基于 XML 的网页信息抽取

XML-Based Web Information Extraction

学生姓名：周津

学　　号：SA01010035

指导教师：朱明　（教授）

单位名称：自动化系

专业名称：模式识别与智能系统

中国科学技术大学

二零零四年五月

摘　要

随着互联网的迅猛发展，“信息过载”已经成为一个亟待解决的问题。为了使用户准确获取他想要的信息，信息抽取成为必要。从网页中抽取信息的程序称为 Wrapper。关键的任务是：Wrapper 的构造要尽可能快速，不需要过多人为地参与，并且，构造出的 Wrapper 要尽可能健壮，能适应网页的变化，同时，还要尽可能通用，与具体网站无关。

针对 Wrapper 生成问题，人们提出了各种各样的方法。这些方法的抽取模式语言基本上都是自己定制的，往往很简单、难以描述精确或者复杂的信息抽取模式。尽管通过人为标记的样本可以自动归纳出抽取规则，但这些抽取规则很难达到很高的精度、健壮性和通用性。

本文使用标准的 XML 技术来解决网页信息抽取问题。基于标准的 XSLT，可以利用它强大而且灵活的特性编写简单、健壮和通用的抽取规则。为了快速的构造抽取规则，我们开发了一个信息抽取平台。

除了手工编写抽取规则外，本文提出了新颖的方法自动归纳网页模板和记录模板，以及相应的抽取规则。网页模板可以用来抽取网页的主要内容。这对很多基于网页内容的工作很重要，比如网页信息检索，网页聚类与分类等等。记录模板可以用来抽取网页中的列表数据。另外，由于使用的是 XSLT，抽取模式可以很容易理解和修改。

最后，我们还开发了多网页信息抽取框架。实际的应用经常需要对多个网页进行抽取。基于本文所设计开发的 Web 信息抽取平台，可以很快的构建出健壮和通用的网页信息抽取 Wrapper。

关键词：信息抽取，互联网，XML

ABSTRACT

With the explosion of World Wide Web, “Information Overload” has become a serious problem. To help people accurately get the piece of information what he wants from the Web, information extraction from web pages is necessary. The program that performs this task is called wrapper. The key requirements are that a wrapper can be constructed rapidly, without much human intervention, and the wrapper should be robust, adaptable to the change of web page, moreover, the wrapper should be as general as possible, that is, it is independent on particular web site.

Many approaches have been proposed to ease wrapper generation. Almost all of them use proprietary extraction languages. The languages are simple, hard to express accurate or complex extraction pattern. Although through labled examples, extraction rules can be induct automatically, they are not accurate, not robust or general.

We apply standard technologies of XML to web information extraction problem. With standard XSLT, we can exploit strong and flexible features of the language to construct simple, robust and general extraction rules. We have developed a platform to ease wrapper construction.

In addition to manually writing extraction rules, we proposed novel approaches to automatically induct page template and record template, including extraction rules for each template. Page template can be used to extract main content of a web page, which is critical to many works on page content such as web information retrieval, web document clustering and classification and etc. Record template can be used to extract list data in web page. Because the extractin rules are in XSLT, they can be easily understood and revise.

At last, we developed mutli-page information extraction framework. Practial applications often need multi-page information extraction. With our platform, we can devolop robust and general wrapper rapidly.

KEY WORDS：Information Extraction, WWW, XML

目录

1. 概　述

1.1 引言

1.1.1 背景

毫无疑问，互联网已经成为最为流行的信息发布媒介。互联网使得人们无论是发布还是阅读信息都变得极为方便。然而，随着互联网信息爆炸性的增长，人们想要获取一条自己想要的信息却变得像大海捞针一般困难。**如何有效、快速的搜索所需信息，成为亟待解决的问题**。

在这种背景下，搜索引擎出现了。它帮助人们通过给定的关键词来获取相关的页面。然而，搜索引擎只是部分的缓解了信息搜索的问题，结果并不能令人满意。不足之处表现在三个方面：

1. 只是给出了相关页面的链接，用户还是需要手工浏览网页才能找到相关信息。
2. 结果不准确。大量的搜索结果都是用户不想要的。
3. 检索模式简单。无法提供类似 SQL 这样强大的查询语言。由于无法定制精确的查询，想要获取精确的结果是不可能的。

最理想的情景是：互联网作为一个信息源能像数据库一样被查询。然而，互联网上文本信息的格式是半结构化的 HTML，它是无法被机器直接处理的。因此，一种想法是将网页中的信息抽取出来并存放到数据库中[FLM98]。这样，用户就可以利用数据库的各种特性来查询数据了。

1.1.2 网页信息抽取

信息抽取的目标是将文本中的信息抽取出来并表示为结构化、自描述的数据结构。从而将难以操纵的文本数据转化为容易处理和分析的结构化数据。

传统的信息抽取是针对纯文本，主要使用自然语言理解的技术。但由于纯文本没有任何文本之外可利用的信息，这项工作极为困难，进展也很缓慢。随着互联网的出现，Web 文档的信息抽取逐渐成为亟待解决的问题。一个 Web 文档就是一个网页，网页与纯文本的结构差别很大，主要表现为网页中存在大量的标记，这些标记将网页要显示的文本内容分隔开来。大量的标记为网页信息抽取提供了更多可利用的信息，从而可以开发各种不同于传统信息抽取的方法对网页进行信息抽取。

标记为文档引入了结构信息。根据标记可以将一个文档表示为一棵树的结构。但是，网页并不是结构化的。网页所使用的的语言是 HTML[HTML]，HTML 被设计是用来方便数据的表现而不是数据的处理。标记大部分都是用于显示的，并不能描述文本的含义。因而，网页只能算是半结构化的文档。信息抽取也成为必要。

从网页中抽取信息的程序成为 **Wrapper**。

网页的特殊性带来了新的挑战。主要表现为网页的易变性。一个网页由 URL 唯一标识，但是网页内容变化很频繁。不仅如此，网页的结构也可能变化。这使得 Wrapper 很容易失效，如何构造**健壮的** Wrapper 成为一个关键的问题。

1.1.3 XML

XML（eXtensible Markup Language，可扩展置标语言）[XML]是由 W3C（World Wide Web

Consortium，互联网联合组织）于 1998 年 2 月发布的一种标准，同 HTML 一样是 SGML（Standard Generalized Markup Language，标准通用置标语言）的一个简化子集。XML 将 SGML 的灵活性和强大功能与已经能够被广泛采用的 HTML 结合起来。

与 HTML 不同的是，XML 不关心数据的显示，而只关心数据的描述。一个 XML 文档是完全结构化的。这使得它非常适合数据描述、数据交换和数据互操作。

XML 自推出以来就开始迅猛的发展和被广泛的应用。各种围绕 XML 的相关标准和工具不断被开发出来。这使得基于 XML 的应用开发更加高效，更加健壮，更加通用，更加容易维护。

XML 为信息抽取也带来了新的机遇。HTML 可以看成是 XML 的子集，因此，完全可以利用各种 XML 相关的标准和工具操纵 HTML 文档。我们开发了基于 XML 的网页信息抽取平台，而且还在平台基础之上开发了几个应用。实际证明，基于 XML 的网页信息抽取可以达到简单、高效、健壮、通用和易维护的目标。

1.2 本文的工作

本文主要进行了以下几项工作：

1. 开发了网页信息抽取平台，包括一个帮助构造抽取模式的用户图形界面。使用本文的平台和图形用户界面，开发一个网页信息抽取程序只需要几分钟。
2. 研究了几种健壮的抽取模式构造方法。
3. 开发了一个通用的链接组抽取模式。
4. 网页模板的自动归纳。相同网站一般存在外表相似的网页。这些相似的网页都是由同一个网页模板所生成。相似网页除了主要内容不同外，其它的部分完全一样。这些相同的部分都属于网页模板中的内容。归纳模板，获取网页主要内容对各种基于网页内容的分析处理（比如信息检索、网页分类与聚类）有着极大的意义。本文通过比较相似网页结构完全自动的归纳出网页模版并生成网页主要内容的抽取模式。
5. 记录模板的自动归纳。根据数据库查询结果所生成的页面往往包含多条相似的信息块，比如 google 的检索结果页面。每条记录都具有相似的外观和结构，因为它们都是由相同的记录模板所生成的。本文根据记录的相似性完全自动的归纳出一个网页的记录模板并生成相应的抽取模式。
6. 开发了多网页信息抽取框架。基于这个框架，本文还开发了一个实际的 pconline 产品信息抽取的应用。

1.3 本文的组织

第一章是概述，介绍本文的研究背景和贡献。第二章简要介绍了本文方法所涉及到的相关标准技术。这些技术将是支撑整个系统的基础。第三章介绍系统架构以及基于 XSLT 的健壮和通用的模式编写方法。第四章介绍两种类型的模板自动归纳方法。第五章介绍了多网页信息的抽取。在实际应用中，单独的从一个网页中抽取所有必要的信息往往是不可能的。因此，如何有效地进行多网页的信息抽取成为一个关键的问题。最后，是对本文所做工作的一个总结和对未来工作的一些展望。

2. 相关研究

传统构造 Wrapper 的方式是手工编码。这种方式费时费力，容易出错，需要专家完成，而且这种方式难以维护，如果站点结构一变，所有代码就得重新编写。为了更好的解决这个问题，人们提出了各种各样半自动或自动化的方法[LRST02]。

对各种方法进行分类的角度可以有多种。比如根据自动化程度，可以分为手工、半自动和全自动[Eik99]。根据方法的原理可以分为及其机器方法、自然语言理解方法、Ontology 方法、HTML 方法等等[LRST02]。还可以根据结果是否是 XML 的、是否可以抽取复杂数据结构这些角度考虑。本文将主要从方法的原理进行分类，同时着重从实用角度分析每种方法或者系统的各种特性。关于几个和本文方法类似的系统将会详细介绍和分析。

2.1 基于自然语言理解的方法

基于自然语言理解的方法采用了过滤、词性和词汇语义标识来建立短语和语句元素间的关联，通过给定的例子学习抽取规则。这些规则通过语法和语义上的约束来定位元素[LRST02]。主要有三种工具 RAPIER[CM99]、SRV[Fre00]和 WHISK[Sod99]。RAPIER 和 SRV 只能抽取单条记录，而 WHISK 可以抽取多条记录。

2.2 基于机器学习的方法

基于机器学习的方法的抽取规则是基于分隔符来定位要抽取的数据。也是通过人为标记的样本自动学习抽取规则。它和前面一种方法主要不同之处在于它们并不依赖于语言上的约束，而是描绘数据的隐式的格式特性[LRST02]。主要有三种工具 WIEN[Kus00]、SoftMealy[HD98]和 STALKER[MMK01]。WIEN 和 SoftMealy 必须依靠进挨着数据前的分隔符来定位数据，而且还不能抽取复杂格式的数据。STALKER 引入 ECT 树来表示复杂格式的数据。

2.3 基于 Ontology 的方法

基于 Ontology 的方法[ECJ+99]主要依赖一个完全的知识库。知识库定义了各个元素的抽取模式，还有它们之间的联系。在抽取之前，需要将包含数据的纪录块分隔开来，然后依次对每个记录块进行信息抽取。抽取模式没有使用依赖于特定文档的分隔符或者词性这样的自然语言理解技术，而是主要使用通用的词法模式，比如姓名的模式是“[A-Z][a-zA-Z]*\s+([A-Z]\.\s+)?”。这种方法不依赖于任何结构和表现形式。它使用 Ontology 来定位关键信息并使用这些元素构造对象。不过，这事先需要构造一个完整的 Ontology 库，而构造这样一个库要由专家花很长时间。而且，有时很多信息很难给出对应的 Ontology。

2.4 上面三种方法的讨论

基于自然语言理解的方法和基于机器学习的方法都需要人为给定样本来学习抽取规则。而给定样本很耗精力。这两种方法的抽取规则都与具体网页密切相关，因此，如果网页结构

改变了，得重新提供样本并生成抽取规则。

虽然 Ontology 方法很通用，但是构建通用的知识库并不是一件容易的事情。有时一个简单的任务并不需要很高的通用性。

另外，上面这些方法都是线性的处理 HTML 文档，通过字符串模式定位到关键信息，而完全忽略了 HTML 文档本身的结构和语法。而字符串模式有时很难保证定位的精确性。

2.5 基于 HTML 结构的方法

通过利用 HTML 结构的知识，基于 HTML 的方法获得了最大程度的自动化[LRST02]。当然，这种方法无法被用到非 HTML 文档中去。但既然本文的工作集中于网页信息的抽取，那么很大程度上可以忽略这个缺点。下面将介绍几个重要的基于 HTML 的工具。

2.5.1 W4F

W4F[SA99]包含一组自定义的语言用来描述网页获取规则，信息抽取规则以及到 Java 程序对象的转换规则。抽取规则还包含正则表达式来帮助从纯文本中抽取信息。就像它的名字，这个工具还包含一个图形用户界面来帮助用户生成抽取规则。其流程如图 2.1 所示：

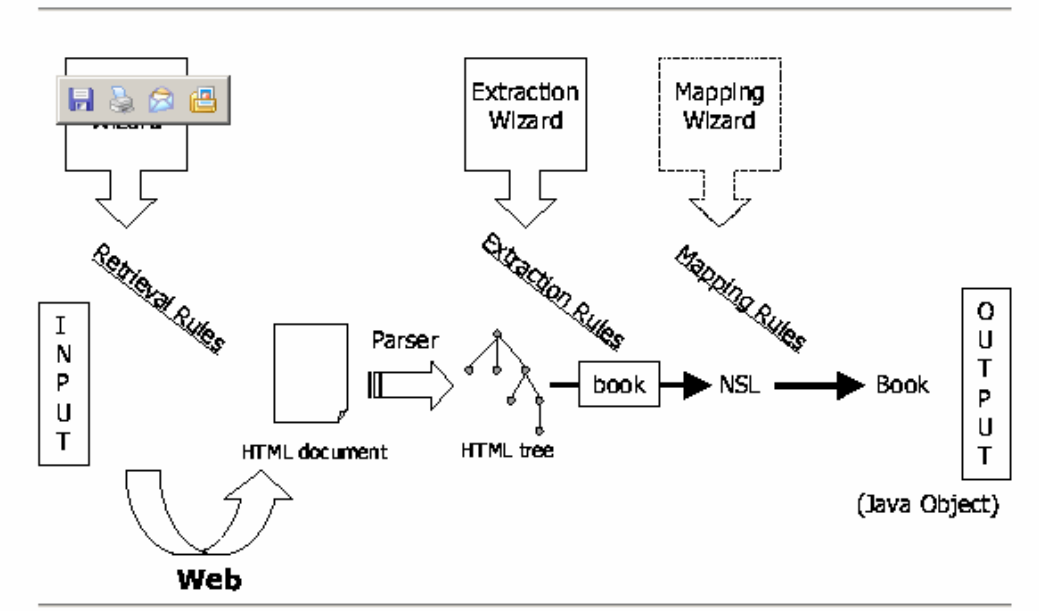
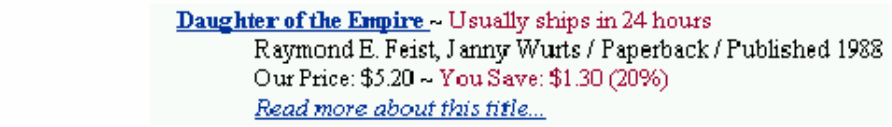


图 2.1 W4F 的流程

这里，本文将主要关注 W4F 的抽取规则，如图 2.2 所示。抽取规则使用了树路径和正则表达式。而这些都可以通过标准的 XSLT 达到。不仅如此，XSLT 还提供了更为丰富的数据定位方法。




```
EXTRACTION_RULES ::
books = html.body.table[2].tr[0].td[1].ul[0].li[2].dl[0].dt[*]
 ( .b[0].a[0].pcdata[0].txt                                    // title
#  .b[0].a[0].getAttr(href)                                    // url
# ->dd[0].pcdata[0].txt, match /Published (19[0-9]{2})/       // year
# ->dd[0].pcdata[0].txt, match /(.*?)\//, split /, /          // authors
# ->dd[0].pcdata[1].txt, match /(\$[^ ]+)/                    // price
  );
```

图 2.2 W4F 的抽取规则

2.5.2 XWrap

XWrap[LPH00]是一个半自动化的 Wrapper 生成器，如图 2.3。首先获取 URL 对应的网页的树结构。随后利用了 HTML 中一些特定标记（比如 HEAD 和 TABLE）以及它们被用作数据表现时的含义作为启发式。通过启发式，它会帮助自动寻找关键信息。并生成由 Java 代码写的 Wrapper。据说用户只要简单的点击几次就可以获得一个站点的 Wrapper。自动化程度应该算很高了。但是，实际生成的 Wrapper 效果并不理想。因为很多站点并不符合那些特定的启发式。而且，对于大部分定制的信息抽取任务，通过几次简单的点击和启发式的搜索并不能准确捕捉用户的需求，因而，尽管人为参与很少，但结果反而并不精确。另外，由于 Wrapper 是 Java 代码描述的，这使得 Wrapper 修改和维护起来都很困难。

XWrap 使用树路径定位要抽取的区域，一般是一个表。然后根据一个模板规则对抽取区域中的数据。模板规则语言自定义的语言，用 XML 描述。有一些简单的循环和提取指令。

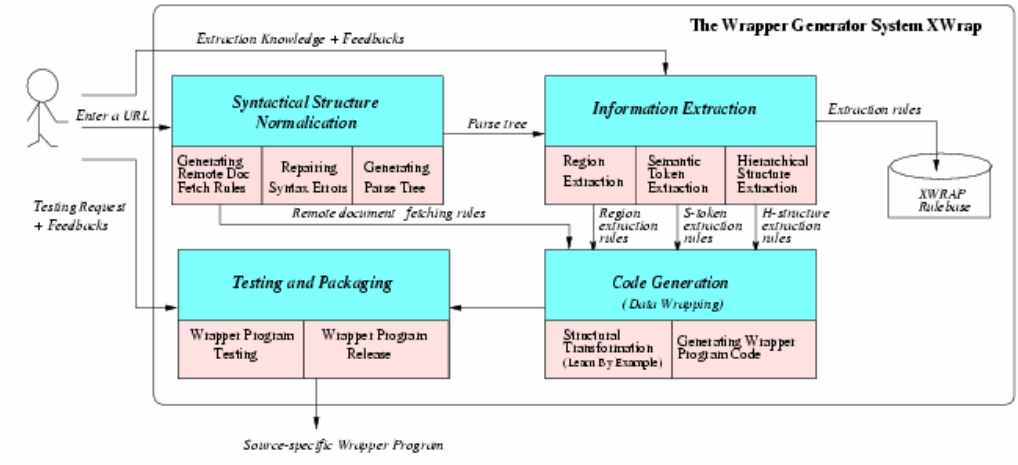


图 2.3 XWrap 的系统结构

2.5.3 ANDES

ANDES[MJ03]使用最新的标准的技术——XML 和 XSLT。通过 XML 和 XSLT，抽取规则可以很容易的被构造出来，而且很有效。如图 2.4 所示：

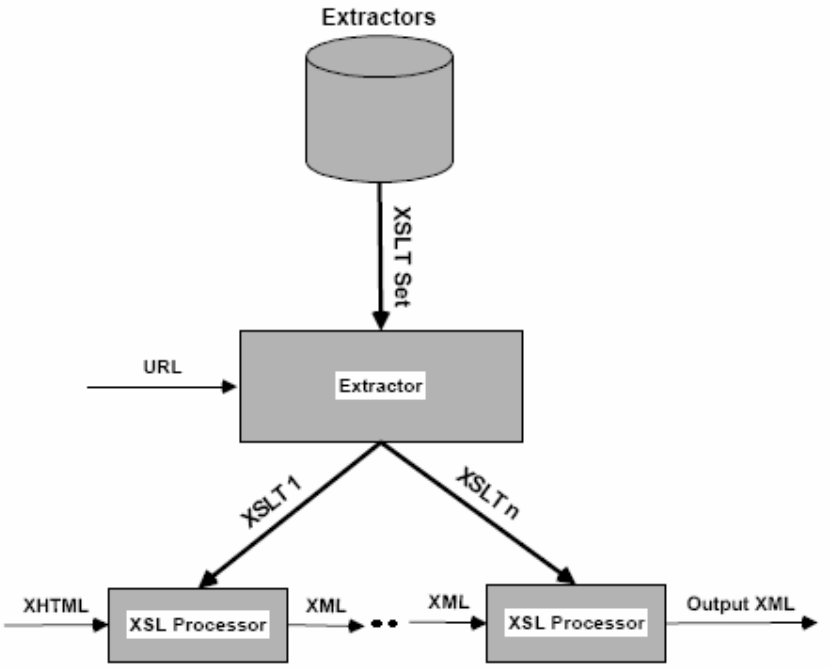


图 2.4 ANDES 的抽取框架

利用文本搜索和相对路径相结合的方法，可以使所产生的 Wrapper 更加健壮。定位模式如下：

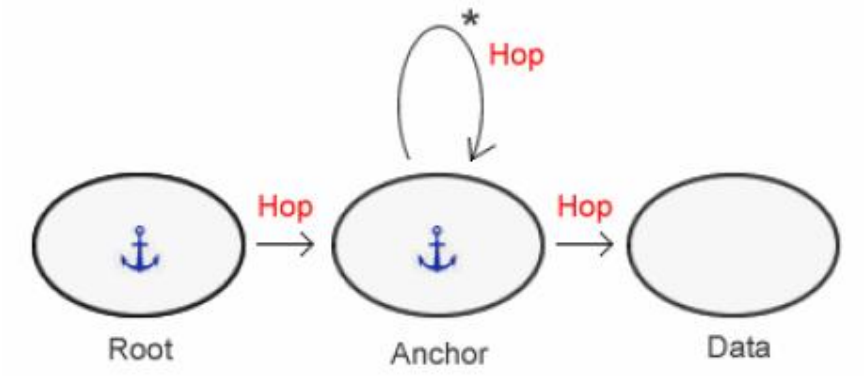


图 2.5 Anchor-Hop 定位模式

2.5.4 小结

W4F 和 XWrap 都设计了各自的抽取规则。这种抽取规则基于 HTML 的树结构。根据树结构可以精确的定位数据。因此它们都包含了图形用户界面以帮助生成抽取规则。用户使用图形用户界面圈定自己想要抽取的内容，系统可以自动生成对应的抽取规则。但是，简单的点击并不能描述复杂数据的抽取模式，因此自动生成的抽取规则可能需要重新改写。XWrap 生成 Java 代码的 Wrapper，这使得修改起来很困难。

