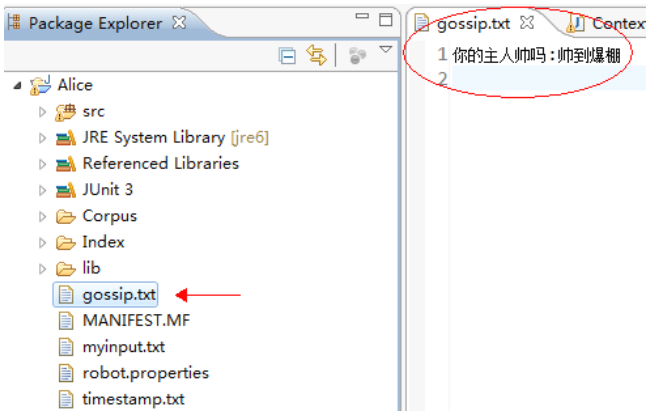
```
public void print(String output) throws IOException {
    outputStream().write(output.replace(" ", "").getBytes());
    outputStream().write('\n');
}
```

看起来像是在什么文件里面写入了什么内容，我们在看 outputStream()函数：

```
public OutputStream outputStream() throws IOException {
    if (output == null) {
        String path = (String) property("bot.output");
        File file = new File(path);
        if (file.isDirectory())
            path = file.getPath() + "/gossip-" + id() + ".txt";// 这里的id()是干什么的
        outputStream(new FileOutputStream(path, true));
    }

    return output;
}
```

上述代码中的 path 其实就是指的这个文件：



很显然，客户每一次对机器人“教学”的内容，都会被写入 gossip.txt 文件当中，而且在写操作时是 append 的（即不会把原来的覆盖掉）。

而当每次重启机器人程序的时侯，GossipLoad.java 类都会去读 gossip.txt 的内容，并构造一个 gossip.xml 文件将其写到预料库中.

该类主要代码如下：

```
public class GossipLoad {
    private String gossipPath = "./gossip.txt";// 数据源文件
    private String destination = "./Corpus/Chinese/gossip.xml";// 目标文件
    private InputStream inputStream = null;
    private BufferedReader bufReader = null;
    private OutputStream outputStream = null;

    /**
     * 构造一个xml
     *
     * @throws IOException
     */
    public void load() throws IOException {
        String line;
        init();
        StringBuffer stringBuf = new StringBuffer();
        stringBuf
                .append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<aiml>\n");
        // 读取gossip.txt文件内容
        while ((line = bufReader.readLine()) != null) {
            stringBuf.append(analyzing(line) + "\n");
        }
        stringBuf.append("</aiml>");
        outputStream.write(stringBuf.toString().getBytes());
    }

    /**
     * 构造一个category
     *
     * @param line
     * @return
     */
    private String analyzing(String line) {
        String[] pattern = line.split(":");
        return "<category><pattern>" + pattern[0] + "</pattern><template>"
                + pattern[1] + "</template></category>";
    }
```

Load 函数在加载 xml 语料库到内存之前调用。这是必须的，因为必须先通过 load 函数生成 gossip.xml 文件后，然后统一加载到内存中去。

代码如下：

```
public class AskToAIML implements IAskApproach {

    private static AliceBotMother mother = null;
    private static AliceBot bot = null;
    private static GossipLoad gossipLoad = null;

    public AskToAIML() {
        gossipLoad = new GossipLoad();
        try {
            gossipLoad.load();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            gossipLoad.clean();
        }
        mother = new AliceBotMother();
        mother.setUp();
        bot = mother.newInstance();

    }

    public String response(String input) {
        return bot.respond(input);
    }
}
```

这部分代码将把所有的xml预料文件加载到内存中

Gossip.xml 中的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml>
<category><pattern>你的主人帅吗</pattern><template>帅到爆棚</template></category>
</aiml>
```

八．聊天机器人存在的不足

1.数据库匹配做的不好，或者说匹配率低，而且还不是很准确

2.两种语料库的结合显的有点牵强

3.xml 语料库的设计还是比较欠缺的，首先预料不够丰富，而且靠人工编写预料不是一个好办法