AnswerSeeker:基于互联网挖掘的智能问答系统①

阴红志　张　帆　丁　鼎　赵　斌　(中国矿业大学 计算机科学与技术系 北京 100083)

摘　要： 智能问答系统是一种处理自然语言的新型的信息检索系统。介绍了 AnswerSeeker 智能问答系统，该系统采用了模块化和可扩展的框架，以便整合多种智能问答技术和多样化数据源。通过将与语言无关的代码和语言相关的代码分离，并且将语言相关的代码封装为组件，只要替换相应的组件，该系统可以适用于多种语言。由于很多自然语言处理技术还没有针对中文的，目前为止，我们系统的内核只支持英文，所以将以英语自然语言为例介绍 AnswerSeeker 的架构和工作原理。该系统采用了两种互联网挖掘的方法来寻找问题的答案：知识挖掘和知识诠释。AnswerSeeker 使用网络作为一个知识源，当然它也可以使用其他小的语料库或面向专业领域的知识库作为知识源。此外，提出了一种新的问题的表示和答案提取的方法—文本模式，文本模式分为问题模式和答案模式；其中问题模式用来表示问题，答案模式用来提取精确的答案。AnswerSeeker 通过将问题-答案对作为训练数据，自动学习答案模式。实验表明将互联网作为知识源，将模式学习和知识诠释的技术集成在同一系统中进行答案挖掘是一种这种很有前途的方法。

关键词： 互联网挖掘 知识挖掘 知识诠释 模式学习 智能问答

AnswerSeeker: Question Answering System Based on Web Mining

YIN Hong-Zhi, ZHANG Fan, DING Ding, ZHAO Bin

(China University of Mining & Technology (Beijing), Beijng 100083, China)

Abstract: This paper describes the AnswerSeeker question answering engine, a modular and extensible framework that allows integrating multiple approaches to question answering in one system. It supports the two major approaches to question answering, knowledge annotation and knowledge mining. In addition, it proposes one novel approach to question interpretation which abstracts from the original formulation of the question. Text patterns are used to interpret a question and to extract answers from text snippets. Our system automatically learns the patterns for answer extraction, using question-answer pairs as training data. Experimental results reveal the potential of AnswerSeeker.

Keywords: Web mining; knowledge mining; knowledge annotation; pattern learning; question answering

1　引言

Q&A 是处理自然语言问题的一种信息检索的新形式。用户向智能问答系统输入用自然语言表示的问题，系统在知识源中搜索用户想得到的信息，并将一个或多个精确的答案返回给用户。传统的信息检索系统，例如 Google、Baidu 等搜索引擎，强迫用户将问题转化为若干关键词，有些搜索引擎允许使用少量的逻辑操作符来表示关键词之间的关系。问答系统采用了一种更人性化、更灵活的方式来完成信息检索的任务，用户与该系统交互就像与一位智者交谈一样自然。此外，用自然语言表达的问题比关键词更具有表达力，问题中的疑问词可以表达用户想要得到问题答

① 基金项目:北京市大学生科学研究和创业行动计划(0832)
收稿时间:2009-03-04

案的类型，例如一个以疑问词 when 开头的问题表明用户期待得到一个日期类型的答案，而以 who 开头的问题表明用户想得到一个合适的人名；这样，问答系统就可以直接返回给用户一个精确的答案，而不是一个包含答案的整个文档。

Q&A 的概念并不是现在才有，早期的 Q&A 系统是面向某一专业领域的，很多人更愿意把这些 Q&A 称为专家系统，而现在大多数研究人员研究的焦点是面向开放领域的 Q&A。通常，互联网或者封闭的语料库(例如数以百万计的新闻报道文章组成的集合)会被作为开放领域的非结构化知识源，有时一些结构化的(例如关系数据库)和半结构化的数据源(例如 xml 文档)也会作为知识源。将网络作为知识源的 Q&A 通常使用统计的方法从大量重复、冗余的信息中挖掘答案，而把封闭的语料库作为知识源的 Q&A 系统则有必要应用一些很复杂的自然语言处理技术来完成问题和答案的匹配工作。

在本文中，我们描述了面向开放领域的基于互联网挖掘的智能问答系统 AnswerSeeker，由于该系统是模块化的，所以一些新的问题处理和答案提取技术很容易集成到该系统当中；并且该系统在设计和开发过程中，我们将语言相关的代码和语言无关的代码相分离，将语言相关的代码封装为组件，这样，只要开发那些语言相关的组件，该系统内核就可以支持多种语言。但是，该系统的内核现在只支持英文。为了方便用户使用，我们在系统内核和用户接口之间增加了一个语言处理的模块(语言转换器)，该模块主要由 Google 翻译 API 和 Yahoo 翻译 API 组成，主要负责将用户输入的非英文的问题转换为用英语表示的字符串并交给系统内核处理，将系统返回的英文答案转换为用户使用的语言并返回给用户。针对互联网的特征，AnswerSeeker 在寻找答案时使用了两种方法：知识挖掘和知识诠释。前者用于从搜索引擎返回的非结构化的文档片段中挖掘答案，在该方法中，我们实现了模式学习的方法，AnswerSeeker 通过问题-答案对训练集自己学习答案模式，然后利用这些答案模式去提取精确答案，此外，我们利用问题模式创建保留了语义信息的问题解释(interpretation)，这些解释将作为查询交给搜索引擎检索整个互联网。虽然这种知识挖掘的方法对于大部分问题可以获得很精确的答案，但是答案的质量有时并不令人满意，因为每个人都可以自由地在互联网上发布信息。为了提高答案质量，我们利用知识诠释的方法对某些特定类型的问题从专业的网站或网络服务那里获得即精确又准确的答案，例如某用户询问关于公交线路的问题，我们的系统将通过知识诠释的方法到公交网获取用户问题的答案。

2 相关工作

现在的大多数问答系统都遵循这样一种架构[1](如图 1)，该架构主要由以下五个模块组成一条流水线：question analysis, query generation, search, answer extraction and answer selection。但是不同的系统可能会将两个或多个相邻的模块在一个模块中实现，例如 JAVELIN[2,3]系统将 question analysis 和 query generation 两个模块在同一组件中实现，我们的 AnswerSeeker 系统将 answer extraction 和 answer selection 两个模块在同一模块中实现。