自己设计的抽取模式语言一般都比较简单，与特定的系统绑定到一起。缺乏维护和更新，容易过时。ANDES 使用标准的 XML 和 XSLT 技术来进行信息抽取。XML 和 XSLT 是被广泛支持的标准语言，功能强大。利用标准技术不仅可以快速的构造抽取规则，而且还可以编写更加健壮的抽取规则。因此本文也采用 XSLT 作为抽取规则语言。

ANDES 仅仅提出了一种简单的构造健壮抽取规则的方法。本文不仅提供了一个帮助构造 XSLT 的图形用户界面，还提出了几种其他的构造健壮抽取规则的方法，同时，还开发了一个通用的独立与具体网站的链接组提取方法。

2.6 完全自动化的方法

完全自动化的方法不需要人为标记样本，也不需要人为编写抽取规则。这种方法根据网页的相似性结构自动找到网页中的数据并归纳出抽取规则。用户只需要最后对数据模式进行标记。

2.6.1 IEPAD

IEPAD[CL01]通过构造 PAT 树来发现频繁出现的连续标记来定位和抽取数据。这种方法只适用于有限的数据模式：不包含嵌套结构的记录。比如它们的实验对象：搜索引擎。由于并不是所有重复出现的模式都包含有用的数据，IEPAD 使用了各种启发式来进行标识。

2.6.2 RoadRunner

RoadRunner[CM03]通过比较相似的网页归纳出网页的模版，模版使用正则表达式描述。归纳出来的模版就是抽取规则。这种方法主要适用于由数据库查询生成的页面，这种页面包含有类型相同的数据，网页是由同一个模板所生成。

RoadRunner 将信息抽取模式生成等同为正则表达式归纳问题。而正则表达式归纳是到目前都解决得不好的一个问题。RoadRunner 做了很多假定，假定标记都是模板的一部分，假定不存在或模式的数据，假定数据是上下文无关的，而这些假定往往是不成立的。除此之外，RoadRunner 为了归纳出正则表达式，使用了大量的复杂的启发式搜索算法。这使得 RoadRunner 的算法特别敏感，归纳时很容易失败。

2.6.3 小结

自动化的方法通过网页的相似性来发现网页中的数据并归纳相应的抽取规则。这种方法是完全自动的。但是，另一方面，这种方法并不适用于定制的信息抽取任务，比如抽取某个网页中某一块的信息，因为很多任务并不是简单的获取所有网页中变化的数据或者频繁出现的结构中的数据。

2.7 方法总结和本文的工作

为了方便有效的进行信息抽取，各种各样的方法被提出来。信息抽取的核心实际上是抽取模式。各种方法都致力于抽取模式的自动构造。抽取模式并不统一，几乎每种系统都有各自的一套抽取语言。大部分的抽取语言都难以用于手工编写模式，因此需要人为标记样本来学习抽取模式。基于 HTML 结构的抽取语言是基于 HTML 文档的树结构，通过树路径定位数据简单、直观、而且精确，因此适用于人为定制抽取规则。而且基于 HTML 的抽取语言可以抽取复杂的数据结构，这对于实际的应用是很有效的。XSLT 本来是用于转换 XML 文

档的，它定义了强大而且灵活的一套数据定位语言（XPath）以及抽取指令，可以进行各种复杂的抽取和变换。由于 HTML 可以看作 XML 的子集，因此完全可以利用 XSLT 作为抽取语言。使用 XSLT 不仅具有强大灵活的语法，易于理解和修改的结构，而且还具有众多的工具支持。因此使用 XSLT 是最为理想和实用的方法。

在 HTML 树结构中寻找数据以及获取路径并不是一件容易的事，本文开发了一个图形用户界面方便这两步工作。使这两项工作都极为简单。

网页的易变性给信息抽取带来新的问题。Ontology 的方法最为健壮和通用，因为与网页结构无关。但是代价是很多情况下很难找出通用的模式，因此并不实用。实际上，使用 XSLT 也能构造健壮而且通用的抽取规则，本文就这方面进行了研究。

各种各样的方法都希望最大最大程度减少人为的参与，完全自动化的方法在这方面达到了极致。这种方法根据相似的网页结构自动寻找数据并归纳抽取模式。用户所要做的工作仅仅是对结果进行标记。虽然这种方法达到了最大程度的自动化，但是，由于缺乏人为的参与，这种方法无法准确了解用户的需求，因而生成的抽取规则往往并不是有用的，一般需要一定程度的修改以满足实际的应用要求。

本文也进行了这方面的研究。本文将网页模板和记录模板分离开来。网页模板用于分离网页框架，获取网页主要内容。这种任务不要求复杂数据的精确定位，归纳出来的模版不需要修改和对数据进行标记，而只要求不需要任何人为参与的将冗余信息去除掉。自动归纳网页模板针对这种任务是有效的。相反，记录模板归纳的目标是精确定位每一项数据。实际上，如前面所说的，这种方法对实际应用来说不一定有效，因为算法根本不知道用户具体要什么？只是假定频繁出现的模式中的数据就是用户想要的，实际上并非如此。因此，生成的抽取规则还需要一定的修改才能满足实际需要。本文的抽取规则是基于 XSLT 的，因此，修改和维护都很方便。

最后，到目前为止所有的方法都是针对单网页信息抽取的，而实际应用中往往需要抽取多个网页中的数据。本文进行了这方面的研究，并开发了一个通用的多网页信息抽取框架。使用本文的平台，已经实现简单、快速、有效构造健壮的信息抽取。

3. 相关标准

本文的信息抽取方法涉及到很多标准技术，这些标准技术是本文的工作基础。本章将介绍这些相关的标准技术。由于本文处理的对象是一个 HTML 文档，因此，首先介绍 HTML。

3.1 HTML（Hyper Text Markup Language）

HTML[HTML]是一个用于创建网页的标记语言。它是一个由 W3C 组织创建并维护的 Internet 标准。最新的版本是 HTML4.01。

HTML 通常表现为连接到互联网的计算机中的文本文件。这些文件中包含很多标记。这些标记是告诉浏览器如何表现或处理文本内容的指令。

考虑下面的例子 3.1：

```
<html>
<head><title>Title of page</title></head>
<body>
This is my first homepage.
<b>This text is bold</b>
</body>
</html>
```

例子 3.1 一个简单的 HTML 源文档

使用 IE 浏览器察看这个文件，结果如图 2.1 所示：

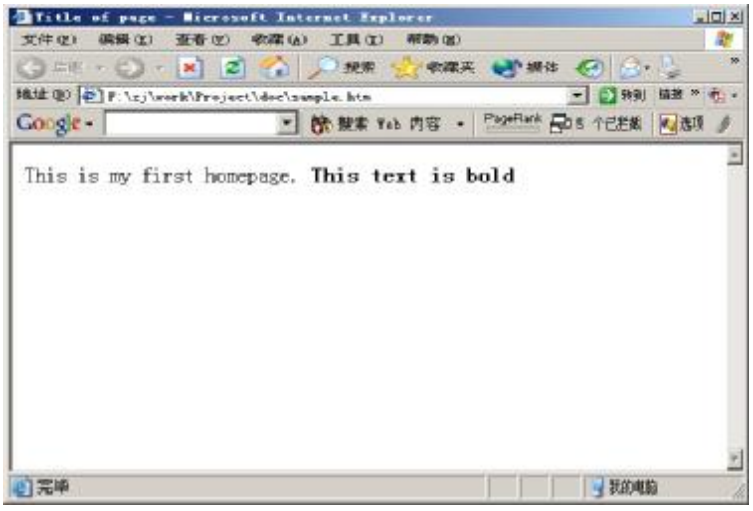


图 3.1 简单的 HTML 页面

在上面的文件中，被两个尖括号括起来的内容就是**标记**，比如 html, title, b 等等。一般来说，每个标记都由开始标记和结束标记组成。不加斜杠的标记为**开始标记**，比如。加斜杠的标记为**结束标记**，比如。开始标记和结束标记之间的内容为这个标记的作用内容。

比如，b 标记中的内容显示为粗体。Title 标记中的内容为文档标题。

尽管 HTML 在展示数据方面十分成功，但是，对于机器来说，一个 HTML 文档是很难理解的。因为，HTML 着重于数据的表现而不是数据的描述。比如，根据<b><i>这样的标记根本无法获得它们所包含文本究竟是什么内容。着重于数据描述的一个新的语言是 XML。

3.2 XML

XML[XML]的全名是可扩展标记语言（e**X**tensible **M**arkup **L**anguage）。它允许开发人员制定自己的标记，从而使得文档具备自描述性。

3.2.1 XML 的产生

首先，让我们来了解一下可扩展标记语言 XML 的发展简史。

XML 有两个先驱--SGML 和 HTML，这两个语言都是非常成功的标记语言，但是它们都在某些方面存在着与生俱来的缺陷。SGML（Standard Generalized Markup Language）的全称是标准通用标记语言，它为语法标记提供了异常强大的工具，同时具有极好的扩展性，因此在分类和索引数据中非常有用。但是，SGML 非常复杂，并且价格昂贵，几个主要的浏览器厂商都明确拒绝支持 SGML，使 SGML 在网上传播遇到了很大障碍。

相反，超文本标记语言 HTML（HyperText Markup Language）免费、简单，在世界范围内得到了广泛的应用。它侧重于主页表现形式的描述，大大丰富了主页的视觉、听觉效果，为推动 WWW 的蓬勃发展、推动信息和知识的网上交流发挥了不可取代的作用。可是，HTML 也有如下几个致命的弱点，这些弱点逐渐成为 HTML 继续发展应用的障碍。

- HTML 是专门为描述主页的表现形式而设计的，它疏于对信息语义及其内部结构的描述，不能适应日益增多的信息检索要求和存档要求。
- HTML 对表现形式的描述能力实际上也还非常不够，它无法描述矢量图形、科技符号和一些其他的特殊显示效果。
- HTML 标记集变得日益臃肿，而其松散的语法要求使得文档结构混乱而缺乏条理，导致浏览器的设计越来越复杂，降低了浏览的时间效率与空间效率。

正因为如此，1996 年人们开始致力于描述一个标记语言，它既具有 SGML 的强大功能和可扩展性，同时又具有 HTML 的简单性。XML 就是这样诞生的。

正象 SGML 和 HTML 一样，可扩展标记语言 XML 也是一种标记语言，它通过在数据中加入附加信息的方式来描述结构化数据。不过，XML 并非象 HTML 那样，只提供一组事先已经定义好的标记。准确地说，它是一种元标记语言，允许程序开发人员根据它所提供的规则，制定各种各样的标记语言。

3.2.2 XML 语法

XML 的语法规则很简单而且非常严格。正因如此，开发读取和操纵 XML 的软件很简单。

3.2.2.1 XML 文档

下面是一个简单的 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

例子 3.2 一个简单的 XML 文档

文档的第一行是一个声明，它定义了 XML 的版本和文档所使用的字符编码。第二行描述了文档的根元素，接下来四行是根元素的子元素，最后一行是根元素的结束标记。

从这个例子可以看出，XML 文档是自描述的。我们很容易理解每个元素的意义。这是因为每个元素都有标记信息来描述元素的内容。

XML 是一个元标记语言。也就是说，允许开发人员定义自己的标记。如上面的文档所示，所有的标记都是自定义的。

XML 的语法很严格。形式良好(well formed)限制是 XML 文档的最低要求。不满足形式良好限制的文档就不是 XML 文档。解析器将无法读取这种文档。简单的说，XML 文档应有而且只有一个根元素，所有开始标记要有相应的结束标记，元素要正确的嵌套，所有属性值要放到引号中等等。

除了把 XML 文档看成符合一定规则的字符序列外，还可以用另一种更有意义的方法考察 XML 文档，特别是在编写处理 XML 文档的程序时要用到这种方法。XML 文档是一棵树（tree），有一个根节点，包含不同的子节点，其中有些子节点自己又有子节点，而有些则是叶节点，没有子节点。

XML 树中大致可以分出 5 种不同节点：

根节点

也称为文档节点，是一个抽象节点，包含整个 XML 文档，其子节点包括注释语句、处理指令和文档的根元素。

元素

XML 元素，具有名称、一组属性、一组范围名称空间和一列子节点。

文本节点

两个标记（或者任何其它非文本节点）之间的字符数据。

注释语句

<!—This needs to be fixed. -- >之类的 XML 注释语句。注释语句的内容是其数据。注释语句没有任何子节点。

处理指令

<?xml-stylesheet type=”text/css” href=”order.css”?>之类的处理指令。处理指令具有目标和值，没有任何子节点。

一个 XML 文档的大部分内容都是元素和文本节点。文本节点是文档的数据部分。元素节点则是 XML 文档的结构部分。一般来说应用程序主要就是处理元素节点和文本节点。下面将详细介绍元素。

3.2.3 元素（Element）与标记（Tag）

XML 的基本单元是元素。从语法上来说，一个起始标记，对应的结束标记，以及标记之间的内容就是一个元素。例如<To>Tove</To>或者<Note>…</Note>。从逻辑上来说，每个元素有 4 个关键部分：

1. 名称
2. 元素属性
3. 元素名称空间
4. 元素内容

名称就是标记的名字，比如 To, Note。元素属性是标记中所含的名称/值对，元素属性在下一节详细介绍。名称空间不属于本文的考虑范围。元素内容是标记之间的数据，可以包含文本或者一个或多个子元素。这样，这些元素就构成了一个树的结构。实际上，XML 文档从逻辑上就是一个树模型。正因为如此，对 XML 文档进行转换以及提取数据都变得很容易。

3.2.4 属性（Attribute）

XML 元素可以在开始标记中包含属性，就像 HTML 一样。属性提供了元素的附加信息。考虑下面的例子：

```
<person sex=”female”>…</person>
```

其中，person 元素具有一个属性 sex，对应得值为 female。一个元素可以包含多个属性，这些属性是没有顺序的。

3.2.5 XML 验证（Validation）

除了形式良好外，XML 还可以加上有效性限制。合法的数据不一定有效。比如 3 米的身高。可以使用模式语言（schema language）来对 XML 文档进行验证。主要的模式语言有 DTD（Document Type Definition）和模式（Schema）。

3.2.5.1 DTD

DTD 可以定义文档包含什么元素、每个元素可以包含什么、按什么顺序以及每个元素有哪些属性。

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

例子 3.3 例子 3.2 的 DTD

3.2.5.2 模式

W3C XML 模式[XML SCHEMA]语言克服了 DTD 的几个局限。首先，模式用 XML 实例文档愈发编写，使用标记、元素和属性。第二，模式完全支持名称空间。第三，模式可以

指定元素的数据类型（如 integer 和 date），验证文档时不仅可以根据元素结构，而且可以根据元素内容。

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.w3schools.com"
    xmlns="http://www.w3schools.com"
    elementFormDefault="qualified">

<xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

例子 3.4 例子 3.2 的 Schema

3.2.6 样式单

XML 关于文档浏览的基本思想是将数据与数据的显示分别定义，XML 文档本身不涉及各种数据的具体显示方式，文档的显示实际上是通过一个外部样式表，又称为样式单来描述的。

样式单(StyleSheet)是一种描述结构文档表现方式的文档，它既可以描述这些文档如何在屏幕上显示，也可以描述它们的打印效果，甚至声音效果。与传统使用的<font>等标记相比，样式单有许多突出的优点：

- Ø 表达效果丰富。
- Ø 文档体积小。
- Ø 便于信息检索。
- Ø 可读性好。

迄今为止，W3C 已经给出了两种样式单语言的推荐标准，一种是层叠样式单 CSS（Cascading StyleSheets），另一种是可扩展样式单语言 XSL（eXtensible Stylesheet Language）。

内容与形式相分离使 XML 文档更偏重于数据本身，而不受显示方式的细枝末节的影响。而且通过定义不同的样式单可以使相同的数据呈现出不同的显示外观，实现 XML 数据的可重用性。

CSS 主要用于文档的显示。XSL 功能更加强大，它包含两个互为补充的组件：XSL 转换（XSLT Transformation）与 XSL 格式化对象（XSL-FO，XSL Formatting Object）。XSLT

可以将一个 XML 文档转换为另外一个 XML 文档，我们使用他来作为抽取规则描述语言。下一节将简要介绍 XSLT。在 3.6 节将详细介绍。

3.2.6.1 XSLT（XSL Transformation）

XSLT 是图灵完整的功能性编成语言，专门用来描述从一种 XML 格式向另一种 XML 格式的转换。XSLT 处理器读取 XML 文档和 XSLT 样式单，并根据 XSLT 样式单中的指令转换文档，然后输出转换的文档。XSLT 用于 XML-XML 转换，但也可以从 XML 转换到 HTML，从 XML 转换到普通文本或从 XML 转换到任何其他文本格式，例如 TeX 与 troff。

3.2.7 XML 带来的好处

3.2.7.1 更有意义的搜索

数据可被 XML 唯一的标识。没有 XML，搜索软件必须了解每个数据库是如何构建的。这实际上是不可能的，因为每个数据库描述数据都是不同的。有了 XML，书就可以很容易以标准的方式按照作者、标题、ISBN 序号或其他的标准分类。搜索书就变得十分方便。

3.2.7.2 开发灵活的 Web 应用软件

数据一旦建立，XML 能被发送到其他应用软件、对象或者中间层服务器做进一步地处理。或者它可以发送到桌面用浏览器浏览。XML 和 HTML、脚本、公共对象模式一起为灵活的三层 Web 应用软件的开发提供了所需的技术。

3.2.7.3 不同来源数据的集成

现在搜索多样的不兼容的数据库实际上是不可能的。XML 能够使不同来源的结构化的数据很容易的结合在一起。软件代理商可以在中间层的服务器上对从后端数据库和其他应用处来的数据进行集成。然后，数据就能被发送到客户或其他服务器做进一步的集合、处理和分发。

3.2.7.4 多种应用得到的数据

XML 的扩展性和灵活性允许它描述不同种类应用软件中的数据，从描述搜集的 Web 页到数据记录。同时，由于基于 XML 的数据是自我描述的，数据不需要有内部描述就能被交换和处理。

3.2.7.5 本地计算和处理

XML 格式的数据发送给客户后，客户可以用应用软件解析数据并对数据进行编辑和处理。使用者可以用不同的方法处理数据，而不仅仅是显示它。XML 文档对象模式(DOM)允许用脚本或其他编程语言处理数据。数据计算不需要回到服务器就能进行。分离使用者观看数据的界面，使用简单灵活开放的格式，可以给 Web 创建功能强大的应用软件，这些软件原来只能建立在高端数据库上。

3.2.7.6 数据的多样显示

数据发到桌面后，能够用多种方式显示。通过以简单开放扩展的方式描述结果化的数据，XML 补充了 HTML，被广泛的用来描述使用者界面。HTML 描述数据的外观，而 XML 描述数据本身。由于数据显示与内容分开，XML 定义的数据允许指定不同的显示方式，使数据更合理地表现出来。本地的数据能够以客户配置、使用者选择或其他标准决定的方式动态地表现出来。CSS 和 XSL 为数据的显示提供了公布的机制。

3.2.7.7 粒状的更新

通过 XML，数据可以粒状的更新。每当一部分数据变化后，不需要重发整个结构化的数据。变化的元素必须从服务器发送给客户，变化的数据不需要刷新整个使用者的界面就能够显示出来。目前，只要一条数据变化了，整一页都必须重建。这严重限制了服务器的升级性能。XML 也允许加进其他数据，比如预测的温度。加入的信息能够流入存在的页面，不需要向浏览器再发送一个新的页面。

3.2.7.8 在 Web 上发布数据

由于 XML 是一个开放的基于文本的格式，它可以和 HTML 一样使用 HTTP 进行传送，不需要对现存的网络进行变化。

3.2.7.9 升级性

由于 XML 彻底把标识的概念同显示分开，处理者能够在结构化的数据中嵌套程序化的描述以表明如何显示数据。这是令人难以相信的强大的机制，使得客户计算机同使用者间的交互作用尽可能的减少了，同时减少了服务器的数据交换量和浏览器的响应时间。另外，XML 使个人的数据只能通过更新的布告发生变化，减少了服务器的工作量，大大增强了服务器的升级性能。

3.2.7.10 压缩性

XML 压缩性能很好，因为用于描述数据结构的标签可以重复使用。XML 数据是否要压缩要根据应用来定，还取决于服务器与客户间数据的传递量。XML 能够使用 HTTP1.1 中的压缩标准。

3.2.7.11 开放的标准

XML 基于的标准是为 Web 进行过优化的。微软和其他一些公司以及 W3C 中的工作组正致力于确保 XML 的互用性，以及为开发人员、处理人员和不同系统和浏览器的使用者提供支持，并进一步发展 XML 的标准。

3.2.7.12 XML 包括一套相关的标准：

- Ø 可扩展标记语言(XML)标准，这是 W3C 正式批准的。这意味着这个标准是稳定的，完全可用于 Web 和工具的开发。
- Ø XML 名域标准，这用来描述名域的句法，支持能识别名域的 XML 解析器。

- Ø 文档对象模型(DOM)标准，这为给结构化的数据编写脚本提供了标准，这样开发人员就能够同计算机在基于 XML 的数据上进行交互作用。
- Ø 样式单(XSL)标准，用于显示和转换 XML 文档。
- Ø 可扩展链接语言(XLL)标准和 XML 指针语言(Xpointer)标准是当前的工作草案。XLL 提供类似与 HTML 的链接，但功能更强大。例如，链接可以是多方向的，可以存在于对象上而不仅仅是页面上。IE5 内在不支持 XLL。

3.2.7.13 新的机会

作为表示结构化数据的一个工业标准，XML 为组织、软件开发者、Web 站点和终端使用者提供了许多有利条件。更多的纵向市场数据格式建立起来，被应用于关键市场诸如高级的数据库搜索、网上银行、医疗、法律事务、电子商务和其他领域，这使得机会更进一步地扩大。当站点更多地进行分发数据，而不仅仅是提供数据浏览时，特别的机会就产生了。

顾客服务正从电话和地理位置转移到 Web 站点上来，而且将会由于 XML 的强大功能受益更多。并且，由于大多数商业应用软件包括数据的处理和转移，如购买单、发货单、顾客信息、合同、图纸等等，XML 将会改革终端用户在 Internet 上的行为，许多商业应用将能实现。另外，使用基于 XML 的面向企业内部互连网的词汇库，Web 站点上的信息，无论是储存在文档中还是数据库中，可以被标识。这些词汇也能够对那些需要在顾客和供应商之间交换信息的中小型企业提供帮助。

一个重要的未开发的市场是开发使终端用户很容易建立自己的 Web 站点的工具，包括用来从数据库信息和存在的使用者界面中产生 XML 数据的工具。另外，标准模式可以开发用来描述数据，可以使用规划、图表、Excel 或其他电子数据表的功能。开发公布的用来描述从数据库中产生的 XML 的可视化工具是个很好的机会。观看 XML 数据的工具可以用 Visual Basic，Java 和 C++编写。

XML 需要强大的新工具用来在文档中显示丰富的复杂的 XML 数据，可以在分层的动态变化的数据上映射用户友好的显示层来实现这一目的。XML 数据的布局图包括数据透视表等。

Web 站点可以提供股票报价、新文章或实时的交易数据。通过制定信息老化的规则，信息超载可以避免。开发用户用来制定规则和服务器和客户软件用来实现规则的基于 XML 的工具是个巨大的机会。可以用脚本编写一个标准对象模式用来过滤进来的信息，检查储存的信息，创建输出的信息，进入数据库等等。

3.3 XHTML

从某种角度上来说，HTML 是 XML 的一个子集。HTML 是一种标记语言，有一套预定义的标记。但是，由于历史的原因，HTML 并不符合严格 XML 的语法。而且，由于各浏览器厂商的竞争，很多语法错误的 HTML 文档也都能被浏览器显示出来。这样，大量的 HTML 文档根本无法直接用于解析。

为了改变这种状态，W3C 推出了 HTML 的替代标准 XHTML[XHTML]。与 HTML 不同的是，XHTML 严格建立在 XML 基础之上，并且明确定义了合式的文档规则。

这意味着，可以像对待一般 XML 文档一样对待 XHTML 文档，这样，我们就可以利用各种强大的 XML 标准技术来操纵 XHTML 文档，从而可以大大简化应用程序的开发和维护。

本文就是先将 HTML 转化为 XHTML，然后使用 XSLT 作为抽取规则从 XHTML 文档

中抽取数据，转换为新的 XML 文档。

3.4 DOM（Document Object Model）

前面已经讲过，XML 文档可以看成一个树结构。文档对象模型（Document Object Model, DOM）[DOM]就是这样一个抽象数据结构，它将 XML 文档表示为由节点构成的树。org.w3c.dom 包中的不同接口可以表示元素、属性、已分析的字符数据、注释和处理指令，他们都是公用 Node 接口的子接口。Node 接口提供了在树中导航与处理的基本方法。

考虑下面的 XML 文档，

```
<?xml version="1.0"?>
<?order alpha ascending?>
<art xmlns="http://www.art.org/schemas/art">
  <period name="Renaisance">
    <artist>Leonardo da Vinci</artist>
    <artist>Michelangelo</artist>
    <artist>Donatello</artist>
  </period>
  <!-- insert period here -->
</art>
```

例子 3.5 一个简单的 XML 文档

它对应的 DOM 模型：

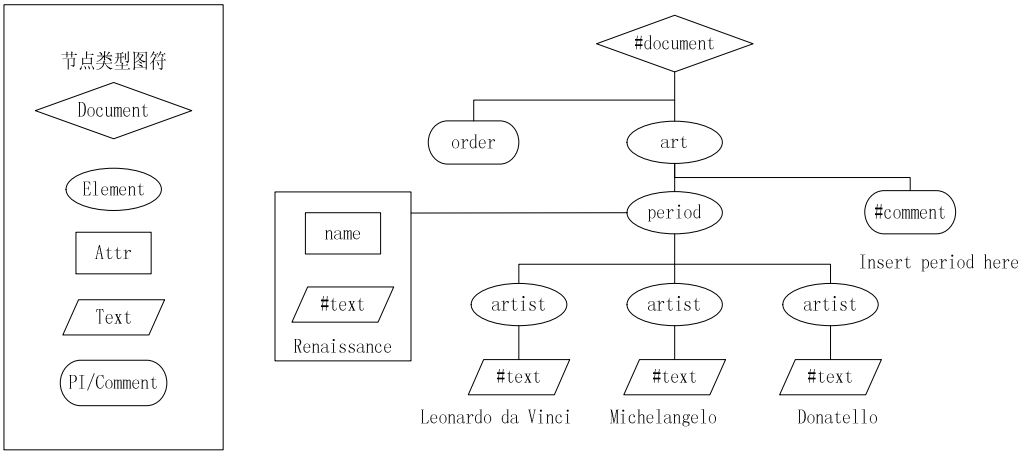


图 3.2 例子 3.5 对应的 DOM 树

3.5 XPath

XPath[XPATH]是第四代声明式语言，用于定位 XML 文档中的节点。XPath 位置路径指定需要文档中的哪些节点，而不指定寻找这些节点的算法。只要向方法传递一个 XPath 语句，XPath 引擎就会负责确定如何寻找满足这个表达式的所有节点。这比自己用 DOM，SAX 与 JDOM 编写详细的搜索与导航代码健壮得多。即使文档格式不符合预期，XPath 也通常能搜

索成功，例如，文本段落中间的说明可能破坏连续文本的 DOM 代码，但这不会影响 XPath。许多 XPath 表达式还可以对付更大的变化，如改变祖先元素的名称或名字空间，重排元素的子节点，以及在树层次中增加或减少整个层级。

大致来说，在 Java 程序中使用 XPath 与在 Java 程序中使用 SQL 差不多。要从数据库中取得信息，就要编写 SQL 语句表示需要什么信息，并且 JDBC 会取得所要的信息。你不知道也不关心 JDBC 如何与数据库通信。同样，在 XML 中，只要编写一个 XPath 表达式来表示需要什么信息，XPath 引擎会到 XML 文档中取得该信息，你不知道也不关心 XPath 搜索 XML 文档的算法。

XPath 是 XSLT 的主要构件。如果不了解 XPath 的话，将根本无法创建 XSLT 文档。

3.5.1 查询

可以把 XPath 看成与 SQL 相似的查询语言。但 XPath 不是从数据库中提取信息，而是从 XML 文档中提取信息。下面具一个例子加以说明。例子 3.6 是一个简单的 CD 目录文档。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

例子 3.6 一个 CD 目录的 XML 文档

这里用一些 XPath 表达式标识 XML 文档的特定部分：

- /catalog 选择根元素 catalog。
- /catalog/cd 选择 catalog 元素的所有 cd 元素。
- /catalog/cd[1] 选择 catalog 元素的第一个 cd 元素。
- /catalog/cd/price 选择 catalog 元素的所有 cd 元素的所有 price 元素。
- /catalog/cd[price>10.80] 选择第一个 cd 元素。
- /catalog/cd[1]/@country 选择第一个 cd 元素的 country 属性。
- /catalog/cd[@contry=”USA”] 选择第一个和第三个 cd 元素。

可以看出，这些表达式和目录路径和 url 很类似。这是因为他们都是针对一个树模型。但是 XPath 更加灵活和强大。

另外，上面的使用的都是缩写定位路径，缩写定位路径的介绍见 3.5.2.6。

3.5.2 定位路径（Location Path）

定位路径选择 XML 文档中的一组节点。每个定位路径包括一个或几个定位步（location step）。每个定位部有一个轴（axis）、一个节点测试（node test）和一个或几个谓词（predicate）。此外，每个定位步相对于特定的上下文节点（context node）求值。轴与节点测试用双冒号（::）分开，每个谓词放在方括号中。

一个定位步的语法如下所示：

```
axisname::nodetest[predicate]
```

比如：

```
child::price[price=9.90]
```

3.5.2.1 轴

轴选择了相对于当前节点的一组节点。比如

- **self** 节点本身
- **child** 上下文节点的所有子节点（属性不属于子节点）
- **parent** 父节点
- **attribute** 下下文节点的属性

还有很多种轴，但用的都不多，这里不详细介绍了。

3.5.2.2 节点测试

轴选择移动上下文节点的方向，而节点测试确定沿这个轴选择那种节点。可以通过名字或者类型来作为节点测试。节点测试如下：

- **name** 具有指定名称的任何元素或属性。
- ***** 沿着属性轴，星号匹配所有属性节点。沿着名称空间轴，星号匹配所有名称空间节点，沿着其他轴，星号匹配所有元素节点（注意，文本节点不属于元素节点）。
- **text()** 任何文本节点。
- **node()** 任何节点。包括文本节点或元素节点或其他任何节点。

还有几个其他节点测试，但不在这里介绍了。

下面是几个例子：

- child::cd 选择当前节点的所有 cd 元素。
- attribute::country 选择当前节点的 country 属性。
- child::* 选择当前节点的所有孩子元素。
- attribute::* 选择当前结点的所有属性。
- child::text() 选择当前节点的所有文本子节点。

3.5.2.3 谓词

每个定位步可以用0个或多个谓词进一步过滤节点集。谓词是方括号中的XPath表达式，对定位步选择的每个节点求值。如果谓词为 true，则节点保留在节点集中；如果谓词为 false，则节点不保留在节点集中。比如：

- child::cd[price=9.90] 选择价格为 9.90 的 cd 节点。
- child::cd[attribute::country=”USA”] 选择国家属性为 USA 的 cd 节点。
- child::cd[position() =1] 选择第一个 cd 子节点。

3.5.2.4 复合定位路径

斜杠符（/）将定位步组合成定位路径。第一步选择的节点集成为第二步的上下文节点集，第二步选择的节点集成为第三步的上下文节点集，等等。例如以第二个 cd 元素作为上下文节点：

parent::*/child::cd 选择父节点下的所有 cd 节点。

3.5.2.5 绝对定位路径

以斜杠（/）开头的定位路径是绝对路径，从文档根节点开始（而不是从根元素开始）。例如：

- /child::catalog/child::cd 选择 catalog 元素下的所有 cd 节点。
- /child::catlalog/child::cd[child::price>10.80] 选择 catalog 元素下价格大于 10.80 的 cd 节点。
- /child::catalog/child::cd[position()=1]/child::title/child::text() 选择第一个 cd 节点的 title 元素的文本子节点。
- / 选择文档的根节点。

3.5.2.6 缩写定位路径

XPath 定位路径可以使用下表所示的缩写，两者语义是相同的，但是缩写语法更容易输入。

表 XPath 的缩写语法

缩写语法	*扩展形式*
Name	child::Name
@Name	attribute:Name
//	/descendant-or-self::node()/
.	self::node()
..	parent:node()

例如：

- /catalog/cd[1]/price 选择第一个 cd 节点的 price 元素。

- /catalog/cd[@country=”USA”] 选择国家属性是 USA 的 cd 元素。

实际上，2.5.1 节中的所有例子使用的都是缩写定位路径。

3.5.3 表达式

并非所有 XPath 表达式都是定位路径。定位步谓词中方括号内的内容是更一般的 XPath 表达式。每个 XPath1.0 表达式返回下列四种类型之一：

- **string** 字符串。
- **number** 对应 double 类型。
- **boolean** 真或假
- **node-set** 节点集。没有顺序。

XPath 表达式的语法包含字符串型或者数字型的常量，以及操纵上面四种 XPath 数据类型的运算符和函数。

3.5.3.1 常量

XPath 可以定义字符串型或者数字型的常量。比如”red”，1.0。字符串常量包含在双引号中。尽管没有布尔型常量，但是可以用 true()和 false()函数代替。

3.5.3.2 运算符

XPath 对基本浮点运算提供下列运算符：

+	加法
-	减法
*	乘法
div	除法
mod	求余

XPath 还提供了比较与布尔逻辑运算符：

<	小于
>	大于
<=	小于等于
>=	大于等于
=	等于（不是赋值）
!=	不等于
or	布尔或
and	布尔于

3.5.3.3 函数

XPath 定义了一些重要函数，用于对四种基本 XPath 数据类型进行操作和返回这些 XPath

数据类型。这里简单介绍几个常用的函数：

number position()
返回上下文节点在上下文节点列表中的位置，第一个节点位置为 1 而不是 0。

boolean not(Boolean)
取非

boolean starts-with(string, string)
第一个字符串以第二个字符串开始时返回 true，否则返回 false。

boolean contains(string, string)
第一个字符串包含第二个字符串时返回 true，否则返回 false。

3.6 XSLT

XSLT[XSLT]是一个转换语言。XSLT 样式单描述一种样式的文档如何转换成另一种样式的文档。输入与输出文档使用 XPath 数据模型表示。XPath 表达式从输入文档中选择要进一步处理的节点。包含 XSLT 指令的模板作用于选择的节点，产生新节点，然后将其加入输出文档中。

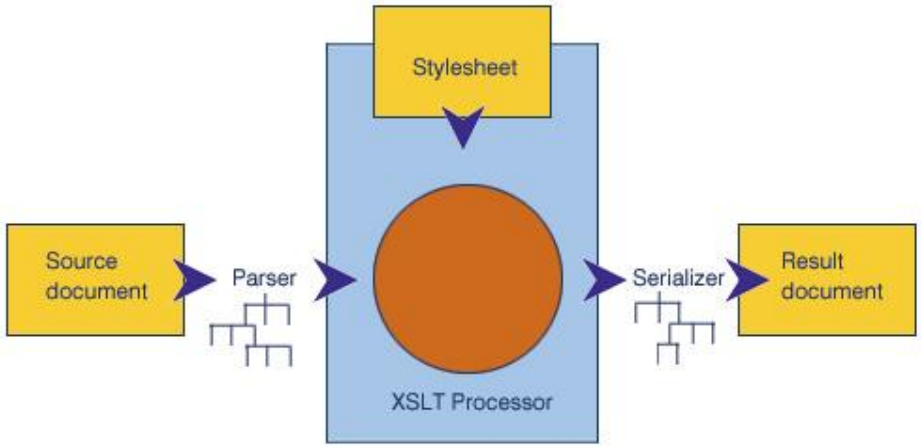


图 3.3 XSLT 输入和输出的树状结构

下面简单的介绍一个例子。源文档是例子 3.6，编写 XSLT 如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
  <titles>
  <xsl:for-each select="catalog/cd">
    <title><xsl:value-of select="title"/></title>
  </xsl:for-each>
  </titles>
</xsl:template>

</xsl:stylesheet>
```

例子 3.7 catalog.xsl

输出结果如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<titles>
<title>Empire Burlesque</title>
<title>Hide your heart</title>
<title>Greatest Hits</title>
</titles>
```

例子 3.8 转换结果

可以看出，结果文档提取了所有 cd 的 title 信息，并表示为新的 xml 文档。

从抽取的角度看，XSLT 就是抽取规则。由于 XSLT 是一个广泛使用标准，语言强大而且支持工具很多，因此本文使用 XSLT 作为抽取规则。

3.6.1 模板

XSLT 包含一组称为模板的规则。模板规则用 xsl:template 元素表示。每个<xsl:template>元素包含当一个特定节点匹配时所应用的规则。

如 catalog.xsl 所示，它包含了一个匹配文档根节点时所应用的模板规则。

3.6.2 取得节点值

也许最常用的 XSLT 指令是 xsl:value-of，其返回 XPath 表达式所选择对象的 XPath 字符串值。例如，元素值为它的所有子孙文本节点内容的合并。每个 xsl:value-of 元素有一个 select 属性，其数值包含 XPath 表达式。这个 XPath 表达式标识取值的对象。例如，下列 xsl:value-of 元素取得根 catalog 的值：

```
<xsl:value-of select=”/catalog” />
```

这个 xsl:value-of 元素取得树中根 price 元素的值

```
<xsl:value-of select=”/catalog/cd[2]/price”
```

下列 xsl:value-of 元素是用相对定位路径，它计算上下文节点第三个 cd 子节点的 title 子节点字符串值（上下文节点通常是所在模板匹配的节点）：

```
<xsl:value-of select=”cd[3]/title” />
```

XPath 元素可以计算任何一种 XPath 数据类型（number, boolean, string 与 node-set）的值，例如，下列表达式计算 e 乘以 pi 的值：

```
<xsl:value-of select=”2.71828 * 3.141592” />
```

事实上，可以在 select 属性中使用任何合法的 XPath 表达式。下列 xsl:value-of 元素将 price 元素乘以 0.85，并返回结果：

```
<xsl:value-of select=”0.85 * price” />
```

在模板中使用 xsl:value-of 时，上下文节点时模板匹配的节点，模板要实例化为这个节点。

3.6.3 应用模板

可能最重要的 XSLT 指令就是让处理器继续处理输入文档中的其他节点和实例化匹配模板。这个指令是 xsl:apply-templates 元素。其 select 属性包含一个 XPath 表达式，表示要采用模板的节点。当前匹配的节点是这个表达式的上下文节点。例如，下列模板匹配 catalog 元素，但不是输出固定响应，而是产生 titles 元素，其内容通过一一实例化每个 cd 子元素的模板来建立：

```
<xsl:template match=”catalog”>
  <titles>
    <xsl:apply-templates select=”child::cd” />
  </titles>
</xsl:template>
```

这个模板规则产生的完整输出取决于 cd 元素的模板规则：

```
<xsl:template match=”cd”>
  <title>
    <xsl:value-of select=”title” />
  </title>
</xsl:template>
```

3.6.4 默认模板规则

XSLT 默认模板规则在没有匹配节点的显示规则时使用。第一个默认模板规则用于根节点和元素节点，这只是对这个节点的子节点采用模板，而不产生任何输出：

```
<xsl:template match="*|/">
   <xsl:apply-templates />
</xsl:template>
```

这就使 XSLT 处理器默认可以从上向下遍历树，除非其他模板要求作不同的工作。任何显示模板都可以覆盖默认模板规则。

第二个默认规则作用于文本和属性节点，将这些节点的复制到输出树：

```
<xsl:template match="text()|@*">
   <xsl:value-of select="."/>
</xsl:template>
```

这两个规则结合到一起可以将元素或文档的文本内容复制到输出，并删除标记结构。当然，可以改变这些行为，用自己的模板规则覆盖内置模板规则。

3.6.5 循环

有时使用元素显示驱动选择过程式很方便的，就像传统的编程技术。再这种方法中，可以显示的循环处理一个节点集合，而不必像<xsl:apply-templates>那样实例化一个单独的模板。循环使用<xsl:for-each>元素。select 属性中的 XPath 表达式选择一个节点集，然后 for-each 内的模板对节点集中的每个节点应用一次，并将实例化结果加入到结果树中。