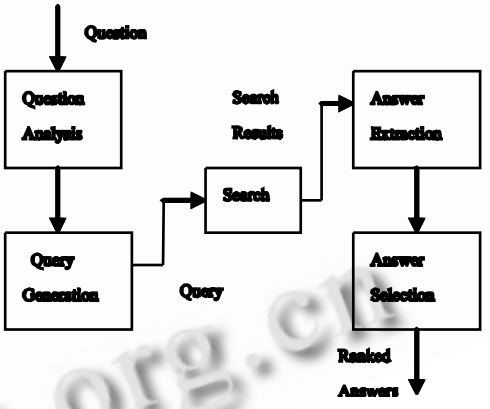

图 1 Q&A 系统的流水线式架构

Lin 等人最先在他们的 START[4]和 Omni-base[5]系统中使用了知识诠释技术[6]。知识诠释技术主要用于从半结构化或结构化的知识源中提取答案，这样的知识源通常是一些网站或网络服务，例如 Biography.com 专门提供关于知名人物的出生日期、去世日期和寿命等信息；CIA World Factbook 专门提供世界上所有国家的政治、经济和地理等信息；GobalWeather 专门提供关于全球天气的信息；Wikipedia 提供关于一些概念的定义及相关内容的信息。这些知识源可以回答我们日常生活中 20%-30%[5]的问题。知识挖掘主要是将统计方法应用到非结构化知识源的整个互联网，而不是半结构化知识源的某个网站或网络服务，这种技术通常是基于搜

索引擎。AnswerSeeker 同时使用了这两种技术，并在此基础上使用了模式学习的技术。

为了使读者更容易理解 AnswerSeeker 以及其它将网络作为知识源的问答系统的工作原理，我们有必要阐述一下互联网和单个网站(或网页文档)作为知识源的区别。网络即可以指整个互联网，也可以指单个网站。通常我们说互联网是非结构化的数据源或知识源，是从搜索引擎返回的结果的角度来说，因为搜索引擎所返回的结果是与查询相关的网页文档的片段或摘要，而这些文档片段是非结构化的，并且 Q&A 系统使用搜索引擎的时候，也只是利用搜索引擎返回的网页文档的片段或摘要作为知识源，并不是真的根据每个文档片段下面的超链接去下载整个网页文档然后逐个分析每个文档，这样太消耗时间了，用户是无法忍受的。从搜索引擎的角度看整个互联网是很有道理的，因为迄今为止，搜索引擎是我们唯一可以检索整个互联网的工具。而单个网页是指半结构化的 html 或 xml 文档。此外，很多网页是 Web 服务器根据用户发送的服务请求从数据库中提取信息而动态生成的，这种动态网页中的信息一般很难被搜索引擎搜索到。

LAMP 问答系统[7]采用了模式挖掘和匹配的方法。该系统通过手工创建 question pattern，然后用 question pattern 对问题进行匹配分类，从问题中提取一个短语，该短语一般是问题中的一个对象或事件。第二个文本模式是 answer pattern，该模式是系统自动学习获得的，用于从搜索的结果中提取答案。但是 LAMP 只能处理一个问题中包含一个关键短语的情况，下面这类问题它是无法处理的，例如

“How many calories are there in a Big Mac?” (TREC8, question 56)

因为在该问题中包含了两个短语“calories”和“Big Mac”。LAMP 的另一个缺点就是问题分类太粗糙了，问题也种类很少，只有 22 个问题类型，这样导致所提取答案的准确度大大降低。我们的 AnswerSeeker 系统采用更一般性的方法来表示一个问题，允许在一个句子中有多个关键短语的出现，并且细化和增加了问题的分类，从而提高了答案的准确性。

3 系统架构

由上所述 AnswerSeeker 是一个模块化框架，系统很容易把其他技术或知识源集成进来并对其进行扩展。本文介绍的 AnswerSeeker 系统内核只嵌入了针对英文的语言组件和资源文件。但为了方便用户使用该系统，我们在系统内核和用户接口之间加了一个语言处理模块，该模块主要完成其他语言和英语之间的转换功能。AnswerSeeker 的系统架构如图 2 所示。

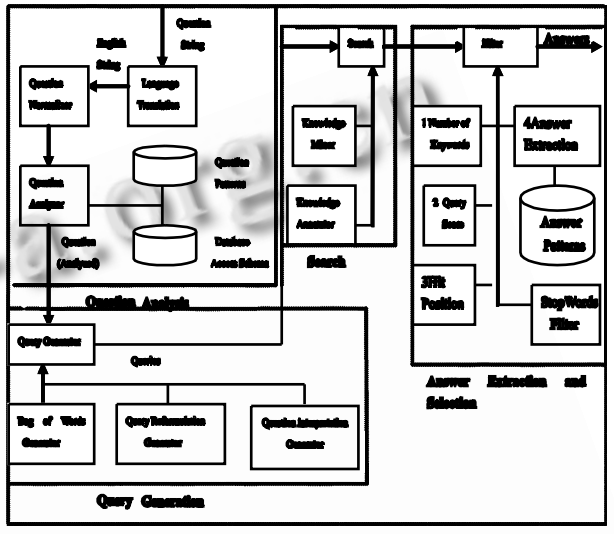

图 2　AnswerSeeker 的整体架构

3.1 问题分析

Language Translation 模块通过调用 Google 在线翻译 API 和 Yahoo 在线翻译 API 将用户输入的问题转化为用英文表示的问题，并记录用户使用的语言，然后将系统产生的答案转化为用户所使用语言的种类。这样，该系统可以通过多种语言与用户交互。

Question Normalizer 组件应用一系列规范化步骤将问题字符串转换为两类规范化的形式：一类用于问题分析，另一类用于用户查询生成。不论哪一类的规范化过程，我们都要将问题字符串中多余的空格和标点符号去掉，然后将疑问词的第一个字母变为小写，将缩写的动词还原，例如“who’s”要还原为“who is”。此外，用于问题分析的一类还要将问题字符串中的动词和名词还原，包括将名词复数还原为名词单数，将动词过去式、过去分词和第三人称单数转化为动词原形；生成的用户查询不需要将动词和名词还原，单如果问题字符串中含有助动词，需要将助动词和疑问词去掉，并改变问题字符串中动词的时态。如：“When did Shakespeare write Hamlet?”，一个可以回答该问题的文字段落可能包含这样的字符串“Shake-

speare wrote Hamlet in……”，但是该字符串中并不含有助动词“did”，当然也不会包含疑问词“when”，并且动词的时态是过去式。表 1 给出了用于查询生成的一类的规范化规则。我们 通过 POS (part of spee－chtagger) 技术来识别问题中主要动词，由表 1 不难看出，在某些情况下我们需要将助动词或情态动词移到谓语动词前面，而在其他情况下我们需要将助动词去掉，相应改变谓语动词的时态。。例如：问题字符串为 “When did Shakespeare write Hamlet ?”，

表 1 含有辅助动词问题的范化规则

Auxiliaries in questions	Transformations
Is/are/were/was/ [...] gerund/pastparticiple	is/are/were/was/ gerund /past participle
can/could/will/ should/shall/ may/ may/might/must[...] infinitive	can/could/will/ should/shall /may/ might/ must/ infinitive
have/has/had[...] past participle	have/has/had past participle
do [...] infinitive	infinitive
does [...] infinitive	the third person singular present tense
did [...] infinitive	simple past

规范化后生成的第一类形式为 “when do Shakespeare write Hamlet ”，规范化后生成的第二类形式为 “Shakespeare wrote Hamlet ”，第一种类型规范形式用于问题分析，第二种类型规范化形式用于查询生成。

Question Analyzer 组件完成对问题的分类。问题的分类主要有两层含义：第一层是根据 Database Access Schema 中的问题模式(正则表达式)来决定哪些问题可以由 Knowledge Annotation 来寻找答案并为后面的 Knowledge Annotator 生成查询，为了保证能找到答案，这些可以由 Knowledge Annotation 找到答案的问题也会交给 Knowledge Miner 去挖掘答案，但是由 Knowledge Annotation 返回的答案的分数要比 Knowledge Miner 挖到的答案的分数高得多，因为 Knowledge Annotation 是从比较专业的网络服务那里获得答案，系统认为其答案质量应比从互联网中挖掘出的答案质量高得多。如果前者获取答案失败，系统将把后者获取的答案返回给用户。第二层是将交给 Knowledge Miner 的问题进行抽象和概括形成问题的 interpretation。在该系统中使用了两类文本模式：question pattern 和 answer pattern。question pattern 就是用于对问题进行抽象和概括形成问题的 interpretation。这种方法是建立在下面这种假设之上的：任何一个问题都可以分解为以下三部分：Property，Target，Context。我们将这三部分称为问题的 interpretation，一个问题通常是问 a property of a target in a specific context。让我们看下面一个例子：

“How many calories are there in a Big Mac?”

这个问题的 interpretation 如下：

- Property: NUMBER
- Target: “calories”
- Context: “Big Mac”

3.2 查询生成

Query Generator 模块为 Knowledge Miner 生成问题的查询，在该系统中我们对每一个问题至少生成三类不同的查询，然后将这些查询交给 search 模块，这些查询是并行处理的，我们通过多线程实现，为每一个 query 产生一个线程。每个查询都是被打分的，查询的分数越高，由该查询返回的答案分数越高，分数越高的答案就会排在前面。在该系统中，为 Knowledge Annotation 生成的查询分数最高，其次是 Query Reformulation Generator 为 Knowledge Miner 生成的查询，由 Bag of Words Generator 生成的查询分数是最低的。

Bag of Words Generator 模块从 Question Normalizer 产生的第二类规范化形式中提取关键词，并将一些只起语法功能的词和高频词去掉，生成一个问题的查询。

Reformulation Generator 将原始问题字符串转化为一组查询。每一个问题模式(question pattern)和一组查询模式(query pattern) 以及这些查询对应的分数 (score) 都被定义在了一个资源文件当中。问题模式是用 java 语法形式表示的正则表达式，每个查询是由原始问题当中的子字符串组成的。让我们看一个这种资源文件的例子，方括号中的数字是指正则表达式中的组(group)号，操作符‘<’是指将操作符右边的组(group)放在操作符左边组(group)的每两个单词中间。

下面是某一查询重构的资源文件：

Question Pattern:
When (is | are | was | were|) (.*)
Question Reformulation:
[2]
3.0
[2][1]　　[2] < [1]
4.0　　5.0
[2][1] in　　[2] < [1] in
5.0　　6.0
[2][1] on　　[2] < [1] on
5.0　　6.0
[2][1] at　　[2] < [1] at
5.0　　6.0

下面让我们以问题“When was Einstein born”为例来分析他们是如何工作的，该问题匹配上面例子中的问题模式。其中表达式[1]指的是问题中的“was”，表达式[2]指的是问题中的“Einstein born”。该问题生成的一组查询如下：

- “Einstein born”　(score: 3.0)
- “Einstein born was”　(score: 4.0)
- “Einstein was born”　(score: 5.0)
- “Einstein born in”　(score: 5.0)
- “Einstein was born in”　(score: 6.0)
- “Einstein born on”　(score: 5.0)
- “Einstein was born on” (score: 6.0)
- “Einstein born at”　(score: 5.0)
- “Einstein was born at”　(score: 6.0)

Question Interpretation Generator 将 Question Analyzer 模块生成的 question interpretation 转化为搜索模块的查询。该模块生成的查询由 question interpretation 中的 Target 和 Context 连接而成，并加上了 Bag of Words 模块从问题中提取的关键词。我们仍以“How many calories are there in a Big Mac? ”为例，该问题由该模块生成的查询如下：

“calories”“Big Mac”calories Big Mac

该查询的分数会比 Bag of Words 生成查询的分数高，但会低于由 Reformulation Generator 生成查询的分数。

3.3 知识检索

互联网中大部分信息是可以被搜索引擎检索到的，但是也有一部分是很难被搜索引擎检索到的，例如提供全球天气预报信息的 Global Weather、提供影视信息的 Internet Movie Database 和提供公交路线查询的各种网站等网络服务。针对这两种互联网资源，我们在信息检索模块使用了两种不同的信息检索方式：知识挖掘和知识诠释。

KM(Knowledge Miner)和 KA(Knowledge Annotator)是作为线程实现的，这样每个问题对应的多个查询可以并行执行，然后把多个查询结果聚集交给下一个模块(Answer Selection and Extraction)处理。