例如，循环抽出 cd 的 title。

```
<xsl:for-each select="catalog/cd">
  <title><xsl:value-of select="title"/></title>
</xsl:for-each>
```

循环指令中还具有排序功能，这里不详细介绍了。

3.6.6 选择

增加 XPath 谓词以匹配模式和选择表达式通常可以提供所要的 if-then 功能。如果不够，还可以利用 xsl:if 与 xsl:choose 元素。

xsl:if

xsl:if 指令是样式单可以确定是否做某个工作，其中包含一个模板。如果 test 属性中的 XPath 表达式求值为 true，则模板将被实例化并加入到结果树。如果 test 属性中的 XPath 表达式求值为 false，则不将模板实例化以及加入到结果树。如果 XPath 表达式求值为非布尔值，则用 boolean()函数转换成真假值，即 0 和 NaN 为 false，所有其他数字为 true；空字符串和空节点集为 false，非空字符串与节点集为 true。

例如，现在想获得价格大于 10 并且国家属性是 USA 的 cd 的 title，下面的 XPath 表达式检查价格是否大于 10：

```
price > 10 and @country = 'USA'
```

下面是应用这个表达式对应的模板：

```
<xsl:template match="cd">
  <xsl:if test="price &gt; 10 and @country = 'USA'">
    <title>
      <xsl:value-of select="title" />
    </title>
  </xsl:if>
</xsl:template>
```

没有 xsl:else 与 xsl:else-if 指令，要选择多个选项中的一个，可以用 xsl:choose 指令。

xsl:choose

xsl:choose 指令选择多个选项中的一个，其中有一个或几个 xsl:when 元素，其中每个元素各有一个 test 属性和一个模板，并将第一个 test 属性求值为 true 的 xsl:when 元素实例化，忽略其余的 xsl:when 元素。还可以放上最后一个可选 xsl:otherwise 元素，在所有 xsl:when 元素都为 false 时实例化它的模板。

例如，当 cd 的价格大于 10 时，打 8 折，否则打 9 折。

```
<xsl:template match="cd">
  <cd>
  <title>
    <xsl:value-of select="title" />
  </title>
  <xsl:choose>
  <xsl:when test="price &gt; 10">
    <price><xsl:value-of select="0.8 * price"/></price>
  </xsl:when>
  <xsl:otherwise>
    <price><xsl:value-of select="0.9 * price"/></price>
  </xsl:otherwise>
  </xsl:choose>
  </cd>
</xsl:template>
```

3.6.7 变量

可以再 XSLT 中声明并使用变量。可以使用<xsl:variable>和<xsl:param>元素。例如，下面声明了一个 price 变量，它的值为 price 元素的值：

```
<xsl:variable name="cd1price" select="cd[1]/price"/>
```

使用这个变量的方法如下：

```
<xsl:value-of select="$cd1price"/>
```

变量赋值还可以在元素内容中，比如

```
<xsl:variable name="price">
<xsl:value-of select="cd[1]/price"/>
</xsl:variable>
```

变量的值可以是任意 XSLT 类型。<xsl:param>的使用方式和<xsl:variable>一样，不同的是，这种方式声明的值是一个缺省值，在调用模板时，可以使用<xsl:with-param>元素对变量值重新替换。这一点可以在 2.6.8 节看到。

另外，变量一旦被声明并赋值，是不能更改的，这一点类似于程序设计语言中的常量类型。

3.6.8 按名称调用模板

还有另一种将模板实例化的方法。除了匹配输入文档中的节点，还可以用 xsl:call-template 元素按名称调用模板。可以将参数传入这种模板。事实上，正是递归才使 XSLT 具有图灵完整性。

例如，下面的 faultPrice 模板

```
<xsl:template name="faultPrice">
<price>10</price>
</xsl:template>
```

xsl:call-template 元素对上下文节点采用一个命名模板。例如

```
<xsl:when test="not(@price)">
  <xsl:call-template name="faultPrice"/>
</xsl:when>
```

命名模板可以通过自上而下的设计分离成在多个地方使用的常用代码。就像 Java 程序中的复杂算法可以分解为多个算法，而不是使用一个大方法。

将参数传入模板

每个模板规则可以有多个参数，用 xsl:param 元素表示。这些参数位于 xsl:template 元素中，放在模板前面。每个 xsl:param 元素有一个 name 属性和一个可选 select 属性。Select 属性提供调用模板是这个参数的默认值，但可以覆盖，如果省略 select 属性，则参数默认值由 xsl:param 元素内容设置。

例如，下列折扣模板中的参数指定打折的数值，默认折扣为 9 折：

```
<xsl:template name="discount">
  <xsl:param name="discnum" select="0.9"/>
  <price>
    <xsl:value-of select="$discnum * price"/>
  </price>
</xsl:template>
```

xsl:call-template 元素可以用 xsl:with-param 子元素提供每个命名参数的值，也可以接受 xsl:param 元素指定的默认值。例如：

```
<xsl:template match="cd">
  <cd>
  <title>
    <xsl:value-of select="title" />
  </title>
  <xsl:choose>
    <xsl:when test="price &gt; 10">
      <xsl:call-template name="discount">
        <xsl:with-param name="discnum" select="0.8"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="discount"/>
    </xsl:otherwise>
  </xsl:choose>
  </cd>
</xsl:template>
```

3.6.9 用 Java 扩展 XSLT

用 Java 编写的大多数 XSLT 处理器也可以反过来讲 Java 代码集成到 XSLT 样式单中。这样做的常见目的是访问 XSLT 没有提供的操作系统功能，如查询数据库、列出目录中的文件和用对话框请用户提供更多信息。也可以用 Java 实现一些复杂算法，这是用 Java 可能比功能性的 XSLT 更方便。例如，尽管可以在 XSLT 中进行复杂字符串搜索与替换，但利用 Java 会容易上千倍，特别是利用好的正则表达式类库会更加有效。最后，即使在纯粹 XSLT 中实现一个功能更容易，有时也会是用 Java 编写以提高性能。这时阶乘与 Fibonacci 数之类的数学函数更是如此。XSLT 优化器不如 Java 优化器那么成熟，他们主要优化 XPath 搜索和求值节点集，而不是优化数学运算。

XSLT 定义了两种将 Java 代码集成到 XSLT 样式单中的机制：扩展函数与扩展元素。编写和使用扩展函数与扩展元素有两个基本部分：

将扩展关联到样式单，这是通过名字空间、类名与 Java 类路径实现的。

将五种 XSLT 类型（number, Boolean, string, node-set 与 result tree fragment）映射为 Java

类型和进行反向映射。

3.6.10 EXSLT（Extensions to XSLT）

EXSLT 是标准的对 XSLT 的扩展。他提供了很多 XSLT 所不具备的元素和函数，比如日期、集合运算、正则表达式、数学运算等等。有很多 XSLT 处理器都内置对 EXSLT 的支持。

下面是一个打印日期的 XSLT：

```
<?xml version="1.0" encoding="gb2312"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:date="http://exslt.org/dates-and-times">
<xsl:output method="text"/>

<xsl:template match="/">
<xsl:value-of select="date:date()"/>
</xsl:template>

</xsl:stylesheet>
```

首先声明名称空间 date，然后就可以使用对应的函数了。

4. 网页信息抽取平台

4.1 网页信息抽取的难点

Web 文档的特殊性使得信息抽取这项工作并不容易，主要表现在：

- 网页内容缺乏语义信息。HTML 着重于数据的表现，而不是数据的描述。这样，想寻找特定数据基本不可能通过标签信息获得。因此，试图自动的在 XML 文档中搜寻想要的数据是极为困难的。
- 网页内容的易变性。网页主要用于发布信息，经常会有新的信息加入进来或者旧的信息被删除。一个健壮的抽取规则必须能尽量不受这种改变的影响。
- 网页表现形式的易变性。网页的表现形式会经常更换。而表现形式一旦改变，抽取规则基本上就不再适用了。
- 网页内容的不规则性。HTML 文档并不是完全结构化的，最多只能算是半结构化的。经常有几种不同的信息糅杂在同一块文本中。这样，必须使用非结构化的抽取方式。

4.2 网页信息抽取平台的目标

任何工具或者软件的目标就是最大程度提高人们的工作效率，尽量减少人的工作量。同样，本文的信息抽取平台目标就是尽量减少人为的参与，简化 Wrapper 的构造。详细地说，要达到以下几个目标：

- **简单性**。用户应该能够很容易，很快的构造一个 Wrapper 程序。
- **健壮性**。这个 Wrapper 程序应该足够健壮。尽量不受网页内容和表现形式变动的影响。
- **通用性**。这个 Wrapper 程序要尽量通用。不仅适应同一个网站内相似的一类网页，甚至是其他网站相似内容的网页。
- **自动性**。Wrapper 能被自动或者半自动生成。

下面将介绍为达到这些目标本文所使用的方法以及所做的工作。

4.3 基于 XSLT 的抽取模式

XML 已经成为一个流行的数据交换、存储与操作的格式标准。围绕这种格式，各种各样的相关标准、工具已经被开发用以操纵 XML 数据。XSLT 可以将一个 XML 文档中的数据抽取出来并表示为一个新的 XML 文档。由于 HTML 可以看作是 XML 的一个子集，可以利用标准的 XSLT 技术对 HTML 文档进行抽取，并将结果表示为 XML 文档。

不过，由于历史的原因，HTML 并不完全符合 XML 语法，而且存在大量的语法错误的 HTML 文档。因此，不能直接将 XSLT 应用到 HTML 文档上，必须先将 HTML 文档转换成符合 XML 语法的 XHTML 文档，然后再应用 XSLT 进行数据抽取。

由于本文的方法是基于 XML 技术的，所以很自然，抽取结果表示为一个 XML 文档或

者程序中的 XML 数据结构，比如一个 DOM 树。

这样，可以得到抽取系统的基本框架：

图 4.1 基本框架

首先获得 HTML 文档对应的 DOM 树，然后应用 XSLT 将这个 DOM 树转换为结果 XML 文档。要构造一个 Wrapper，用户仅仅需要提供一个 XSLT。而不需要编写复杂的搜索与遍历代码。

获取 HTML 文档的 DOM 树

实际中，HTML 文本会存在很多错误，比如不规则的嵌套，开始标记缺少结束标记，属性值没有双引号等等。因此，首先要修正 HTML 网页中的错误。为了得到 HTML 的 DOM 结构，还要将 HTML 转化为对应的 XHTML。

这一步工作很繁重，所幸的是，有一个开源的工具 Tidy[TIDY]可以帮助完成所有这些工作。使用 Tidy，可以很方便的获得一个 HTML 文档的 DOM 树。

不过，对于很多 DOM 特性，Tidy 提供的 DOM 实现并不支持，所以，还要将 Tidy 获得的 DOM 转换为 Xerces-J 的 DOM 实现。

将网页中的链接转化为绝对链接

大部分网页中的链接使用的都是相对链接，相对链接结合网页本身链接才能正确定位到链接指向的页面。仅仅将相对链接抽取出来是没用的，因此，在抽取之前，要将网页中的链接转换为绝对链接。

XSLT 执行引擎

本文使用 Xalan-J[XALAN-J]作为 XSLT 执行引擎。实际上，XSLT 执行引擎可以任意替换而不影响应用程序代码。因为它们都实现了标准的 XML 转换接口。

4.4 示例：利用 GUI 编写 XSLT

基于本文的平台，构造 Wrapper 的工作仅仅是编写一个 XSLT 文档。不过，要编写这个 XSLT 需要了解源文档的结构。但是，HTML 文档与一般的 XML 文档内容上有很大的差异。HTML 文档要复杂得多。

- Ø 一般的 XML 文档结构清晰，都有严格的数据模式。这是因为它们完全是面向数据的。模式用 DTD 或者 Schema 描述。也就是说，单从 XML 模式就可以完全了解 XML 文档的结构。
- Ø 一般的 XML 文档很简单。同样，由于 XML 是面向数据的。完全没有任何 HTML 中大量用于数据表示的标签，因此 XML 文档要比 HTML 文档简单得多，几倍，甚至十几倍，几十倍。
- Ø HTML 文档包含了大量的噪声。几乎所有的网页都包含噪声。噪声是指那些不是页面主要内容的信息，比如广告，导航栏等等。

由于 HTML 文档如此复杂，想在对应的 DOM 树中定位到想要的信息并不是一件很容

易的事情。而即使找到想要的数据，手工计算对应的 XPath 也是极为困难的。针对这个问题，本文开发了一个应用程序 DOMTreeView，方便浏览 HTML 的 DOM 树，定位信息以及获得对应的 XPath。

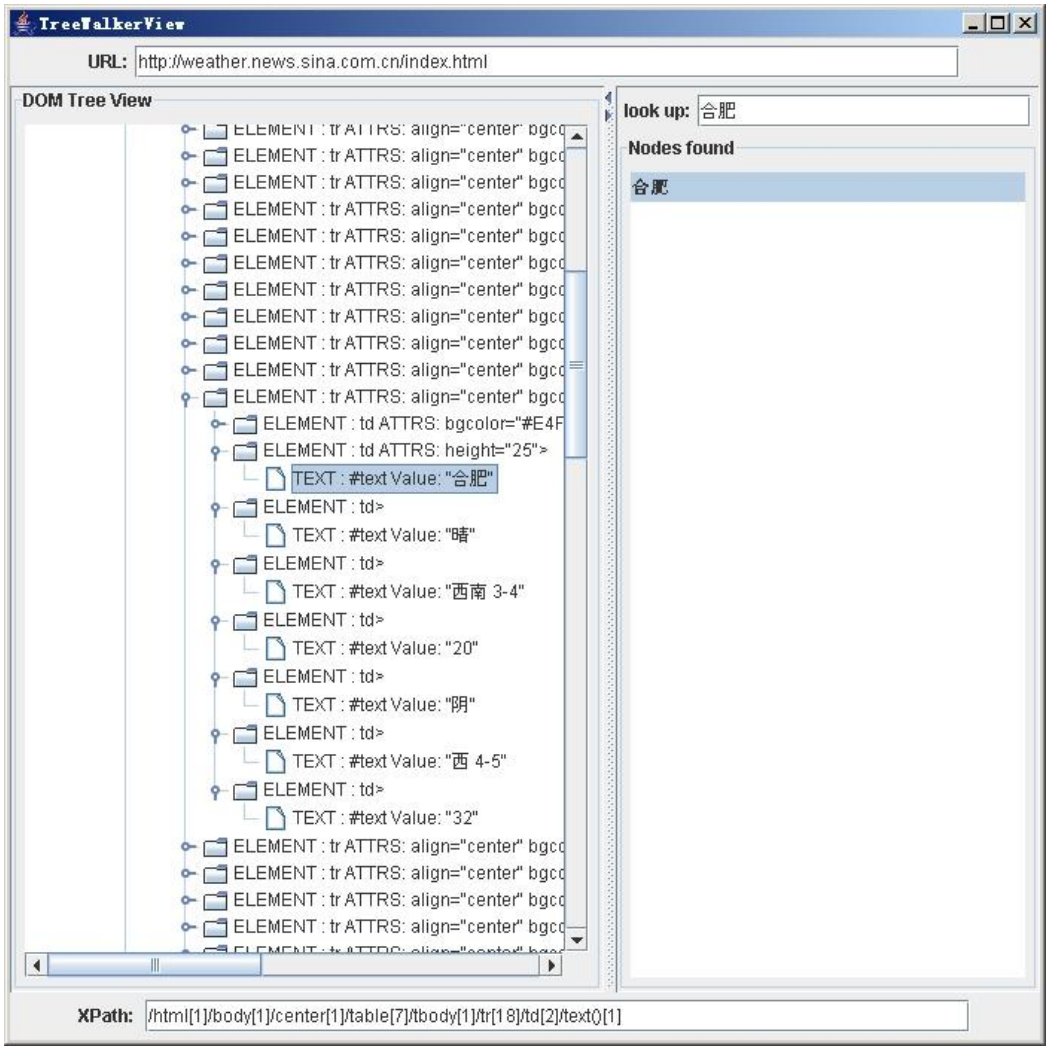


图 4.2 DOMTreeView

下面介绍一个使用这个应用程序编写 XSLT 的一个实际例子：

4.4.1 抽取天气信息

问题描述

抽取合肥市天气预报的信息，信息源使用新浪的天气预报网页。

http://weather.news.sina.com.cn/index.html。如图 4.3 所示。

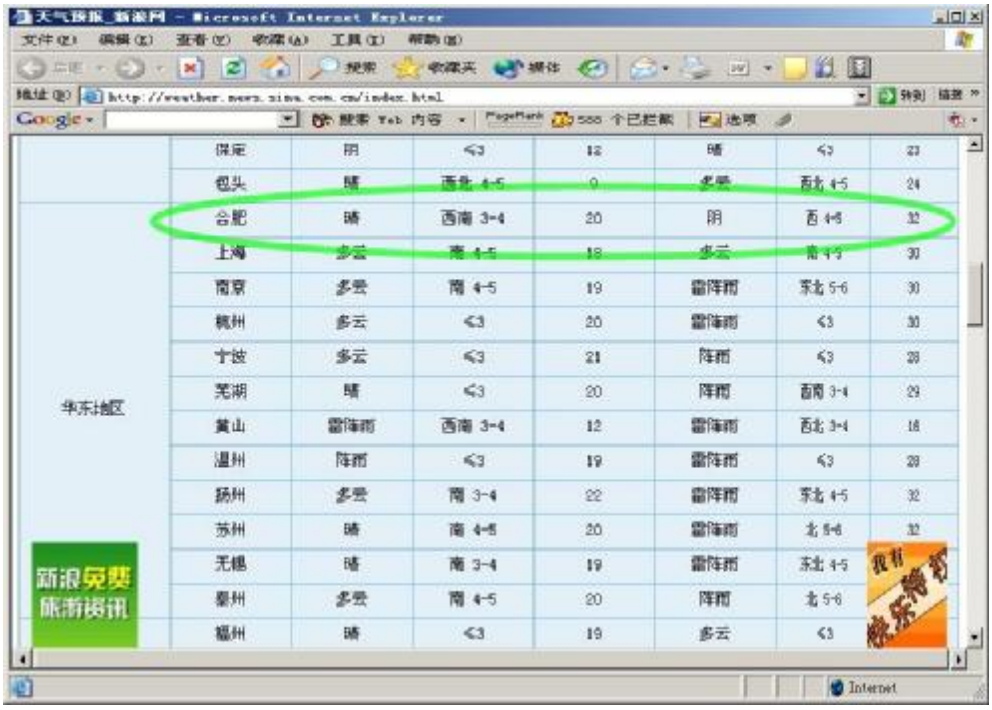


图 4.3 新浪的天气预报网页

希望获得结果：

```
<?xml version="1.0" encoding="GB2312"?>
<hefei>
<tonight>
<condition>雷阵雨</condition>
<wind>北 5-6</wind>
<low>18</low>
</tonight>
<tomorrow>
<condition>阴</condition>
<wind>北 5-6</wind>
<low>21</low>
</tomorrow>
</hefei>
```

解决步骤

运行 DOMTreeView。如图 4.2 所示。

1. 获取源文档的 DOM 树。在 URL 栏输入新浪天气预报网址，并回车。很快会获得对应的 DOM 树（在界面左侧）。
2. 查找“合肥”在 DOM 中对应的位置。在界面右侧的 look up 栏输入“合肥”，并回车。下面的列表框会出现匹配的结果。本例中有一条匹配的结果。点击这条结果，左边的 DOM 树会立刻定位到相应的结点。这个节点被自动选中，背景颜色为靛蓝

色。同时，它所对应的 XPath 会被计算并显示在下方的 XPath 栏中。

3. 接着，定位想要抽取的信息。由于已经定位到“合肥”节点，寻找旁边的文本节点就很容易了。如图，合肥节点下面的几个文本节点都被展开，这几个文本节点就是要抽取的信息。同样，每个节点的 XPath 都会被自动计算并显示在 XPath 栏中，一旦这个节点被选中。
4. 最后，根据每个节点的 XPath 编写 XSLT。

```
   <?xml version="1.0" encoding="gb2312"?>

   <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output encoding="gb2312" indent="yes"/>

   <xsl:template
match="/html[1]/body[1]/center[1]/table[7]/tbody[1]/tr[18]">
   <hefei>
     <tonight>
     <condition><xsl:value-of
select="td[3]/text()"/></condition>
     <wind><xsl:value-of select="td[4]/text()"/></wind>
     <low><xsl:value-of select="td[5]/text()"/></low>
     </tonight>
     <tomorrow>
     <condition><xsl:value-of
select="td[6]/text()"/></condition>
     <wind><xsl:value-of select="td[7]/text()"/></wind>
     <low><xsl:value-of select="td[8]/text()"/></low>
     </tomorrow>
   </hefei>
   </xsl:template>
```

例子 4.1 使用 XSLT 编写的新浪天气预报抽取规则

这一过程不需要 3 分钟就能完成。

5. 抽取规则健壮性研究

网页是易变的。内容几个小时就会变化一次，样式几个月甚至是几个星期就会变化一次。一旦网页发生变化，抽取规则就可能会失效。抽取规则越敏感，面对网页的变化失效的可能性就越大。因此，应当尽可能编写健壮的抽取规则。