每个 KM 通过调用传统的信息检索系统提供的 API 接口来检索整个互联网。AnswerSeeker 实现了两个 KM，一个利用 Google 搜索引擎来检索互联网，另一个利用 Yahoo 搜索引擎检索互联网，Google 和 Yahoo 都为应用程序提供了 API 接口。每个 KM 返回的是网页文档的片段甚至整个网页文档。这些返回的结果从不作为答案提供给用户，而是交给答案选择组件将结果划分为句子，然后由答案提取 组件从句子中提取精确答案再返回给用户。答案选择组件还负责为每个结果打分，从而建立一个排好序的答案列表。为了防止由 KM 返回的结果直接交给用户，我们将这些结果的初始分数设为一个最小值 0。

这种 KM 方法有一定的不足，由 KM 返回的文档的片段在长度上是有限制的，这样导致结果中的句子有可能被压缩了，从而能回答问题的短语(答案)没有出现在这些文档片段中。另一种做法是跟据 KM 返回的结果中提供的超链接下载整个网页文档，对整个文档进行分析并从中提取答案，但是这样会消耗大量时间，在用户可以忍耐的时间内，只有少数几个文档可以被解析。

KA 用于针对一类特定的问题去检索半结构化或结构化的网站或网络服务。来自这些知识源的答案往往都是正确的，所以这些答案的分数被设为最高，并且由 KA 返回的结果可以直接作为答案返回给用户，不必交给答案选择和提取模块处理，因为由 KA 返回就是问题的精确答案而不是网页文档或文档的片段。在后面一节中会给出关于 KA 的详细介绍。

3.4 答案选择和提取

该模块由一系列过滤器组成，主要对 KM 返回的文档片段进行排序并提取精确的答案。每个过滤器代

表着一个具体的特征,通过检查候选答案(指文档片段)是否满足该特征去除潜在的不能回答问题的候选答案，从而缩小候选答案的集合。

一个用于判断候选答案质量的简单特征是候选答案所包含的查询中关键词的个数。一个候选答案就会被过滤掉，如果它不满足下面公式:

$$M \geq \left\lfloor \sqrt{K-1} \right\rfloor + 1 \quad (1)$$

在上面公式中，K 指查询中关键词的个数，M 指候选答案中所包含的关键词个数。如果一个候选答案满足上面的公式，那么它的分数会被增加，增量正比于 M。这个公式是很有道理的，如果一个查询中包含 3 个关键词，那么根据该公式候选答案中至少包含 2 个关键词才不会被过滤掉。

另一个用于判断候选答案质量的特征是查询生成器赋予查询的分数(见第 3.2 节查询生成)。所生成查询被赋予的分数越高，表明该查询越明确，由该查询检索互联网得到的答案越与问题相关。这样,在 Query Score 过滤器中，候选答案的分数会增加，增加量正比于相应查询被赋予的分数。

通常，搜索引擎返回的结果是按这些结果与查询的相似度排好序的。在 Hit Position 过滤器中，我们将这一因素考虑在内。每个候选答案在该过滤器中分数都会减少，减少量正比于它在搜索结果中的位置，如果排在搜索结果中首位的候选答案，那么它的分数减少量将是最少的。

Answer Extraction 是最重要的一个过滤器。每一个候选答案都对应着一个查询，而每一个查询都对应着一个由 Target，Context 和 Property 组成的问题解释(question interpretation)。这些信息和相应的 answer pattern 用于从候选答案中提取精确答案。(详细信息和原理见第 5.0 节模式学习)

Stopwords 过滤器是最后一个过滤器，该过滤器用于过滤已提取的精确答案。如果被提取的精确答案满足下面任意一个条件，该答案就会被过滤掉。

- 该答案是一个语法功能词，但不是数量词。例如“they”“here”“there”“many”等。
- 该答案中只包括括号的左半部分或右半部分，单引号或双引号的左半部分或右半部分，而缺少这些成对符号的另外一部分。
- 答案以疑问词开始或以问好结尾。
- 答案为空

最后，系统将通过所有过滤器检查的精确答案按分数从高到低返回给用户，我们可以设置返回给用户答案的最多个数，因为我们没有必要将所有答案都返回给用户。

4 知识诠释

互联网中存在着大量的半结构化和结构化的知识，这些知识通常来自提供某一类特定信息的网站或网络服务，并且大部分位于公共网关后面的数据库中，搜索引擎是很难检索到的。为了有效利用这些异构的、分布式的网络资源，它们必须在统一的接口或查询语言下进行集成。数据库技术和概念为我们完成这项工作提供了工具。基于数据库访问模式的知识诠释技术可以有效地利用这些知识回答用户提出的问题。不同用户通常会问同一类型的问题，这样用户问的问题虽然很多，但问题类型通常是有限的，这样我们只要精心选取若干个提供这些类型信息的网站或网络服务，就能覆盖很大一部分用户提出的问题。在我们系统上已经证实，精心选取的 10 个网络资源(网站)可以回答 TREC-9 会议 QA track 中 27%的问题和 TREC-2001 会议[8]47%的问题。

知识诠释技术的前身是自然语言诠释技术，由麻省理工学院 Katz 等人提出，最早应用在 MIT 开的 Start[5]和 Omnibase[4]系统中。Start 系统是一个自然语言理解系统，Omnibase 是一个向异构的、分布式网络资源提供统一接口的虚拟数据库。

4.1 数据库访问模式(Database Access Schema)

AnswerSeeker 的知识诠释组件由数据库访问模式和 Knowledge Annotator 组成(见图 3)。

数据库访问模式位于该系统的问题分析模块中，每个模式包含两部分：问题模式和数据库查询，这两部分总是成对出现。一个问题模式是由一组正则表达式组成，对应着一类特定的问题，例如下面这个询问某地天气的问题对应的数据库访问模式:

Question pattern:

(what | how) be the weather (like)? (in | at) (.*) (yesterday | today| tomorrow)

Database Query:

(globalweather.org [3] # [4] # weather)