5.1 数据定位健壮性研究

XSLT 中节点的定位是基于 XPath。XSLT 失效主要是由于 XPath 的失效。也就是说，需要使用尽可能健壮的 XPath。

前面所使用的 XPath 都是基于树路径的，比如：

```
/html[1]/body[1]/center[1]/table[7]/tbody[1]/tr[18]
```

一般来说，基于树路径的 XPath 最常用，因为这种形式最直观。但是，这种形式虽然适合 XML 文档，但并不适合 HTML 文档。因为 XML 的数据模式很少变化，而且标记名都是语义的，不会出现为不同数据使用相同的标记。但是 HTML 的标记基本都是基于表示的，比如 center, table 等等。不同的数据往往也会使用相同的标记，比如 tr，td。这样，当 HTML 文档加入或者删除一部分数据时，XPath 就很可能会有变化。比如，在“合肥”前面再加入一个城市，那么 XPath 就会变成：

```
/html[1]/body[1]/center[1]/table[7]/tbody[1]/tr[19]
```

最后一项的 index 发生了变化，原来的 XPath 也就失效了。

实际上，树路径上只要任何一层发生了变化，整个树路径就会失效。因此，树路径越长就越敏感。

Jussi Myllymaki[MJ03]提出了基于内容的定位。也就是根据网页上不易变化的文本信息进行定位。比如，可以改写上面那个 XPath 表达式为：

```
//tr[contains(normalize-space(.), '合肥')]
```

这个 XPath 就要健壮得多。

他们提出了 Anchor-Hop 的模型。这个模型很直观，Anchor 是一个参照节点。一般是包含所有要抽取的信息的共同的最近的祖先节点。然后使用相对路径定位到各个要抽取的信息，也就是 hop。

由于相对路径很短，因此不是很敏感。但是参照节点的绝对路径一般很长，因此要采用健壮些的方法，比如上面提到的基于内容的定位。

除了这种定位模式外，本文提出了另外两种健壮的定位方法。

5.1.1 完全基于文本的定位

实际上，基于树路径的相对路径也可以改为其他的定位方式。比如，可以完全基于网页的文本内容，而不根据树结构。可以重写 XSLT 如下所示：

```
<?xml version="1.0" encoding="gb2312"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="gb2312" indent="yes"/>

<xsl:template match="//text()[contains(., '合肥')]">
<hefei>
  <tonight>
  <condition><xsl:value-of
select="following::text()[1]"/></condition>
  <wind><xsl:value-of select="following::text()[2]"/></wind>
  <low><xsl:value-of select="following::text()[3]"/></low>
  </tonight>
  <tomorrow>
  <condition><xsl:value-of
select="following::text()[4]"/></condition>
  <wind><xsl:value-of select="following::text()[5]"/></wind>
  <low><xsl:value-of select="following::text()[6]"/></low>
  </tomorrow>
</hefei>
</xsl:template>
<xsl:template match="text()"/>
```

例子 5.1 完全基于文本的定位规则

首先定位到参照文本“合肥”，然后依次获取其后面的文本节点。这样一个模式就完全与树结构毫无关系，因此也不会受到网页结构改变的影响。而且，编写抽取规则是也不用去了解 HTML 的复杂结构。

5.1.2 使用属性模式定位

实际上，XSLT 提供了强大的语言和类库支持，可以方便编写各种健壮的定位规则。关键是选择好的约束条件。其实，定位节点的 XPath 就是约束条件。

除了使用树路径和文本约束这两种方法外，还可以使用属性约束。比如一个图片的大小，一个 URL 的模式等等。任何不变而且与众不同的模式都可以利用来编写健壮的抽取规则。

例如，要定位新浪新闻首页（http://news.sina.com.cn/ ）焦点新闻图片，可以有两种方式：

根据 img src 属性：

```
    //img[contains(@src, 'FocusPic')]
```

根据图片大小属性：

```
//img[@width=263 and @height=198]
```

5.1.3 不同定位模式的讨论

上面介绍了几种不同的定位模式，每种模式都具有各自独特的角度。不同的角度有不同的优点，但也有对应的缺点。对于实际的问题，应当选取最能有效解决问题的模式。下面将详细讨论各种模式的优缺点。本文从三个角度出发：

1. 精确性。或说是唯一性。一个定位应当只定位到指定的节点，而不能同时还匹配其他节点。

2. 通用性。一个定位应当能适应网站结构的变化。

3. 简易性。一个定位应当能很快能找到。

基于路径的 XPath 使用的约束是节点名，节点所处的 index 以及路径结构。他基于网页 DOM 树的结构。因此会受到树结构变化的影响，但是这种方法最为精确，因为树路径是唯一的。定位也能很快找到，如果是用我编写的软件的话。

基于文本的约束使用的则是文本节点的内容。它基于人们发布信息的方式。比如合肥的天气预报前面一般有“合肥”二字以指示后面的内容是合肥的天气，而不是其他城市的天气。这种方式与人发布和处理信息的模型最为相似，因此不会受到网页结构变化的影响，而且甚至能应用到不同的网站上。但是有时要寻找这样的模式并不容易，有可能找不到关键词，也有可能关键词被匹配多次。

基于属性的约束使用的是元素的属性。他基于一个元素特有的一些属性，一般来说是显示属性或者链接属性。显示属性一般选取大小信息（而且是绝对大小），因为大小涉及到一个整体的布局，不会轻易变动，除非网站的显示风格完全改变。背景、颜色这类的信息很容易变动，而且唯一性差，因此不适合用来约束。另外一个问题是属性信息可以被放到一个网页的级联样式单（CSS）中，不是直接引用，而是是用对应的 class。不过 class 往往含有语义性质，因此也可以用来作为显示属性的约束，而且效果更好。链接属性基于的假定是链接对象经常会被放到网站一个特定的位置。基于属性的约束通用性也比较好，但是大部分情况都不容易找到，唯一性也很难保证。

下表是一个简单的总结

	精确性	通用性	简易型
树路径	很高	很差	很简单（GUI）
文本内容	一般	很高	比较简单
属性	较差	较高	较差

从上表可以看出，基于树路径的方式精确而且简单，但是通用性很差。它适合于对通用性要求不高的问题，比如只进行一次抽取的任务。基于文本内容的方式通用性很高，精确性和简易型也还可以。这种方式适合于对通用性要求很高的问题。而基于属性的方式表现最差，但是并非一无是处。它适用于对通用性有一定要求而又找不到合适的文本定位的问题。比如标题新闻图片。

5.2 基于缩略路径的数据抽取

XPath 中，在需要时会将节点转换为对应的字符串。一个节点的字符串是他所有子孙文

本节点的和。因此，在获取数据时，不需要定位到文本节点。例如，下面的 XPath 会返回节点 a 的文本信息：

```
<xsl:value-of select="a">
```

由于 HTML 中，经常会在文字上加入各种各样的字体属性，定位到文本节点的路径就会不通用，而且很脆弱。而直接获取节点的字符串可以避免这些字体元素的影响。比如上面的指令要比下面的指令健壮得多：

```
<xsl:value-of select="a/text()">
```

```
<xsl:value-of select="a/font/text()">
```

5.3 构造通用的链接组抽取模式

网页中出现最多的元素就是链接。特别是新闻网站，首页几乎百分之百都是链接。而链接一般都会被适当的分组，相同主题的链接放在一起。而且每组链接前面都会放置对应的标题信息，以标识这组链接所属的主题。很多情况下，我们希望能抽取出某个主题的链接组，比如新浪新闻首页的国际新闻最新消息，比如 bbs 的今日十大热门话题等等。

理想的情况是，给定标题文字，就能抽取出对应的链接组。

由于 HTML 是基于表示的，外表看起来排列在一起的数据对应的内部结构可能并不排列在一起，而且树结构可能很复杂。因为可以有很多方法显示出一个链接列表，比如可以用 ul（无需列表）元素，也可以用 table 元素，也可以直接使用 br（换行）元素。

但是可以找到链接组的一些特性：

1. 这组链接列表可以找到一个**最小共同父亲节点**，最小是指离根节点最远的那个节点。
2. 最小共同父亲节点除了包含这组链接列表外，不包含其他链接组或者链接（链接组标题链接除外）。
3. 链接组包含多条链接。（一般至少 4 条）

要抽取链接组，就要先定位到他们的最小共同父亲节点。本文希望仅根据标题文字就能定位到。有这两种情况：

1. 标题文字节点不是最小共同父亲节点的子孙节点。这种情况，本文假定标题文字节点后的第一个包含多个链接的节点就是最小共同父亲节点。
2. 标题文字节点是最小共同父亲节点的子孙节点。这种情况，本文假定标题文字节点是最小共同父亲节点下的第一个子孙文本节点。

对应于两种情况，分别编写定位表达式如下：

1.

```
//text()[string(.)=$title]/following::*[count(.//a)>=$min][1]
```

$title 是标题文字，$min 是链接组最少的链接数。首先定位到标题文字节点，然后从这个节点出发，寻找后面包含多个链接的第一个节点。

2.

```
//text()[string(.)= $title)]/ancestor::*[starts-with(.,$title)]
[count(.//a)>= $min][1]
```

首先定位到标题文字节点，然后从这个节点出发，寻找以标题文字开始并且包含多个链接的第一个祖先节点。

合并

将这两个规则合并到一起。先利用第二个规则，因为第二种情况如果出现的话，肯定比第一种情况更合适。也就是应当先考虑第二种情况。合并后的表达式如下：

```
    ($xpath2 | $xpath1)[1]
```

xpath1 是第一个规则，xpath2 代表第二个规则。先取两者集合的并，然后取第一个节点。

不过，简单的这样做会有问题。因为有可能出现这样的情况，标题文字节点不是最小共同父亲节点的子孙节点（也就是属于第一种情况），但是恰巧第二种情况也能找到，即以标题文字结点开头而且包含链接组的祖先节点，而这个节点不仅包含这个标题对应的链接组，还包含了其他链接组。对于这种情况，需要针对第二种情况进一步过滤，

```
   $node1 = select(xpath1)
   ($xpath2[count($node1)=0    or    count(*|$node1)!=count(*)|
$xpath1) [1]
```

$node1 代表了使用第一个规则所选中的节点。$xpath 后面的谓词的意思是$node1 为空或者标题文字节点的祖先节点不是$node1 的祖先节点。

6. 自动归纳网页模板

第三章和第四章主要介绍了如何利用本文的平台和标准的 XSLT 快速有效的编写健壮的抽取规则。对于特定的抽取任务，这种方法是很高效的。本章和下面一章试图从另外一个角度着手，“自动”的发现数据并生成抽取规则。这一角度是基于这样的观察：同一网站的网页彼此相似，网页中大部分内容都是固定不变的，只有那些变化的内容才是真正有用的信息。网页内部也存在这种模式，即存在结构相似的信息块，比如搜索结果列表，每条结果的表现形式都是一样的。本章试图根据这种启发式知识发现网页的模板。根据模板可以滤除不变的内容，而只留下有用的数据。

本文将模板分为两种，相似网页的模板和网页内部相似信息块的模板。本章介绍网页模板的归纳，记录模板的归纳将在下一章介绍。

6.1 引言

大部分网页都是由模板生成的，由同一个模板生成的网页具有相似的结构和外表。比如 npr 的新闻页面（图 6.1）。它们具有相似的外表，除了新闻内容部分不一样之外，其它部分都完全一样。相同的部分不妨称之为**页面框架**。页面框架一般包括导航栏、广告、版权声明之类的内容。页面框架虽然可以帮助用户浏览网页，但是并不代表页面本身的内容，它属于冗余的信息。对于利用页面内容进行分析的问题，框架内容是极为有害的。滤除框架有利于：链接分析（链接分析是很多工作的基础，比如搜索引擎中的 pagerank，网上社区的发现等等）、网页分类、网页聚类、页面相关度计算、信息检索、人们查看信息等等。滤除框架也有利于信息抽取，因为人们不大可能去抽取框架中的信息。

本文希望通过一组（至少两个）相似的页面，就能归纳出对应的模板，并生成相应的抽取规则，进而利用抽取规则抽取页面数据。这里有两个问题本文暂不考虑：1，如何自动找到相似的页面；2，如何自动对抽取出的数据进行标记，例如哪是新闻标题，哪是新闻内容。这两步工作都暂由人来完成。

本文提出了基于 DOM 树的方法，通过比较相似网页的 DOM 树来定位不同的部分，进而生成网页主要内容的抽取规则。

6.2 相关工作

尽管归纳网页模板是很重要的一个工作，但很少有人从事这方面的研究。

[LH02]提出了发现新闻源网页信息块的方法。首先根据<table>标签将网页划分为多个内容块，然后获取每个内容块的特征（关键词），接着根据特征计算内容块的信息熵，信息熵大的内容块就是信息块。这种根据<table>划分方式极为粗略。HTML 中有大量的元素都可以分割信息块。

在[YR02]中，首先将网页分解成子网页（Pagelet）。这些子网页被认为是一个逻辑整体。然后聚类子网页，每一类便是一个模板。他们的目标主要是为了提高链接分析的质量。因此，子网页被定义为链接超过三条的元素，而且这个没有超过三条链接的子孙元素。这样会忽略掉很多网页中的内容。这种基于子网页的方式并不适合获取网页主要内容，也不适合分理出网页中变化的信息。另外，这种方式也不能专门针对某个网站的网页获得相应的网页模板。

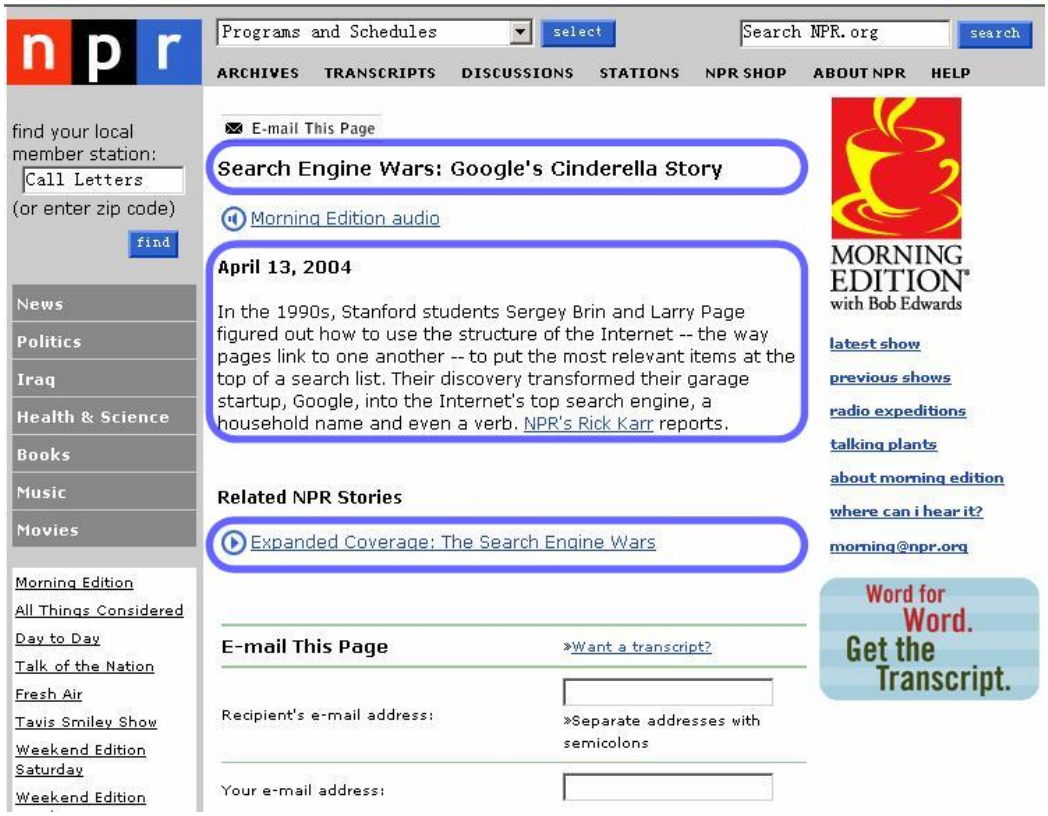


图 6.1(a) npr 一个新闻页面

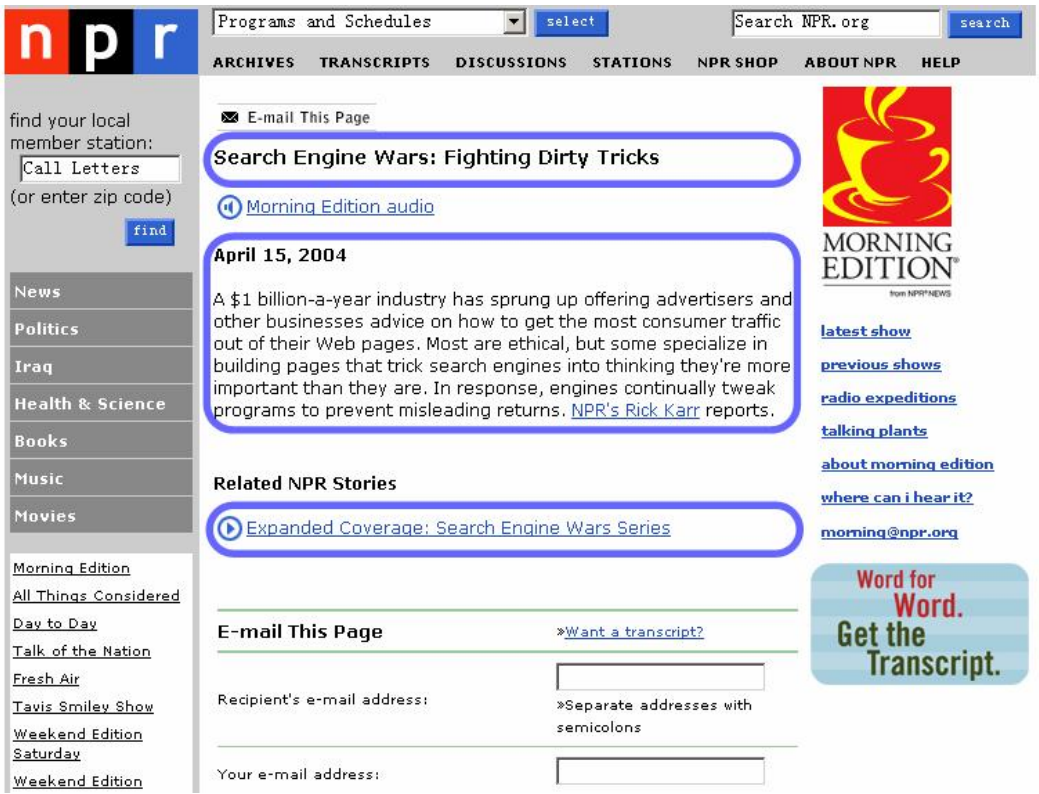


图 6.1(b) npr 的另一个新闻页面。

图 6.1 (a)和(b)是两个相似的新闻页面。除了标题、日期，新闻内容和相关新闻不一样外，网页的其它部分都完全一样

[YLL03]试图解决的问题和本文是一样的，而且他们提出的方法也与本文类似。不过他们是基于样式树（Style Tree）的。样式树是他们提出来的一个基于 DOM 的模型。不同之处在于样式树是多个 DOM 树的合并。与本文的树模式也很类似。不过，样式树在处理具有不同孩子节点的元素时是对每种出现的情况单独作为一类，这种方式对处理子模板与数据元素为兄弟节点的情况很不好。而且用来过滤噪声数据的样式树很臃肿。另外，要计算一个样式树往往需要大量的实例。最后，他们的方法将属性节点的数据都被会略掉了，而对于链接这样的数据往往是有用的。

[Dav00]与[Kus99]使用了机器学习的方法来识别广告条，网页中冗余和不相关的链接。但是，这些方法需要人为标记大量的样本和领域知识，而且，他们并没有过滤出网页中的主要内容。

6.3 模型和假定

本文的模型是基于 DOM 树的。本文假定相似网页具有相似的 DOM 结构。框架对应于 DOM 树中相同的部分，而数据对应于 DOM 树中不同的部分。实际上，只要找出不同的部分，其余的便是相同的部分，反之亦然。本文定义两种模板，它们是相互引用而且是递归的：

树模板（Tree Template, 简称 TT）：

树模板是一棵树，包含 0 个或多个森林模板，森林模板处于树的叶结点上。

森林模板（Forest Template，简称 FT）：

森林模板也是一棵树，具有一个根节点，根节点下包含 0 个或多个树模板，其它的子树属于变化的内容。如图 6.2 所示。

本文假定相似网页的模板就是一个树模板，其中变化的部分处于树模板所包含的森林模板中，而森林模板除了包含不同的子树外还可能包含不变的子树或者树模板。

森林模板中的树模板并不是固定的，也就是说，这个树模板没有一个固定的位置，而且有可能有，也有可能没有。因为数据是任意的，而且模板可能会根据上下文数据来确定插入还是不插入。要识别一个棵树是否是树模板，需要判断这棵树是否匹配树模板，而不能通过第几颗子树或者根节点名来进行判断。通过这种方式，本文可以捕捉森林模板中出现的树模板，而这是其他方法根本无法做到的。