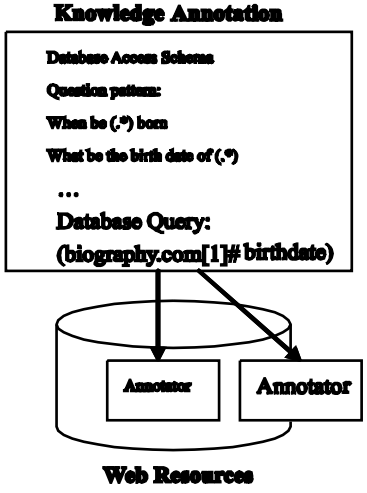

图 3　知识诠释组件

在该例子中，AnswerSeeker 将询问某地某时天气的问题转化为一个 object－property－value 型[]的数据库查询。在该查询中，明确了寻找答案的知识源(gobalweather.org)，问题对象([3]# [4])和问题所关注的对象属性(weather)；该查询返回的对象的属性值就是问题的用户想得到的问题的答案。查询语句中的某个参数可能为空，在本例的问题中可能并不出现时间，导致查询中的参数[4]的值为空，我们在处理查询语句时，把值为空的参数赋一个默认值，在本例中，如果参数[4]为空，我们将“today”设为默认值并赋给参数[4]。

在查询中允许有多个参数(如[3]# [4])，我们在处理查询时认为最后一个#号后面的名词短语或名词是对象的属性，前面若干用#号隔开的是拥有修饰或限定成分的对象。

AnswerSeeker 在问题分析阶段，通过将用户的问题与 Database Access Schema 中的问题模式匹配，找到所匹配的问题模式并生成相应的数据库查询，将数据库查询交给信息检索模块的 Knowledge Annotator 解释执行。

我们需要为每一个提供服务的网站开发一个 KA，因为不同的网站或网络服务提供的 Web GUI 或 XML API 不同，并且返回的包含答案的网页结构也不相同，我们需要为每一个网站返回的网页开发一个 html parser，负责解析包含答案的网页文档和提取精确的答案。所以，一个 KA 由两部分组成：针对某一网站的 Web GUI 的调用 和 html parser。KA 在执行数据库查询的时候，根据该网站的 Web GUI 或 XML API 动态生成 http request，利用 html parser 对返回的 html 文档进行解析、提取答案。

4.2 知识工程

在本节中我们将介绍如何构建数据库访问模式和如何执行数据库查询语句。

教会 AnswerSeeker 的 Knowledge Annotation 组件能回答多种类型的自然语言问题包括以下三个步骤：①识别问题类型和知识源；②为每个问题类型写数据库访问模式；③为每个知识源开发 KA。

知识工程的第一步就是识别用户经常会问哪些问题并对这些问题进行分类，找到能回答这些问题的知识源。经验性地分析问题分布会收到事半功倍的效果。例如我们注意到用户经常会问一些关于天气和出行路线的问题，这些问题的答案分别可以由 global weather.org 和 ditu.google.com 两个网站提供。

一旦问题的类型和知识源确定后，我们要用正则表达式写出每一问题类型对应的问题模式和相应的数据库查询语句。虽然不同的人在表达同一问题时可能使用不同的表达方式，但对于同一类问题人们常用的表达方式是有限的(尤其是在英文中)，一般不超过 20 种，例如我们询问某地的天气时常用的表达方式只有两种：what is the weather like in somewhere 和 how is the weather in somewhere 。所以在该阶段，我们要为每一类问题尽可能多的想出多种提问方式，然后将这些问题中涉及到的具体对象(object)用组(.*) 代替，用正则表达式将这些提问方式合并，最后将所有动词和名词还原就得到该类问题的问题模式了。写出问题模式后，我们根据该类问题的知识源和在问题模式中对象(object) 对应的组号写出该类问题的查询语句。假设我们获得了以下问题模式：What be GDP of(.*)，能回答该类问题的知识源是 world factbook.org，问题所涉及对象(object)对应的组号是 1，我们用[1]表示，则该类问题对应的查询语句为(worldfactbook.org[1] # GDP)。

数据库访问模式创建完以后，我们要为每一个知识源开发相应的 KA。在 AnswerSeeker 系统中，我们为 28 个数据库访问模式确定了 7 个数据源，所以在我们系统中有 7 个 KA。下面是一个 KA 的例子：

Biography.com 该网站专门提供关于著名人物

的出生日期、去世日期和寿命等信息。当回答关于这些属性的问题时，Answer Seeker 中针对该网站的 KA 会动态生成 http request 并向该网站发出服务请求；Biography.com 收到服务请求后，将包含问题答案的网页文档返回给 Answer Seeker ，我们的系统利用 KA 中特定的 html parser 对问题进行分析并从中提取精确的日期。

5 模式学习方法

我们的模式学习方法主要用于知识挖掘中的问题表示和答案提取。模式描述方法使用两种文本模式：问题模式和答案模式，前者用于解释问题，后者用于从文档片段中提取精确答案。最初问题模式和对应的属性手工创建，系统根据问题-答案对自动学习答案模式。

5.1 问题表示

我们的模式学习方法是基于这样一个假设的：任何一个问题都可以分解为以下三部分：Property，Target 和 Context。一个问题通常是询问一个目标对象在某一特定背景下的某一属性值，即 A question asks for a property of a target in a specific context。例如在第 3.1 节中提到过的例子，给定一个问题“How many calories are there in a Big Mac？”，该问题可以被解释成以下三个部分：

- Property: NUMBER
- Target: “calories”
- Context: “Big Mac”

这三部分共同组成了一个问题的解释(question interpretation)。

AnswerSeeker 知道 70 个属性(Property)，如一个事件发生的时间(DATE)，一个组织或者个人的姓名(NAME)，一个简写的全称(LONGFORM)等。每个属性和 1-20 个常用的问题模式相关联，并且该属性及其对应的这些问题模式被定义在了以该属性名命名的同一资源文件中。问题模式是正则表达式的派生，在正则表达式的基础上，我们加入了用来表示目标对象的标签<T>和用来表示目标对象背景的标签<C>，在一个问题模式中有且只有一个目标对象标签<T>，0—N 个背景标签<C> 。每个标签都可以和对象类型组合使用来限制对象的类型，例如<T_ABBR> 表示目标对象必须是缩写形式的，如 GDP。图 4 是属性 NAME 的资源文件。