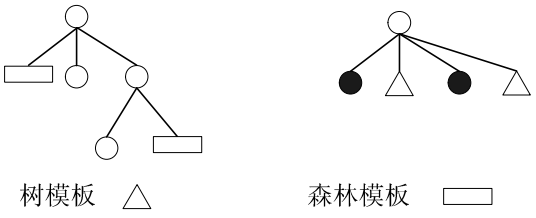


图 6.2 树模板和森林模板模型

树模板的模板

将树模板中的森林模板替换为其根节点，就得到树模板的模板。

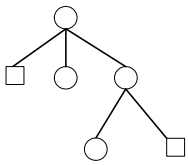


图 6.3 树模板的模板

树模板的变化节点与不变节点

树模板中，森林模板的根节点为变化节点，其它节点为不变节点。不同的树中不变节点的孩子节点完全相同，而变化节点除了本身相同外，孩子节点不同。如图 6.3 所示，圆形节点为不变节点，方形节点为变化节点。方形节点为森林模板的根节点。

树的复杂度

对于一棵树 T，本文定义它的复杂度 C(T)为所有节点（包括属性节点）的数量。例如： hello 的复杂度为 2。包含一个元素节点和一个文本节点。

E-mail This Page 的复杂度为 4。包含一个元素节点，两个属性节点和一个文本节点。

树模板的复杂度

树模板的复杂度为树模板的模板复杂度。

树模板与数据的复杂度

本文假定树模板的最小复杂度为 4。基于这样的假定：

数据是简单的，一般就是一个文本节点，或者一个属性节点，或者一个文本加上一个标签，复杂度一般小于 4。而模板是复杂的，包含复杂的嵌套结构和一堆的属性。

数据一般是人手工填写的或者从数据库中取得的。从数据库中取得的数据一般就是纯文本节点，而手工填写数据不会使用复杂的多层嵌套以及太多的属性。

本文将利用最小复杂度来过滤出候选树模板。

相似度

本文定义两棵树的相似度为它们的树模板的复杂度。如果相似度大于等于最小复杂度 4，那么本文认为这两棵树是相似的。

本文假定如果 t1 和 t2 是相似的，而且 t2 和 t3 是相似的，那么 t1 和 t3 也是相似的。而且它们具有一个共同的树模板，这个树模板的复杂度大于等于最小复杂度。

本文还假定如果 t1 和 t2 相似，而 t1 和 t3 不相似，那么 t2 和 t3 也是不相似的。

6.4 归纳树模板

给定几棵（至少两棵）树，本文希望能归纳出对应的树模板。本文的算法基于前面定义的模型，分为两个模块，同样也是递归的。第一个模块是归纳树模式(inductTT)，首先使用递归的匹配函数(match)寻找变化节点并归纳森林模式，然后根据变化节点创建树模板的模

版（createTemplate）。匹配函数从所有树的根节点开始，自顶向下不断匹配。若根节点都匹配，则匹配根节点下的子树和属性节点。否则，归纳森林模式。根节点匹配(isNodeMatch)当且仅当自身相同，孩子节点相同以及属性相同。四个主要函数说明如下：

```
inductTT(roots)
归纳树模板
输入：  训练样本的根节点
输出：  树模板
match(roots); //自顶向下匹配，获得森林模板集
createTemplate(); //创建树模板的模板
```

```
createTemplate (roots)
创建树模板的模板
T = get first tree of roots;
deleteFT on T; //删除 T 上的所有森林模板，但保留森林模板的根节点。
```

```
match(roots)
从根节点开始自顶向下匹配
if isNodeMatch(roots)
      matchAttributes(roots)
      matchChildren(roots)
else
      addFT(inductFT(roots))
```

```
isNodeMatch (roots)
判断根节点是否匹配
return isNodeEqual(roots)
          && isChildrenEqual(roots)
          && isAttributesEqual(roots);
```

第二个模块是归纳森林模式(inductFT)。从不匹配的根节点开始，寻找根节点下的树模板。首先过滤出候选子树集(getCandidates)，一棵树是候选子树当且仅当它的复杂度至少为树模板最小复杂度时。然后对子树集进行聚类(cluster)，根据树的相似性。最后对每一类归纳对应的树模板。要计算两棵树是否相似，首先归纳他们的树模版，如果树模板的复杂度小于最小复杂度，则这两棵树是不相似的，否则相似。四个主要函数说明如下：

```
inductFT(roots)
归纳树模板
输入：  训练样本的根节点
输出：  森林模板
candidates = getCandidates(roots);
clusters = cluster(candidates,
   treesimilar, 2);
for each cluster in clusters
      addTT(inductTT(cluster));
```

```
getCandidates (roots)
获取候选子树
输入：  训练样本的根节点
输出：  候选子树
children = getChildren(roots);
for each child in children
   if complexity(child) < mincomplexity
         remove child
return children
```

```
cluster(candidates, similar, minsize)
对候选树进行聚类，聚类大小小于 minsize 的类别将被丢弃
输入：
   candidates: 候选树集
   similar: 相似性衡量器
   minsize: 最小聚类大小
输出：  聚类
      clusters = new list();
      for each o in candidates
         for each cluster in clusters
            first = get first object of cluster
      if (similar.isSimilar(o, first)
         cluster.add(o)
         break;
      endif;
         endfor;
         newcluster = new list();
         newcluster.add(o);
         clusters.add(newcluster);
      endfor;
      remove cluster whose size < minsize in clusters
      return clusters;
```

```
      treesimilar:
      isSimilar (t1, t2)
      计算两颗树是否相似
      TT = inductTT(t1, t2);
      if complexity(TT) < mincomplexity
         return false;
      else return true;
```

获得树模板后，就可以根据这个树模板自动生成相应的抽取规则。抽取规则仍然用 XSLT。元素有四种类型：Template, Forest, Atom 和 var。分别对应于树模板、森林模板、原子节点和数据节点。 如果森林模板的根节点是文本节点或者属性节点，使用 Atom。对于森林节点下的非树模板，使用 var 元素。

自动生成的元素标记名为元素类型加上一个递增的整数。

森林模板可能包含树模板，因此首先检测一个节点是否为树模板的根节点，使用 XSLT 中的 choose, when 和 otherwise 指令，类似于程序设计语言的 switch 指令。检测方式为测试根节点代表的子树是否匹配树模板的模版，测试元素名或者属性是否相等，自顶向下选择 10 个测试谓词。

下面是本文对 npr 前面两个例子进行归纳并生成抽取规则的结果：

```
<?xml version="1.0" encoding="gb2312" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output encoding="gb2312" indent="yes" />
  - <xsl:template match="/html">
    - <Template12>
      - <Atom0>
          <xsl:value-of select="head[1]/title[1]/text()[1]" />
        </Atom0>
      - <Atom1>
          <xsl:value-of select="head[1]/meta[2]/@content" />
        </Atom1>
      - <Atom2>
          <xsl:value-of select="head[1]/meta[4]/@content" />
        </Atom2>
      + <xsl:for-each select="body[1]/div[1]/table[1]/tr[1]/td[3]/table[1]/tr[1]/td[1]">
      </Template12>
    </xsl:template>
  </xsl:stylesheet>
```

根节点下有四个森林模板，其中前三个是文本节点或者是属性节点，而且处于 head 节点下。第四个森林模板展开如下：

```
- <xsl:for-each select="body[1]/div[1]/table[1]/tr[1]/td[3]/table[1]/tr[1]/td[1]">
  - <Forest11>
    - <xsl:for-each select="node()">
      - <xsl:choose>
        - <xsl:when test="local-name()='p' and ./@class='text' and a[1]/img[1] and a[1]/img[1]/@align='left' and
            a[1]/img[1]/@alt='audio icon' and a[1]/img[1]/@border='0' and a[1]/img[1]/@height='18' and a
            [1]/img[1]/@src='http://www.npr.org/images/audiospeakericon.gif' and a[1]/img[1]/@width='18'
            and a[2]/text()[1]">
          - <Template5>
            - <Atom3>
                <xsl:value-of select="a[1]/@href" />
              </Atom3>
            - <Atom4>
                <xsl:value-of select="a[2]/@href" />
              </Atom4>
            </Template5>
          </xsl:when>
        - <xsl:when test="local-name()='p' and b[1]/@style='font-size:12px;' and b[1]/text()[1] and br[1] and br
            [2] and a[1]/img[1] and a[1]/img[1]/@align='left' and a[1]/img[1]/@alt='Expanded Coverage: The
            Search Engine Wars' and a[1]/img[1]/@border='0' and a[1]/img
            [1]/@src='http://www.npr.org/images/icon_more.gif'">
          - <Template9>
            - <Atom6>
                <xsl:value-of select="a[1]/img[1]/@alt" />
              </Atom6>
            - <Atom7>
                <xsl:value-of select="a[2]/text()[1]" />
              </Atom7>
            - <Atom8>
                <xsl:value-of select="script[1]/text()[1]" />
              </Atom8>
            </Template9>
          </xsl:when>
        - <xsl:when test="local-name()='form' and ./@action='http://www.npr.org/sendEmail/email_sent.php'
            and ./@method='post' and ./@name='frmSendToFriend' and ./@onsubmit='return formSubmit();' and
            comment()[1] and input[1] and input[1]/@name='pageUrl' and input[1]/@type='hidden' and input
            [1]/@value=''">
            <Template10 />
          </xsl:when>
        - <xsl:otherwise>
          - <var>
              <xsl:copy-of select="." />
            </var>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </Forest11>
  </xsl:for-each>
```

包含三个树模板，分别对应于 Template5，Template9 和 Template10。其中 Template5 和 Template9 下面还包含多个森林模板，不过都是文本节点或属性节点。

使用上面自动生成的抽取规则抽取网页中的结果如下：

```
<?xml version="1.0" encoding="GB2312" ?>
- <Template12>
    <Atom0>NPR : Search Engine Wars: Google's Cinderella Story</Atom0>
    <Atom1>google, search engines, search engine wars, rick karr, internet
      search</Atom1>
    <Atom2>In the 1990s, Stanford students Sergey Brin and Larry Page figured out
      how to use the structure of the Internet -- the way pages link to one another --
      to put the most relevant items at the top of a search list. Their discovery
      transformed their garage startup, Google, into the Internet's top search engine,
      a household name and even a verb. NPR's Rick Karr reports.</Atom2>
  - <Forest11>
    - <var>
        <!-- ==begin paste page content==  -->
      </var>
    - <var>
        <script src="http://www.npr.org/include/javascript/emailafriend_top.js"
          type="text/javascript" />
      </var>
    - <var>
        <!--  COLUMN INFO   -->
      </var>
    - <var>
        <p class="pagetitle">Search Engine Wars: Google's Cinderella Story</p>
      </var>
    + <Template5>
    - <var>
        <b>April 13, 2004</b>
      </var>
    - <var>
        <br />
      </var>
    - <var>
        <br />
      </var>
      <var>In the 1990s, Stanford students Sergey Brin and Larry Page figured out
        how to use the structure of the Internet -- the way pages link to one another
        -- to put the most relevant items at the top of a search list. Their discovery
        transformed their garage startup, Google, into the Internet's top search
        engine, a household name and even a verb.</var>
    - <var>
        <a href="http://www.npr.org/about/people/bios/rkarr.html">NPR's Rick
          Karr</a>
      </var>
      <var>reports.</var>
    - <var>
        <br />
      </var>
    - <var>
        <br />
      </var>
    - <var>
        <br />
      </var>
    + <Template9>
      <Template10 />
    - <var>
        <br />
      </var>
    - <var>
        <br />
      </var>
    - <var>
        <br />
      </var>
    - <var>
        <!-- ==end paste page content==  -->
      </var>
  </Forest11>
</Template12>
```

其中，Template5 和 Template9 的结果展开如下：

```
- <Template5>
    <Atom3>javascript:getMedia('ME', '13-Apr-2004', '15', 'RM,WM');</Atom3>
    <Atom4>javascript:getMedia('ME', '13-Apr-2004', '15', 'RM,WM');</Atom4>
  </Template5>
```

```
- <Template9>
    <Atom6>Expanded Coverage: The Search Engine Wars</Atom6>
    <Atom7>Expanded Coverage: The Search Engine Wars</Atom7>
    <Atom8>/** * echeck function modified from DHTML email validation script. Courtesy of
      SmartWebby.com (http://www.smartwebby.com/dhtml/) */function echeck(str) { var at="@"
      var dot="." var lat=str.indexOf(at) var lstr=str.length var ldot=str.indexOf(dot) if (str.indexOf(at)
      ==-1){ alert("Please check the the format of the email addresses you entered.") return false } if
      (str.indexOf(at)==-1 || str.indexOf(at)==0 || str.indexOf(at)==lstr){ alert("Please check the the
      format of the email addresses you entered.") return false } if (str.indexOf(dot)==-1 || str.indexOf
      (dot)==0 || str.indexOf(dot)==lstr){ alert("Please check the the format of the email addresses
      you entered.") return false } //if (str.indexOf(at,(lat+1))!=-1){ // alert("Please check the the
      format of the email addresses you entered.") // return false //} if (str.substring(lat-1,lat)==dot
      || str.substring(lat+1,lat+2)==dot){ alert("Please check the the format of the email addresses
      you entered.") return false } //if (str.indexOf(dot,(lat+2))==-1){ // alert("Please check the the
      format of the email addresses you entered.") // return false //} if ((str.indexOf(" ")!=-1) &&
      (str.indexOf(" ")!=str.length-1)){ //altered by eme -- a space at the end should be ok alert
      ("Please check the the format of the email addresses you entered.") return false } return true }
      function deleteSpaces(textStr) { // replace any spaces/linebreak characters w/ nothing //var
      textStrSave = textStr.value.replace(/[\t\n\r\f\v ]+/g, ""); var textStrSave =
      textStr.value.replace(/[\s]+/g, ""); textStr.value = textStrSave; } function formSubmit() { if
      (checkStation()==0) { docUrl = document.URL;
      document.frmSendToFriend.title.value=document.title; poundFind = docUrl.indexOf("#"); if
      (poundFind>-1) { docUrl = docUrl.substring(0, poundFind); } if (docUrl.substring(0, 14)
      =="http://npr.org") { docUrl = "http://www.npr.org" + docUrl.substring(14, docUrl.length); }
      document.frmSendToFriend.pageUrl.value = docUrl; from=document.frmSendToFriend.from.value;
      to=document.frmSendToFriend.recipient.value; if ((to==null)||(to=="") || (from==null)||
      (from==""){ alert("Please enter information into the email address fields.");
      document.frmSendToFriend.recipient.focus(); return false; } else if ((echeck(from)==false) ||
      (echeck(to)==false)){ return false } else { document.frmSendToFriend.submit(); } } } function
      checkStation() { //alert("hi"); //alert("|"+document.frmSendToFriend.callletters.value+"|"); if
      ( ((document.frmSendToFriend.callletters.value == "Enter Call Letters") ||
      (document.frmSendToFriend.callletters.value == "")) && (document.frmSendToFriend.localcontact
      [0].checked == true)) { alert ("Please enter the call letters of your local NPR member station if you
      would like to receive information from them."); return 1; } else { return 0; } }</Atom8>
  </Template9>
```

Template5 对应的源文件中的 HTML 代码如下所示：

```
- <p class="text">
  - <a href="javascript:getMedia('ME', '13-Apr-2004', '15', 'RM,WM');">
      <img align="left" alt="audio icon" border="0" height="18"
        src="http://www.npr.org/images/audiospeakericon.gif"
        width="18" />
    </a>
    <a href="javascript:getMedia('ME', '13-Apr-2004', '15', 'RM,WM');">Morning
      Edition audio</a>
  </p>
- <p class="text">
  - <a href="javascript:getMedia('ME', '12-Apr-2004', '14', 'RM,WM');">
      <img align="left" alt="audio icon" border="0" height="18"
        src="http://www.npr.org/images/audiospeakericon.gif" width="18" />
    </a>
    <a href="javascript:getMedia('ME', '12-Apr-2004', '14', 'RM,WM');">Morning
      Edition audio</a>
  </p>
```

可以看出，除了两个 a 元素中的 href 属性不一样之外，其他属性和节点都完全一样，因此，本文自动生成的 Template 为：

```
- <xsl:when test="local-name()='p' and ./@class='text' and a[1]/img[1] and a
    [1]/img[1]/@align='left' and a[1]/img[1]/@alt='audio icon' and a[1]/img
    [1]/@border='0' and a[1]/img[1]/@height='18' and a[1]/img
    [1]/@src='http://www.npr.org/images/audiospeakericon.gif' and a[1]/img
    [1]/@width='18' and a[2]/text()[1]">
  - <Template5>
    - <Atom3>
        <xsl:value-of select="a[1]/@href" />
      </Atom3>
    - <Atom4>
        <xsl:value-of select="a[2]/@href" />
      </Atom4>
    </Template5>
  </xsl:when>
```

Template9 和 Template10 对应的 HTML 源文档中的代码十分复杂，但本文都正确的归纳出了树模板。

6.5 进一步的过滤与转换

查看抽取的结果，会发现出现了很多网页上并没有出现的数据。而且这些数据还不少。这样的数据有四类：

1. HTML 头部的信息。也就是/html/head/下的节点。比如 title。
2. 属性信息。比如 a 元素的 href 属性。
3. 脚本程序。<script>元素内的代码。
4. 注释信息。

不妨称这种并不显示在网页上的数据为**隐藏数据**。隐藏数据并不是用户想表现的数据，虽然也是数据，但是对于分析来说没有多大用处。因此应该过滤掉。例外的是，href 属性信息很重要，因为网页就是依赖 href 信息进行导航的。不过，也要视用途而定，如果是纯粹利用网页中的文本，那么 href 属性也显得没用。另外，title 有时也很重要。

本文首先对抽取规则进行了过滤，使用如下方法进行判断：

```
boolean isXPathValuable(xpath)
   if (xpath.contains("head")) return false;
   if (xpath.contains("@")) return false;
   if (xpath.contains("script")) return false;
   if (xpath.contains ("comment")) return false;
   return true;
```

这样，自动生成的 XSLT 将不包含隐藏数据的抽取规则。

不过，抽取的数据中仍然可能包含 script 元素和注释元素。这是由森林模式生成的。在森林模式中，如果判断一个节点不属于树模板的根节点，那么整颗子树将被全部复制到结果文档里，这颗子树就有可能是 script 元素或者注释元素。

另外，在结果文档中，森林模板下的每一个数据节点都被封装到<var>元素内，需要将它们重新抽取出来组织到一起。但是森林模板下的树模板应被单独抽出来放到根节点下，因为树模板和其他数据并不属于同一逻辑整体。同样，树模板下仅仅将元素节点抽取出来，森林模板则另外放置到根节点下。

本文编写了一个转换结果文档的 XSLT，以进行进一步的过滤和转换，如下所示：

```
<?xml version="1.0" encoding="gb2312" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output encoding="gb2312" indent="yes" />
  - <xsl:template match="/">
    - <out>
      - <xsl:for-each select="//*[contains(name(), 'Template')]">
        - <xsl:element name="{name()}">
          - <xsl:for-each select="*[contains(name(), 'Atom')]">
              <xsl:copy-of select="." />
            </xsl:for-each>
          </xsl:element>
        </xsl:for-each>
      - <xsl:for-each select="//*[contains(name(),'Forest')]">
        - <xsl:element name="{name()}">
          - <xsl:for-each select="*[contains(name(), 'var')][not(comment())]">
              <xsl:copy-of select="node()[not(contains(name(), 'script'))]" />
            </xsl:for-each>
          </xsl:element>
        </xsl:for-each>
      </out>
    </xsl:template>
  </xsl:stylesheet>
```

使用上述过滤后，最终抽取的结果如下所示：

```
<?xml version="1.0" encoding="GB2312" ?>
- <out>
    <Template5 />
    <Template0 />
  - <Template2>
      <Atom1>Expanded Coverage: The Search Engine Wars</Atom1>
    </Template2>
    <Template3 />
  - <Forest4>
      <p class="pagetitle">Search Engine Wars: Google's Cinderella Story</p>
      <b>April 13, 2004</b>
      <br />
      <br />
      In the 1990s, Stanford students Sergey Brin and Larry Page figured out how to use the
      structure of the Internet -- the way pages link to one another -- to put the most relevant
      items at the top of a search list. Their discovery transformed their garage startup, Google,
      into the Internet's top search engine, a household name and even a verb.
      <a href="http://www.npr.org/about/people/bios/rkarr.html">NPR's Rick Karr</a>
      reports.
      <br />
      <br />
      <br />
      <br />
      <br />
      <br />
    </Forest4>
  </out>
```

对比源网页，可以看出，抽取的结果没有任何冗余，这正是本文想要的结果。

6.6 实验结果

本文对十个主要的中文新闻网站进行了测试。对每个网站，选定两个相似的新闻页面作为样本。如表 6.1 所示。主要衡量了四个指标。

1. 冗余模板。冗余模板是指最后归纳出来的抽取规则中仍然包含的部分框架模板。冗

余模板越多，说明框架滤除得越不彻底。相关新闻不算框架。

2. 归纳时间。归纳时间是指由样本的 DOM 树归纳树模板并生成抽取规则所花的时间。
3. 抽取时间。抽取时间是指应用自动生成的抽取规则抽取源文档中的数据所花的时间。抽取过程不包含读取源文档以及生成 DOM 树所花的时间，也不包含写书出文档所花的时间。这一过程仅仅是将源文档的 DOM 树转化为输出文档的 DOM 树。
4. 缩减率。缩减率是指输出文档文本内容大小和源文档文本内容大小的比值。不包括文档中的任何标记。

实验所使用的机器性能指标是：P4 1.7G，256M。

表 6.1 新闻网站测试结果

网站	冗余模板	归纳时间(ms)	抽取时间(ms)	缩减率
Sina	2	1001	62	0.39
Sohu	0	1313	78	0.44
Yahoo	2	1062	78	0.41
tom	0	1250	78	0.41
cri	0	969	62	0.54
Xinhua	0	688	79	0.23
qianlong	0	1313	109	0.44
Southcn	0	1141	78	0.63
Enorth	0	703	63	0.54
Zaobao	1	1062	94	0.75

有三个站点的抽取结果包含冗余模板，分别是 sina，yahoo 和 zaobao。通过分析发现。sina 出现冗余模板是因为存在变化的广告标题。Yahoo 是因为广告模板和文章内容模板是相似的，这使得将广告部分也视作文章内容，不过这部分广告主要是图片，因此对抽取的文本内容影响不大。zaobao 则是因为“上一页”和“下一页”的不同，两篇新闻属于同一系列的不同部分。实际上，通过选定最为相近的样本文档，sina 和 zaobao 中所出现的冗余模板都可以过滤掉。

抽取时间基本上在 1 秒左右。这个时间显得稍长，但是，一旦归纳一次之后，就可以应用抽取规则进行抽取数据了，而不必每次都重新归纳，除非网页结构变化很大。可以看出，抽取时间很短，基本上都在 100 毫秒以内。

缩减率大致在 0.5 左右，也就是说，有一半左右的模板信息被过滤掉了。实际上，缩减率并不绝对。因为每篇新闻的长度都是不一样的，有长的也有短的，而模板信息长度是不会变的。我们所选取的样本都是中等长度的新闻，因此具有一定代表性。

实验证明本文的算法取得了很好的效果。大部分网站都能归纳出完全正确的结果。Yahoo 中由于存在与本文假定不符的情况，因此出现了冗余模板。但是这种情况非常少见，而且，即使是存在这种冗余模板，对抽取结果的影响也不大，因为和文章内容同一个节点下的框架模板大部分是图片之类的广告。

6.7 小结

大部分网页都是由模板生成的。模板中包含了大量与网页内容无关的信息，这些信息严

重影响各种网页分析工作。本文通过比较相似的网页自动的归纳出网页的模板，并自动生成网页主要内容的抽取模式。实验证明本文的方法是有效而且准确的。

7. 自动归纳网页记录模板

7.1 引言

除了网页间的相似性外，一个网页内部也会经常出现相似信息块的现象。比如搜索引擎的搜索结果，每条搜索结果的展现方式都是一样的。本文称这样相似信息块中的一块为一条记录。

一个网页往往包含很多同类型数据。比如导航栏中的每条链接，比如搜索结果的每条记录等等。相同类型的数据一般都用相同的模板显示出来，以达到外观的一致。如图 7.1 所示。

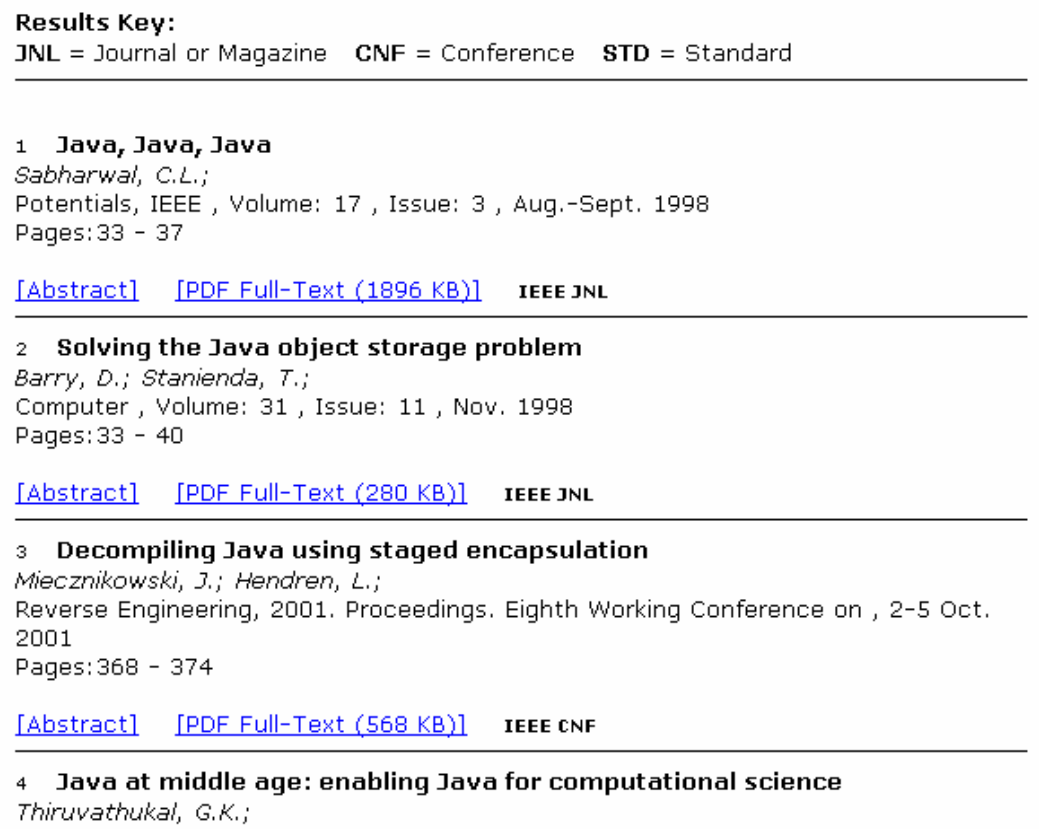


图 7.1 IEEE Xplore 的搜索结果页面

给定一个网页或一组网页。希望自动找出其中的记录模式。进一步可以通过记录模式抽取数据。这种无监督的方式完全不需要人为的标记样本或者编写抽取规则。这样就最大程度的减少了人为的参与。

7.2 相关工作

完全无监督的归纳记录模式这方面的工作做的并不多。

ROADRUNNER[CM03]通过比较同一类网页的两个（或多个）给定的样本页面，归纳

出一个正则文法，不同的文本串泛化为表示数据的#PCDATA，然后就可以生成一个网页所包含的数据模式，正则文法则用来抽取这类网页的信息。它假定所有标签都是由模板生成的。假定没有或关系数据。而这两种假设很多情况下是不成立的。另外，它将网页视为文本流，逐个比较标记，通过各种复杂的启发式算法以归纳模式。这种方式特别敏感，只要网页存在一处不符合假定或者不符合启发式算法的假定，归纳就会失败。

EXALG[AM03]使用了类似的模型。但是归纳方式不一样。它不是逐个比较两个网页中的标记，而是逐步统计最大频繁的连续标记组。这些标记组被作为模板的字符串模式。它假定每个模板的标记组是唯一的，假定每种类型在网页中都出现了很多次，假定不存在会导致形成非法标记组的“正则性”数据，假定数据值之间有分隔符，也就是每个模板都有对应的字符串模式。这种方法比 ROADRUNNER 要健壮一些，但是仍然很敏感，而且假定也会经常失效。

上面的几种方法都是将 HTML 文档视为文本流，完全忽略了 HTML 的 DOM 结构。因此算法极为复杂而且很敏感，效果也不是很好。归纳出来的模式也不健壮，只要网页稍作微小的变动，模式就会失效。而且，这种模式很难人为修改和创建。

[YMT+03]基于网页的 DOM 结构。它试图从一个网页中寻找频繁出现的记录 Schema，从而进行语义划分。它的方法是从叶结点开始，自底向上归纳结构模式。它在归纳模式时没有从全局考虑。比如相同模式的叶结点往往分布在不同的子树下面，这种方法就有可能在自底向上构建模式的过程中将本来相同的模式构建成不同的模式，从而导致归纳失败。它也缺乏对噪声的处理，因此也很敏感。另外，由于没有比较多个网页，因此没能区分模板中的数据部分和非数据部分。

还有一些其它的寻找记录的工作[EJN99, BLP01, LGZ03]，但是他们仅仅是将整个记录段划分开来，并没有归纳记录模式。

7.3 模型和假定

7.3.1 数据类型

本文递归定义了三种数据类型（用 T 描述）：

原子（Atom）类型。用 A 描述。对应于一个字符串。

结构（Struct）类型。用<T1, T2, …, Tn>描述。一个结构类型包含多个域（Field）。每个域代表这个结构的一个属性或说是组成部分。每个域对应一个类型。比如一个长方形类型包含两个域，宽和高。

列表（List）类型。用{T}描述。列表类型的实例是一组元素，这组元素具有相同的类型 T。

使用这三种类型，就可以描述任意复杂的数据类型。

7.3.2 模板

每种数据类型都有对应的模板以展示数据。由于本文视 HTML 文档为 DOM 树结构，因此，模板就对应于一棵树或者一个森林。数据是其中变化的节点或者子树。

本文使用 f 来表示一个森林。定义 F(f1, f2, … fn)为一个森林模板，fi 附加于森林中的一个节点上或者不附加于任何节点。注意，这里并没有指出森林模板为何种形式，如何构建的，

仅仅是指出存在不变的部分，这部分是一个模板，而数据子树则以固定的方式插入到这个模板之中。

对于原子类型 A。它的模板是 F(A)。

对于结构类型<T1, T2, … Tn>。它的模板是 F(F1(T1), F2(T2), … Fn(Tn))。

对于列表类型{T}。它的模板是 F(Ft(T), ….Ft(T))。其中，所有 Ft(T)都具有共同的一个父亲。

一个模板可以有很多个**等价模板**。等价模板是指给定相同的数据，模板生成的结果也完全相同。

结构类型的域可能也是结构类型，本文将子结构类型进行分解，直到结构类型中不包含子结构类型，而只包含原子类型和列表类型。很显然，分解后并没有损失数据，也没有增加新的数据。这样分解的目的是简化模型。

本文并不试图归纳模板，而仅仅是从抽取的角度搜索对对应的抽取模式。下一节本文介绍抽取模型。

7.3.3 抽取模型

本文将 HTML 文档转换为 DOM 树，使用 XSLT 作为抽取规则，抽取结果为一个 XML 文档。XSLT 定义了很多指令可以用于抽取数据。

数据定位

数据抽取其实就是一个数据定位的过程。XSLT 中使用 XPath 进行定位。

对于原子类型，只需要定位到相应的节点就行了。

对于列表类型或者结构类型，则要先定位到子元素共同的父结点。然后针对子元素，进行进一步的定位。

定位使用<xsl:for-each>指令。虽然这条指令返回一组结点，但通过指定的 XPath 可以只返回一个节点。

原子数据的抽取

使用<xsl:value-of>指令。

结构数据的抽取

对于一个结构<T1, T2, … Tn>，首先定位子元素的共同父亲节点，也就是结构的根节点。然后定位各个子元素。

列表数据的抽取

对于一个列表{T}。首先定位子元素的共同父亲节点，也就是列表的根节点，然后使用<xsl:for-each>指令定位每个子元素的根节点。

多子树结构列表数据的抽取

由于 HTML 并不是基于语义的，因此使用树路径时，每一步的节点名并不具有唯一性。例如图 7.2：

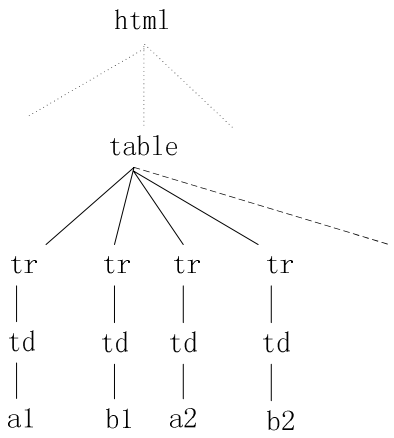


图 7.2 多子树结构列表的一个例子

叶节点都是数据。数据类型为{<a, b>}。table 为列表的的根节点。这个例子中，结构类型的模版是两棵树，而且根节点名相同，这样，就无法用<xsl:for-each>语句来定位结构元素。本文称这种列表元素的模板有多个子树的模型为**多子树结构列表**。要抽取其中的数据，需要将结构类型分解，用[a, b]描述。它等价于{<a, b>}。这样就不必先定位到结构元素，而是直接定位每个子元素。本文首先设定一个变量 index，它从 1 到 n，n 为列表下结构的个数。我们知道每个结构占多少个 tr，然后，根据每个 index，就可以分别定位结构元素中的子元素。由于 XSLT 中没有类似于程序设计语言中的 for 语句，因此，本文使用了特定的技巧以解决这个问题，如例子 7.1 所示：

```
<xsl:for-each select=".../table">
<StructList>
  <xsl:for-each select="tr[(position() - 1) mod 2 = 0]">
  <Struct>
    <xsl:variable name="index" select="position()-1" />
    <xsl:for-each select="../tr[position()=($index*2+1)]">
      <a>
        <xsl:value-of select="td[1]/text()[1]" />
      </a>
    </xsl:for-each>
    <xsl:for-each select="../tr[position()=($index*2+2)]">
      <b>
        <xsl:value-of select="td[1]/text()[1]" />
      </b>
    </xsl:for-each>
  </Struct>
  </xsl:for-each>
</StructList>
</xsl:for-each>
```

例子 7.1 抽取多子树结构列表的方法

7.3.4 简化后的模型

由于结构类型被简化为只包含原子类型和列表类型，而结构列表类型被定义为新的类型，因此，现在的模型为：

- **原子类型**。模板对应于仅仅包含原子数据的一棵树。
- **简单列表类型**。元素类型不是结构类型。也就是说，只包含一种元素。元素的模版是一棵树。列表类型的模板对应于一颗树。根节点的孩子为元素树的排列。
- **多子树结构列表类型**。列表元素的模板为多棵树。还可能存在根节点名相同的树。列表类型的模板对应于一棵树。根节点的孩子为结构元素森林的排列。实际上，可以把简单列表类型看作是一种特殊的多子树结构列表类型。

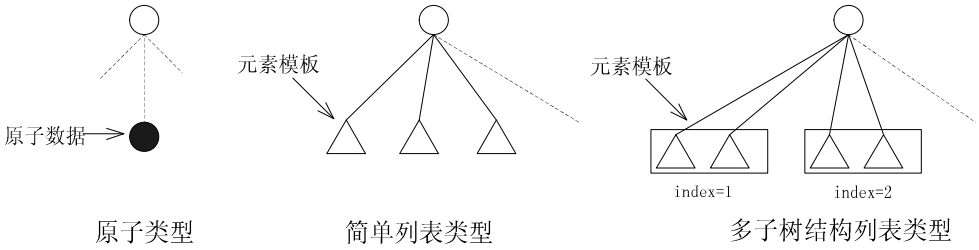


图 7.3 简化后的三种模型

7.4 归纳记录模板

记录的模版的归纳是建立在网页模板归纳的基础之上的。通过比较两颗树，发现变化的数据部分。如果两个网页记录的个数相同，那么，通过归纳树模板，会直接找到每条记录中的每个数据项。由于存在多个记录，因此，那些数据项会出现多次，并且，同类记录中的数据项的树路径是相似的。本文可以根据树路径的相似性来聚类属于同一类记录的数据，并生成记录的抽取模式。下面本文将详细介绍归纳方法及其假定。

7.4.1 列表数据的路径模式

记录其实就是列表。本文首先分析列表数据的路径模式。

根据前面介绍的列表模型，列表元素的模板都是一样的，因而元素中的每一项数据的根节点到元素根节点的路径都是一样的。列表元素的树结构都具有相同的父亲节点，也就是说列表元素根节点的路径除了最后根节点的 index 不一样外，到上层根节点的路径都完全一样。而列表元素中的数据的路径是它到元素根节点的路径和元素根节点的路径到上层根节点路径的和，因此，除了元素根节点处的 index 不一样外，路径的其它部分都是一样的。如图 7.4：

其中，叶节点是文本节点，并且是本文想要的数据。数据模型为{<a, f>}。a 数据节点的路径分别为：

```
a1: p/ol[1]/li[1]/a[1]/text()[1]
a2: p/ol[1]/li[2]/a[1]/text()[1]
a3: p/ol[1]/li[3]/a[1]/text()[1]
. . .
```

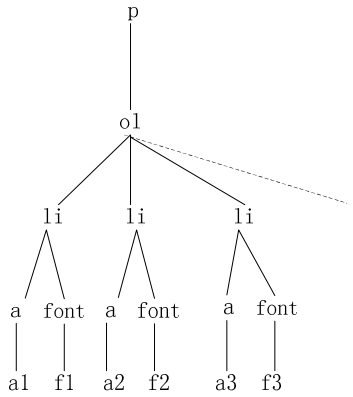


图 7.4 一个列表的 DOM 子树

可以看出，a 数据节点除了 li 节点的 index 不一样外，其它部分完全一样。本文可以得到 a 数据节点路径的模式：

```
a: p/ol[1]/li[1..n, 1]/a[1]/text()[1]
```

[1..n, 1]并不是标准的 XPath 表达式，它是指从 index 为 1 的节点开始，到 index 为 n 的节点结束，每一步到上一步的间隔是 1。例如[2..8, 2]是这样一个序列：2, 4, 6, 8。

本文称 li 为**变化节点**（**varNode**）。它的模型是 **nodename [beginIndex .. endIndex, span]**。变化节点前面的路径 p/ol[1]为**前置路径**（**preXPath**）。变化节点后面的路径 a[1]/text()[1]为**后置路径**（**postXPath**）。前置路径加上变化节点加上后置路径就是一个**路径模式**（PathPattern）。

数据 f 也有类似的路径模式：

```
f: p/ol[1]/li[1..n, 1]/font[1]/text()[1]
```

可以发现，数据 a 与数据 f 的路径模式的前置路径是完全一样的，这是因为它们都属于共同的结构元素，而结构元素具有共同的父亲节点。

7.4.2 树路径聚类与归纳

本文可以根据相同数据元素树路径的相似性来聚类路径，并归纳出数据抽取模式。

7.4.2.1 树路径的相似性

本文定义两条树路径 path1 与 path2 相似当且仅当树路径每一步的节点名都相同。比如下面的例子：

```
p/ol[1]/li[1]/font[1]/text()[1]
p/ol[1]/li[2]/font[1]/text()[1]
p/ol[1]/li[1]/font[2]/text()[1]
```

虽然第二条路径与第三条路径有两个节点的 index 不一样，但是本文仍然认为它们是相似的。

很显然，相同元素的路径是相似的。但是，不同元素的路径也有可能是相似的。

7.4.2.2 相似路径的聚类与归纳

根据树路径的相似性可以将路径进行聚类。每一类中的路径都是相似的，而不同类中的路径是不相似的。如果某个聚类的大小小于 3，本文认为认为这个聚类不代表列表（也就是说一个列表包含至少三个元素），将这个聚类分解。

对于每一类，本文试图寻找对应的路径模式。

给定两条相似路径，路径模式的归纳方法为：从头开始往后匹配路径节点，找到第一个不匹配路径节点节点 ni。两个路径节点匹配当且仅当节点名相等并且 index 也相等。然后从后往头匹配，找到第一个不匹配节点 nj。如果 ni 和 nj 不是同一个路径节点，则归纳失败，否则将 ni 定位变化节点，变化节点的 beginIndex 为较小的 index，endIndex 为较大的 index，span 为两者的差。变化节点前面的路径为前置路径，后面的路径为后置路经。

两条路径就可以归纳出一个路径模式，当加入新的路径时，可以判断新的路径是否匹配路径模式。只有当这条路径的每个节点都匹配路径模式种对应的节点时，这条路径才匹配路径模式。对于变化节点，当一个路径节点的节点名等于变化节点的节点名并且 index 和 beginIndex 的距离是 span 的整数倍时，这个路径节点才匹配变化节点。

路径模式也可以用来继续归纳。归纳时只对路径模式的前置路径进行归纳。比如路径模式 pp1(preXPath1, varNode1, postXPath1)和 pp2 (preXPath2, varNode2, postXPath2)。当 varNode1 等于 varNode2，postXPath1 等于 postXPath2，preXPath1 和 preXPath2 相似时，可以继续归纳 preXPath1 和 preXPath2，得到新的前置路径模式 prep(preXPath3, varNode3, postXPath3)。这样 pp1 和 pp2 归纳得到的新路径模式为 pp(preXPath3, varNode3, postXPath3+postXPath1)。