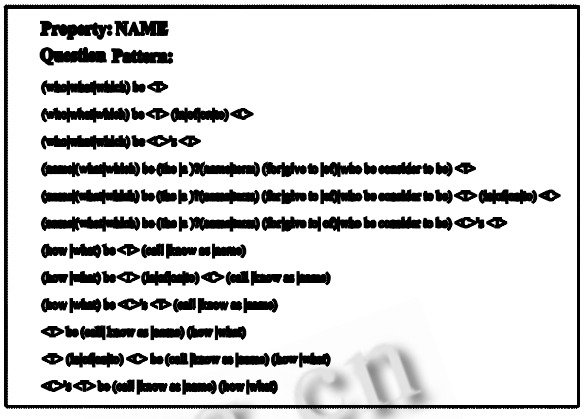

图 4 属性名 NAME 的资源文件

Answer Seeker 通过依次将问题和所有问题模式匹配来确定问题的解释(interpretation)。问题模式在匹配前，模式中的<T>和<C>标签会被正则表达式中组(.*)代替，而组合类型的标签会被更具体的正则表达式中的组代替，例如 <T_ABBR> 会被([A-Z0-9][A-Z0-9]+) 代替。如果一个问题模式和该问题匹配，那么该问题模式对应的属性名就是该问题的属性(Property)，问题模式中的<T>和<C>从问题中提取出的子字符串就是该问题的目标对象(object)和背景(context)。我们上面例子中的问题可能会与下面这个问题模式匹配“how many <T> be (there)(in |on |at) <C>”，对应着属性 NUMBER。

基于属性(property)的问题分类是很有道理的。虽然覆盖所有问题可能问及的属性是不可能的，但也是不必要的，因为同一问题可能会有多种解释(interpretation)方式，例如下面这个问题“What is the name of the wife of Bill Clinton”可以有如下的解释(interpretation)：

- Property: WIFE, Target: "Bill Clinton"
- Property:NAME, Target: "wife of Bill Clinton"

所以，即使我们没有属性 WIFE，该问题仍能被更一般的属性 NAME 正确解释。但是这又带来了另一个问题，如果同一问题可以被多个属性解释的时候，我们选择哪一个。假设我们系统中同时有 NAME 和 WIFE 两个属性，当问及上面的问题时，Answer Seeker 会选择 WIFE 属性，因为该属性比 NAME 属性更名确、更具体，更重要的是目标对象更简单，属性名 WIFE 拥有很多诸如 SPOUSE 等同义词。当然 AnswerSeeker 是不会考虑到这些语义的，它的选择

标准很简单，优先选择解释(interpretation)最短的，即一个 Target 和所有 Context 字节数之和最小的那个解释(interpretation)，显然“Bill Clinton”字节数要比“wife of Bill Clinton”少很多。

问题的解释(interpretation)会被交给查询生成模块中的 Question Interpretation Generator 子模块，该子模块将问题解释中(question interpretation) 的 Target 和 若干 Context 连接，并加上 Bag of Words 模块从问题中提取的关键词组成了问题的查询(query)。我们仍以“How many calories are there in a Big Mac?”为例，该问题由该模块生成的查询如下：

“calories”“Big Mac”calories Big Mac

该查询交给 Google 和 Yahoo 搜索引擎去检索整个互联网。

5.2 答案提取方法

每个属性除了与若干问题模式相关联外，还与若干答案模式相关联。这些答案模式用于从包含着目标对象(Target)和背景(Context)的文档片段中提取精确答案。答案模式的形式与上一节谈到的问题模式的形式很相似，每个答案模式都包含一个目标对象标签<T>和属性标签<P>，以及任意多个背景标签<C>。此外，每个答案模式都被赋予一个置信度，用来评估用该模式提取答案的可靠程度。

AnswerSeeker 用标签<T>和<C>将搜索引擎返回的文档片段中出现的目标对象和所有背景对象替换，然后用替换后的文档片段依次与所有答案模式匹配。在答案模式匹配之前，模式中的属性标签<P>会被正则表达式中的组 (.*) 代替。如果某一答案模式与文档片段匹配，那么文档片段中与该答案模式里的属性标签<P>相对应的字符串会被提取出来。在我们例子中，搜索引擎返回的文档片段“One Big Mac contains 560 calories and 32 grams of fat”被转换为“One <C> contains 560 <T> and 32 grams of fat”，答案模式“contains <P> <T>”可以用于从转换后的问文档片段中提取属性值 560。如果提取的字符串从未出现在候选答案的集合中，那儿该字符串会被当作新的候选答案添加到候选答案的集合中，并且将相应答案模式的置信度作为初始分数赋给这个新的候选答案；如果被提取出的属性值已经作为候选答案出现在了候选答案的集合中了，那么该候选答案的分数会被增加，增加量等于相应答案模式的置信度。这种机制产生了一个排好序的候选答案列表，并且最有前途的答案排在最前面，优先返回给用户。

5.3 模式学习

最初，问题模式和属性是我们通过知识工程手工创建的，然后 AnswerSeeker 通过将问题-答案对作为训练数据自动学习答案模式：(1)从搜索引擎返回的文档片段中提取答案模式；(2)对这些答案模式进行评估，将置信度太低的或太具体的答案模式去掉。

属性和问题模式。我们通过分析 TREC9[8]会议的 700 多个问题来决定问题属性和模式。首先，我们通过将这些问题中的目标对象和背景对象用标签<T>和<C>替换得到一个初始的问题模式的集合。然后将它们的第一个字母变为小写，最后的问号去掉，所有动词用原型代替，所有名词用单数形式代替，这是很有必要的，因为在问题分析阶段，我们使用范化的问题和这些问题模式匹配的。接着，我们使用正则表达式将一些问题模式合并(例如 who (assassinate |kill |murder |shoot) …)，此外，我们还利用我们的个人知识和各种词典(如 WordNet)，通过加入同义词进一步扩充问题模式。有时很难确定问题所问的属性应该划分为哪一类，例如一个形如“what is …”问题，它即可以询问一个术语的定义(DEFENITION) ，又可以询问一个地方的名字(NAME)。为了对这类问题有一个确定的解释(interpretation) ，我们将<T>标签和对象类型组合起来限制目标对象的形式。在问题模式预处理的时候，我们不能简单地用一般的正则表达式中的组(.*)代替，而需要用更具体的正则表达式组来代替，例如<T_ABBR>不能简单地用(.*)代替，它需用更具体的组([A-Z0-9][A-Z0-9]+)代替。在 AnswerSeeker 中有 4 种对象类型(如表 2)，这四种对象类型可以和标签<T>和<C>组合使用，例如<T_NE>表示目标对象必须是命名实体，<T_ABBR>表示目标对象必须是缩写形式。最后，我们将很少出现的问题属性去掉并且将一些相似的问题模式合并。

表 2　对象类型

Object type	Meaning	Capturing group
<ABBR>	abbreviation	([A-Z0-9][A-Z0-9]+)
<NOABBR>	noabbreviation	(.*[^A-Z0-9].*?)
<NE>	named entity	((?:an? the)?[A-Z]\w*)
<NONE>	no named entity	(.*?[^A-Z].*?[^A-Z]\w*)

答案模式的提取。AnswerSeeker 系统可以利用问题-答案对作为训练数据提取答案模式的。我们仍使

用来自TREC9会议的700个问题-答案对作为训练数据，当然也可以使用其他问题-答案对作为训练数据。首先，我们用已获得的问题模式对训练集中问题进行解释(interpretation)， 去掉不能被解释的问题。我们为每一个问题的解释(interpretation)创建一个含有目标对象、背景对象和问题答案的元组，例如问题-答案对“How many calories are there in a Big Mac”“560”对应的元组是(“calories” “Big Mac” “560”)，元组中的“calories” 是问题中的目标对象，“Big Mac”是问题中的背景对象，而“560”则是问题的答案。

AnswerSeeker 利用这些元组去查询 Google 和 Yahoo 搜索引擎，从搜索引擎返回的结果中获取文档片段。在我们的例子中，查询字符串是“calories” “Big Mac”“560”，在该查询中含有问题的答案，因为我们感兴趣的是即包含目标对象又包含属性值的文档片段。AnswerSeeker 将文档片段中出现的目标对象、背景对象和属性值(问题的答案)分别用标签<T>、<C>和<P>代替。例如文档片段“One Big Mac contains 560 calories and 31 grams of fat”经转换后变为“One <C> contains <P> <T> and 31 grams of fat”。现在我们就可以利用下面两个正则表达式来提取答案模式了：

- \B<T>\B (.*?)\B<P>\B\s*(\W|\w+)
- (\W|\w+)\s*\B<P>\B (.*?)\B<T>\B

答案模式也是正则表达式的派生，在正则表达式的基础上增加了标签<T>和<P>。一个答案模式包含一个目标对象标签<T>和属性标签<P>，在这两个标签之间可以有任意多个字符；此外，在标签<P>前面或后面必须包含一个单词或一个特殊的字符。如果标签<T>在标签<P>的前面，那么在标签<P>的后面必须有一个词或一个特殊的字符；如果标签<T>在标签<P>的后面，那么在标签<P>的前面必须有一个词或一个特殊的字符。在我们的例子中，我们应用上面第二个正则表达式得到的答案模式为：“contains <P> <T>”，因为标签<T>在标签<P>的后面，所以我们在标签<P>的前面保留了一个单词“contains”。我们之所以为标签<P>保留一个词或一个特殊的字符是因为 AnswerSeeker 在利用答案模式提取精确答案前，先将标签<P>用正则表达式的组 (.*) 代替，如果在标签<P>的前面或后面没有一个词限制，我们将无法提取精确的答案。利用这种方法获得的答案模式，可以从文档片段中提取精确答案，但是由于这种答案模式太具体了，导致很多包含着目标对象、背景对象和答案的文档片段因找不到匹配的答案模式被丢掉了。为了获得更一般的答案模式，我们扩展了上面的学习算法。首先，利用上面的学习算法获得原始的答案模式。然后我们利用 OpenNLP 提供的命名实体识别器来识别这些原始答案模式中的命名实体(Named Entity)，用相应命名实体的类型名来替代这些命名实体。接着，我们将答案模式中的属性与命名实体的类型相关联，这样做是很有道理的：①如果一个属性的命名实体类型知道了，我们就没有必要在标签<P>的前面或后面保留一个词或特殊字符作限制条件了，这样我们可以得到更一般的答案模式。②可以保证只有期望类型的属性值可以被提取。最后，问题模式中所有的词或短语都是可选的，除两个特例外：①用来表示目标对象和属性之间关系的词必须保留；②出现在问题模式中的关键词必须保留。在下面我们看一个询问某人出生地的例子：

原始答案模式：<T> was born in 1879 in <P> and

一般化后的模式：<T> [?<]*born [?<]*(<NEdate>)? [?<]* <P_NElocation>

原始答案模式中的 1897 为时间类型的命名实体，故用时间类型的标签<NEdate>代替，属性<Property>为地点类型，故用<Property_NElocation>代替，由于 born 代表了目标对象和属性之间的关系，所以 born 必须保留，而其他<T>和<P>之间的词和短语都是可选的。此外，在该例子中，<T>和<P>之剑除了可以有时间类型的命名实体外，不能再有其他命名实体了。

我们在使用一般化的答案模式提取答案时，除了将文档片段中出现的目标对象和背景对象分别用标签<T>和<C>代替之外，我们还要用 OpenNLP 识别出文档片段中的命名实体，将这些命名实体用它们对应的类型名替代，并且我们用一种数据结构将命名实体序号、命名实体和命名实体类型 作为一条记录保存下来，用于后面提取精确答案。在使用答案模式前，我们将属性标签用相应的命名实体类型的标签代替，并将这种映射关系保存下来，如果文档片段与某

一答案模式匹配，我们利用这种映射关系由属性名找到对应的命名实体类型名，在利用命名实体类型名从数据结构的记录中找到对应的命名实体，该命名实体就是精确答案。例如下面的文档片段“John Smith was born in 1986 in California”，我们在使用一般化的答案模式匹配前，对该文档片段进行处理，处理后为“<T> was born <NEdate> <NElocation>”，并将命名实体序号、命名实体和其对应类型作为一条记录保存，该例子中产生了两条记录。我们发现，处理后的文档片段和上面例子中的答案模式匹配，属性对应的命名实体类型为 NElocation，我们利用命名实体类型 NElocation 在第二条记录中找到对应的命名实体 California，最后将该命名实体加入候选答案列表。

答案模式评估。利用这种模式学习的方法，我们可以产生大量的答案模式，AnswerSeeker 要对这些答案模式进行评估，去掉那些太具体或不可靠的答案模式。在该阶段我们仍使用 TREC9 会议的 700 个问题-答案对作为评估数据源，首先利用问题模式对每个问题进行解释，然后利用这些问题的解释生成查询交给搜索引擎，然后我们再利用答案模式从搜索引擎返回的文档片段中提取精确答案。对于每一个答案模式 a，我们的算法都会记录该答案模式提取的正确答案的数目 correct_a 和错误的答案数目 incorrect_a，并且对于每个属性，我们的算法都会记录搜索引擎返回的文档片段数 snippets_p。这三个值用于计算下面两个答案模式的度量：

$$Confidence\_a = \frac{correct\_a}{correct\_a + incorrect\_a} \quad (2)$$

$$Support\_a = \frac{correct\_a}{snippets\_p} \quad (3)$$

我们的答案评估算法将这两个度量与门限值做比较。如果一个答案的置信度 Confidence_a 低于门限值，那么我们认为该答案模式不可靠，将其丢掉。如果一个答案模式的支持率 Support_a 低于门限值时，我们认为这样的答案模式太具体，也将其丢掉。此外，答案模式的置信度还用于判断一个答案(见第 5.2 节答案提取)。

6 实验

我们使用 TREC11[9]的 200 个问题在 AnswerSeeker 上进行了实验。对于每个问题，AnswerSeeker 返回一个包含 5 个答案的列表，在 5 个答案中，只要有一个是正确的，那么我们就认为得到了正确答案。下面表 3 总结了 AnswerSeeker 的测试结果。我们的系统最先尝试用知识诠释组件回答用户的问题，然后用模式匹配，如果这两种方法都失败的话，该系统启用后备技术，由于篇幅原因，本文并没有讲述后备技术相关的内容。

表 3　Performance of AnswerSeeker on TREC11 questions

	Questions	# Correct	Precision	Recall
Knowledge annotation	68	67	0.93	0.34
Pattern learning	104	86	0.83	0.50
Backup	27	18	0.67	0.13
All	200	171	0.85	0.97

在表中，Questions 列表示相应方法能够识别出的问题数量，#Correct 列表示相应方法能正确回答问题的数量，Precision 列表示相应放法和整个系统的正答率，Recall 列表示相应方法和整个系统对问题的覆盖率。Backup 方法在 Recall 列的值很低，并不真的代表该方法的问题识别率很低，因为我们是把前两种方法不能识别的问题才交给该方法来处理。

实验结果说明用知识诠释的方法获得的答案的准确率相当高，因为我们在该方法中只选用了 7 个网站作为知识源，所以问题的识别率比较低，只要增加知识源的数量，我们有理由相信问题覆盖率会大大增加。模式学习的方法获得答案的准确率较其他方法而言也很高，但问题的覆盖率不是很高，这是因为我们的答案模式是从 700 个问题中学习得到的，问题模式及属性也是从 700 个问题中人工分析得到的，如果增加训练集的问题数量，模式学习方法获得答案的准确率和问题覆盖率都会相应增加。整合多种技术的 AnswerSeeker 系统即获得很高的准确率，同时也获得了很高的问题覆盖率。所以，在同一系统中集成多种技术是问答系统获得高准确率和高覆盖率的不错选择。

7 结论

在本文中我们介绍了一个基于互联网挖掘的智能

问答系统 AnswerSeeker，它是一个模块化的框架，允许在同一系统中很容易地集成多种问答方法和技术。此外，它还是一个包含了问题处理、查询生成、知识检索、答案选择和提取等组件的组件库，我们可以很容易地利用这些组件动态地组建一个问答系统。如果一位开发者想到了一个更好的答案提取的方法，那么他只要将他的算法实现为一个答案过滤器集成到答案选择和提取模块就可以了，而没必要考虑查询生成和信息检索这些模块。

现在的系统内核是为英语量身定做的，我们通过一个语言转换器，用户可以使用任意一种他喜欢的语言提问，我们的系统会记下用户使用的语言，并将答案用该语言返回给用户。此外，我们的系统内核也是可以应用在其他语言环境中的，因为我们在设计和开发该系统时将语言相关的代码和语言无关的代码相分离，并且将语言相关的代码封装为组件，把用于问题分析的问题模式和用于答案提取的答案模式定义在了资源文件中，而不是写在了代码里面。我们只要为其他语言开发那些语言相关的组件和相应的文本模式，该系统就可以支持多种语言了。

针对互联网的特征，在该系统中，我们使用了两种问答技术：知识挖掘和知识诠释。前者用于从搜索引擎返回的非结构化的文档片段中挖掘答案，而后者从特定的网站或网络服务等半结构化的知识源中寻找答案。在知识挖掘中，我们实现了模式学习的方法，AnswerSeeker 通过问题-答案对训练集自己学习答案模式，然后利用这些答案模式去提取精确答案。

实验证明，基于互联网 AnswerSeeker 智能问答系统，即获得了很高的答案准确率，同时也获得了很高的问题覆盖率。

参考文献

1 Prager J.Open-Domain Question Answering. America: World Scientific, Jan. 2007.

2 Nyberg E, Frederking R, Mitamura T, Bilotti M, Hannan K, Hiyakumoto L, Ko J, Lin F, Lita L, Pedro V, Schlaikjer A. JAVELIN I and II systems at TREC 2005. Proc. of the 14th Text REtrieval Conference. 2005.

3 Metamora M, Lin F, Shima H, Wang M, Ko J, Betteridge J, Bilotti M, Schlaikjer A, Nyberg E. JAVELIN III: Cross-lingual question answering from Japanese and Chinese documents. Proc. of the NTCIR-6. 2007.

4 Katz B, Felshin S, Yuret D, Ibrahim A, Lin J, Matron G, Jerome A, Temelkuran B. Omnibase: Uniform access to heterogeneous data for question answering. Proc. of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB 2002). 2002.

5 Katz B, Lin J, Felshin S. The START multi-media information system: Current technology and future directions. Proc. of the International Workshop on Multimedia Information Systems (MIS 2002). 2002.

6 Katz B. Annotating the World Wide Web using natural language. Proc. of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO'97). 1997.

7 Zhang D, Lee W. Web-Based pattern mining and matching approach to question answering. Proc. of the 11th Text Retrieval Conference. 2002.

8 Ellen M. Overview of the TREC-9 Question Answering Track. Proc. of the Ninth Text Retrieval Conference. 2001. 71－80.

9 Ellen M. Overview of the TREC 2002 question answering track. Proc. of the 11th Text Retrieval Conference. 2002.