不断进行这样的归纳，直到不能继续归纳为止。

有可能最终得到多个路径模式。多子树结构列表就很有可能造成这种情况。

本文假定这多个路径模式具有相同的前置路径，相同的变化节点名。本文称这个前置路径为聚类的父亲路径。有时会出现路径模式的前置路径不同的情况，这时本文通过投票确定共同的前置路径。并将其作为聚类的父亲路径。

7.4.2.3 不同元素路径模式的聚合

对于一个列表类型{<a, b>}。a 和 b 的数据的路径会分别聚类并归纳出对应的路径模式。他们的路径模式不同，但是前置路径都是一样的。前置路径其实就是列表根元素的路径。通过将前置路径相同的路径模式聚合到一起，可以得到完整的结构列表路径模式。进一步，就可以自动生成列表数据的抽取模式。

7.4.2.4 示例

对于前面那个图，可以得到所有数据的路径：

```
p/ol[1]/li[1]/a[1]/text()[1]
p/ol[1]/li[1]/font[1]/text()[1]
p/ol[1]/li[2]/a[1]/text()[1]
p/ol[1]/li[2]/font[1]/text()[1]
p/ol[1]/li[3]/a[1]/text()[1]
p/ol[1]/li[3]/font[1]/text()[1]
```

首先进行聚类，得到两个聚类：

```
p/ol[1]/li[1]/a[1]/text()[1]
p/ol[1]/li[2]/a[1]/text()[1]
p/ol[1]/li[3]/a[1]/text()[1]
```

```
p/ol[1]/li[1]/font[1]/text()[1]
p/ol[1]/li[2]/font[1]/text()[1]
p/ol[1]/li[3]/font[1]/text()[1]
```

分别对两个聚类归纳路径模式：

```
p/ol[1]/li[1..3, 1]/a[1]/text()[1]
```

```
p/ol[1]/li[1..3, 1]/font[1]/text()[1]
```

由于这两个模式的 preXPath 都是 p/ol[1]。将这两个路径模式聚合：

```
p/ol[1]
li[1..3, 1]/a[1]/text()[1]
li[1..3, 1]/font[1]/text()[1]
```

变化节点 li 的 span 为 1，beginIndex 为 1，endIndex 为 3。进一步得到：

```
ParentXPath: p/ol[1], VarNode: li[1..3, 1]
li[1..3, 1]/a[1]/text()[1]
li[1..3, 1]/font[1]/text()[1]
```

最终可以生成抽取规则为：

```
<xsl:for-each select="p/ol[1]">
<StructList>
  <xsl:for-each select="li[position()>=1
and (position() -1) mod 1 = 0 and position() <=3]">
  <Struct>
    <xsl:variable name="index" select="position()-1"/>
    <xsl:for-each select="../li[position=($index*1+1)]">
      <Atom1><xsl:value-of select="a[1]/text()[1]"></Atom1>
    </xsl:for-each>
    <xsl:for-each select="../li[position=($index*1+1)]">
      <Atom2><xsl:value-of select="font[1]/text()[1]"></Atom2>
    </xsl:for-each>
  </Struct>
  </xsl:for-each>
</StructList>
</xsl:for-each>
```

首先定位父亲路径 p/ol[1]，接着生成变量 index，从 0 到 2。然后在每个 index 下，分别针对每个元素，定位元素的根节点。比如元素 a 的根节点为../li[$index+1]。最后定位到元素数据，比如 a[1]/text()[1]。这个自动生成的 XSLT 看起来很复杂，但是这种处理方式可以应对任何复杂的情况。

7.4.2.5 校正

当纪录中的某些数据缺失时，路径就会不连续，这时有可能会归纳出不正确的路径模式。比如 beginIndex 不对，span 不对。通过聚合不同元素的路径模式，本文可以从其它元素的模式中获得 span 和 beginIndex 的信息，并根据 span 和 beginIndex 重新对相似路径进行再次的归纳。

7.4.2.6 对噪声的处理与容忍

前面提到在不同元素路径聚合时根据其他元素路径的模式信息来重新进行归纳调整就是一种对噪声的容忍。有时还存在其他的噪声，比如那些偶然相似的路径，对于这种情况，本文对聚类进行限制，当一个聚类中最大的路径模式（匹配的路径最多）的大小小于 3 时，本文认为这个聚类是稀疏的，全部分解。

7.5 实验结果

本文首先拿两个比较复杂的网站进行测试，分别是 IEEE Xplore 和 elsevier 论文搜索网站。它们的论文搜索结果的模板都是多子树结构列表类型，而且子树根节点名相同。不仅如此，这些数据中还包含了很多噪声，但是本文都成功的归纳出了纪录模式，下面是本文的结果：

IEEE 的抽取规则：

```
<?xml version="1.0" encoding="gb2312" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output encoding="gb2312" indent="yes" />
  - <xsl:template match="/html">
    - <Template030>
      - <xsl:for-each select="body[1]/table[4]/tr[1]/td[2]/table[1]/tr[1]/td[1]">
        - <StructList024>
          - <xsl:for-each select="font[position() >= 1 and (position() - 1) mod 3 = 0 and
              position() <= 43]">
            - <Struct025>
                <xsl:variable name="index" select="position()-1" />
              - <xsl:for-each select="../font[position()=($index*3+1)]">
                - <Atom007>
                    <xsl:value-of select="text()[1]" />
                  </Atom007>
                </xsl:for-each>
              - <xsl:for-each select="../font[position()=($index*3+3)]">
                - <Atom009>
                    <xsl:value-of select="text()[1]" />
                  </Atom009>
                </xsl:for-each>
              - <xsl:for-each select="../font[position()=($index*3+3)]">
                - <Atom011>
                    <xsl:value-of select="text()[2]" />
                  </Atom011>
                </xsl:for-each>
              - <xsl:for-each select="../font[position()=($index*3+2)]">
                - <Atom013>
                    <xsl:value-of select="strong[1]/text()[1]" />
                  </Atom013>
                </xsl:for-each>
              - <xsl:for-each select="../font[position()=($index*3+3)]">
```

从 mod 3 可以看出，这是一个多子树结构列表，并且存在三个根节点名都是 font 的子树。

对应的抽取结果为：

```
<?xml version="1.0" encoding="GB2312" ?>
- <Template030>
  - <StructList024>
    - <Struct025>
        <Atom007>1</Atom007>
        <Atom009>Potentials, IEEE,Volume: 17,Issue: 3,Aug.-Sept. 1998</Atom009>
        <Atom011>Pages:33 - 37</Atom011>
        <Atom013>Java, Java, Java</Atom013>
        <Atom015>Sabharwal, C.L.;</Atom015>
        <Atom017>/search/srchabstract.jsp?
          arnumber=714612&isnumber=15498&punumber=45&k2dockey=714612@ieeejrns&query=java&p
        <Atom019>/iel4/45/15498/00714612.pdf?tp=&arnumber=714612&isnumber=15498</Atom019>
        <Atom021>[PDF Full-Text (1896 KB)]</Atom021>
        <Atom023>IEEE JNL</Atom023>
      </Struct025>
    - <Struct025>
        <Atom007>2</Atom007>
        <Atom009>Computer,Volume: 31,Issue: 11,Nov. 1998</Atom009>
        <Atom011>Pages:33 - 40</Atom011>
        <Atom013>Solving the Java object storage problem</Atom013>
        <Atom015>Barry, D.; Stanienda, T.;</Atom015>
        <Atom017>/search/srchabstract.jsp?
          arnumber=730734&isnumber=15756&punumber=2&k2dockey=730734@ieeejrns&query=java&pos
        <Atom019>/iel4/2/15756/00730734.pdf?tp=&arnumber=730734&isnumber=15756</Atom019>
        <Atom021>[PDF Full-Text (280 KB)]</Atom021>
        <Atom023>IEEE JNL</Atom023>
      </Struct025>
    - <Struct025>
        <Atom007>3</Atom007>
        <Atom009>Reverse Engineering, 2001. Proceedings. Eighth Working Conference on,2-5 Oct. 2001</Ato
        <Atom011>Pages:368 - 374</Atom011>
        <Atom013>Decompiling Java using staged encapsulation</Atom013>
        <Atom015>Miecznikowski, J.; Hendren, L.;</Atom015>
```

其中，Atom013 是标题信息，Atom009 是论文出处，Atom015 是作者信息，还有一些其它的信息，包括论文的链接和摘要链接等等。可以看出，网页中的记录数据均被正确的抽取出来。

7.6 小结

大量的网页都是由模板生成的，特别是各种搜索引擎的搜索结果页面，这些页面都包含大量的记录，这些记录的数据是后台数据库的查询结果，具有相同的数据模式。显示每条记录时使用的都是相同的模板，因而每条记录的显示风格完全一样，这导致了网页中出现大量相似的信息块，对应的树结构也是相似的。本文根据树结构的相似性自动归纳出了数据的抽取模式，试验证明本文的方法是成功的。网页中的记录模式不一定完全规范，比如数据的缺失，本文从全局进行分析和归纳以及校正，因此可以有效的容忍噪声，实验证明本文的方法是健壮的。

自动生成的抽取模式不可能百分之百精确，即使是精确的，本文也不一定需要所有的数据，因此，抽取模式的适当修改是必要的。由于本文的抽取规则使用的是 XSLT，因此理解和修改起来特别容易，这比其他方法要好得多。另外，这种模式也比传统的正则表达式的模式要健壮的多，因为依赖的信息少的多。

未来还有很多需要做的工作。首先是根据一些启发式或者人给的知识对抽取的结果进行标记，也可能根据模板中频繁出现的数据。其次是自动定位相似的网页。另外，自动生成查询提交网页表单也是很重要的。当这三个工作都完成了，就可以达到完全自动的获取互联网信息，将其中关键的数据抽取出来并存放到数据库。

8. 多网页信息抽取

8.1 引言

前面所讨论的都是针对单网页进行数据抽取，实际上，单网页中的数据往往只是一部分信息。很多情况下，由于数据量太大，这些数据会分散到多个网页中。当然，数据并不是孤立的，它们通过超链接联系起来，从而形成一个整体。就像一本书，有很多页，首先是目录，通过目录指示的页数可以直接到达每一章、每一节，进而可以获得下一页数据。要获取整本书的数据，必须对所有页面进行抽取。同样，对于大多数网上的数据，要获取想要的数据往往需要对多个网页进行抽取。可惜 XSLT 并不直接提供多文档的转换功能。尽管有 document() 函数可以引入其他的文档，但是功能很弱，并不适合多网页信息抽取。

对于大量的数据，也不适合直接存放为 XML 文档，一般是存放到数据库中，这样可以利用数据库的各种特性查询和操纵数据。

对此，本文开发了一个多网页信息抽取框架，提供了一个脚本语言，将 XSLT 形式的抽取规则和程序设计语言有机的结合起来，从而可以方便的开发各种抽取网上信息并存放到数据库中的应用。

8.2 模型和框架

8.2.1 问题描述

本文将问题抽象为图 8.1 所示的图。总共有三层页面，第一层页面包含很多数据，同时

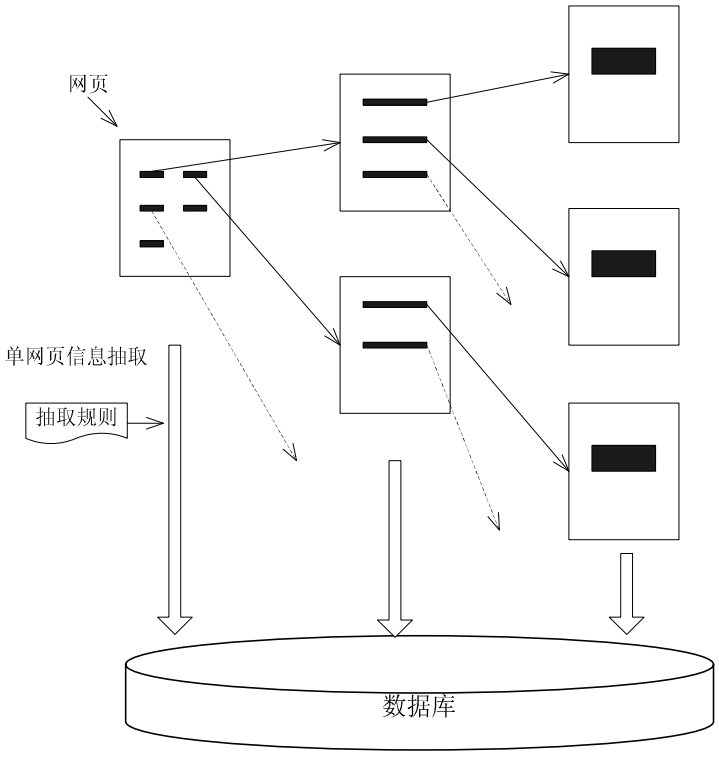


图 8.1 多网页信息抽取

每条数据包含指向下一层对应数据所在的页面的链接，第二层数据包含指向第三层数据页面的链接，第三层不再包含指向更多数据的页面的链接。当然，实际问题不一定恰好是三层，有可能是两层，也有可能有更多层。每一层的数据模式是一样的，对应的网页结构也是一样的。对于一个 N 层的问题，最多只需要编写 n 个抽取规则。本文希望将所有层的数据都抽取出来，同时数据之间的联系也被保留下来。

8.2.2 抽取框架

对于多网页信息抽取，每一层的数据处理模型都是一样的，如图 8.2 所示：

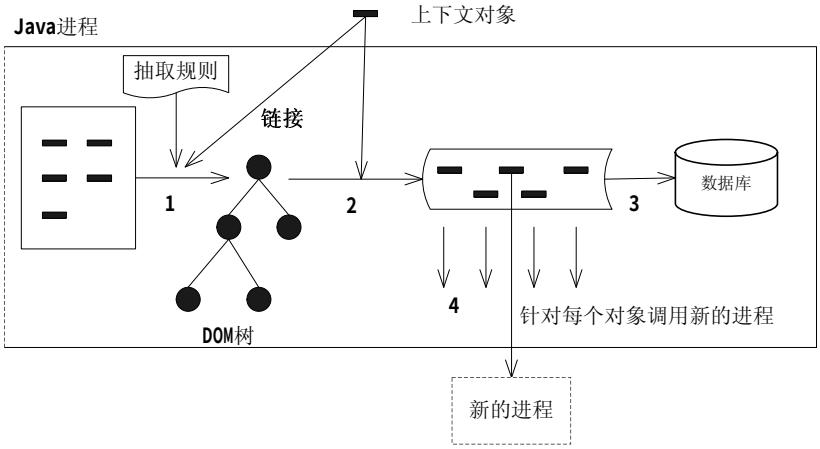


图 8.2 每一层的处理模型

针对每一层，关联一个 Java 进程。这个 Java 进程负责处理信息抽取，把数据存放到数据库，以及激发新的进程。

网页中的数据本身是孤立的，要将多个网页间的数据关联起来，必须针对每个网页提供一个**上下文对象**，这个上下文对象包含了这个网页中的数据的上下文信息。比如一篇文章所属的类别，是属于体育新闻还是娱乐新闻？尽管文章网页中也可能提供了这样的上下文信息，但根据网页本身获取上下文信息并不可靠，因为上下文信息的抽取模式并不固定，还可能根本不存在。上下文对象一般是抽取过程中前一步的对象，是通过这个对象激发了下一步抽取工作。

上下文对象必须包含一个链接，或者其他数据以指示剩余的信息所在的页面，以便进行完整的抽取。

现在本文详细解释一个典型的 Java 进程：

1. 首先，抽取上下文对象链接所指向的网页中的数据。数据为 DOM 数据结构。
2. 根据上下文对象，将 DOM 数据结构转化为特定类别的数据对象。
3. 将所有数据对象存储到数据库中。
4. 针对每个数据对象激发新的 Java 进程。数据对象作为新的 Java 进程的上下文对象。

当到达最后一层时，就没有第四步了。另外，第三步也有可能推迟到下一层才进行处理，因为有时数据还不完全，需要下一层的数据才能存放到数据库中。

抽取参数

每一层的抽取模式是一样的，但是参数并不一样。在激发一个新的 Java 进程时，除了

提供上下文对象外，还至少需要提供抽取模式这样的参数。完整地说，有以下五个参数：

1. 自身的 ID。
2. 抽取模式文件名。
3. 抽取对象的名称。有时一个抽取模式会抽取出很多对象。
4. 下一步 Java 进程的名称。
5. 下一步抽取参数的 ID。

当到达最后一层时，没有最后两个参数。配置文件的格式如下所示：

```
ecs.xml 抽取参数配置文件
<ExtractConfigSeries>
<ExtractConfig id="Category">
   <OER>extraction rules filename</OER>
   <EID>extraction object id</EID>
   <ProcessMethod>process id</ProcessMethod>
   <NextExtractConfig>next config id</NextExtractConfig>
</ExtractConfig>
<ExtractConfig id="..">.. </ExtractConfig>
..
</ExtractConfigSeries>
```

根据第四步，还需要提供进程名称到类的关联。格式如下：

```
   pmf.xml 进程绑定配置文件
   <ProcessMethodFactory>
     <ProcessMethod>
       <id>id</id>
       <class>classpath</class>
     </ProcessMethod>
     <ProcessMethod>..</ProcessMethod>
     ..
   </ProcessMethodFactory>
```

每个进程类都跟应用相关，但都必须实现下面的接口：

```
   public interface ProcessMethod {
      public void process(Object o, ExtractConfig ec);
   }
```

o 是上下文对象。ExtractConfig 对应于抽取参数。包含抽取规则文件名，抽取对象 ID，下一步进程对象，下一步抽取参数。

框架会自动装载每个进程类，根据配置文件。然后，在应用程序中就可以获得进程类的对象了，进而调用对应的 process 函数激发进程。

8.3 小结

本章介绍了多网页信息抽取。本文开发了一个多网页信息抽取框架，基于这个框架，可

以很容易的开发各种实际的信息抽取应用。而且，由于抽取规则以及抽取参数和代码是分开的，即使是网页变化了甚至是网站结构变化了，应用程序代码都不需要改写。

9. 总结和未来的工作

9.1 总结

随着互联网的迅猛发展，信息过载已经成为一个严峻的问题。虽然搜索引擎可以帮助人们方便的检索信息。但是它只提供了很简单，很粗糙的检索方式，远远没有达到理想的状态。理想的情况是，互联网上的信息可以像数据库一样被查询。但是互联网是基于 HTML 的，而 HTML 文档并不是结构化的文档，并不能直接用来查询。因此，一个很自然的想法是将 HTML 文档中的数据抽取出来并存放到数据库中。

虽然信息抽取很早就被提出来了，但是传统的信息抽取只是针对纯文本文档，并不适合 HTML 文档。90 年代末期 Kushmerick 提出了 Wrapper Induction。试图通过人为标记网页中要抽取的数据来学习抽取模式。数据的定位是基于一系列关键词的逐个匹配。这之后，各种机器学习的 Wrapper Induction 算法被提出。除了机器学习的算法之外，还有各种其他的方法被提出，比如基于 Ontology 的，基于自然语言理解的，基于 HTML 的等等。[Eik99][Mus99]和[LRST02]提供了很好的概览。

这些方法中，每种方法都有各自的优缺点。基于 Ontology 的方法最为通用，但是编写知识库并不容易，很多情况下根本找不到合适的模式。基于机器学习的方法也不实用。这些方法都很难保证百分之百的精确。只有基于 HTML 的方法利用了 HTML 文档本身的结构，抽取规则最为简单有效。但是通用性程度并不高，而且基于路径的抽取规则也很敏感。

随着近年来 XML 的推出和发展，各种相关的标准和工具被制定和开发。基于 XML 的应用开发变得很简单，而且健壮性和互操作性更好。虽然 XML 是针对 HTML 的弱点而制定的，但是 XML 不仅没有取代 HTML，反而为操纵 HTML 文档提供了更加灵活强大的机制。

本文在第三章中详细介绍了使用 XSLT 编写简单、健壮和通用的抽取规则的方法。本文还开发了一个工具，使得抽取规则的编写极其简单容易。

大多数网页都是由模版生成的，因此网页和网页间，网页内部的信息块间都存在相似的外观和模式。基于这种相似性，本文研究了两种模版归纳方法，分别针对网页间的模版和网页内部的记录模版。归纳网页模版很有意义，可以大大改善信息检索、文本分类和聚类等各种基于网页内容的应用。记录模版的归纳一般是为了自动生成抽取规则。但是自动化的方法都很难达到百分之百的精确，因为没有人为的参与，也就不知道具体的需求。由于本文的抽取规则都是基于 XSLT 的，因此可读性和可修改性都很好。

在实际的应用中，往往需要对多个网页进行抽取才能获得完整的数据。第五章本文介绍了多网页信息抽取方法和框架。基于本文的框架，可以很快开发一个健壮的多网页信息抽取应用。

9.2 未来的工作

本文已经开发了网页信息抽取的方法、工具和框架，基于本文的平台，可以开发各种网页信息抽取的应用。未来主要有两方面的工作：

1 扩展信息抽取平台

虽然目前平台已达实用化的阶段。但仍然有很多值得改善和加强的地方。

1) 更精细的文本抽取。

HTML 文档虽然有很多标记将文本分隔开来，但并不完全。仍然有很多大块的文本，这些文本除了包含要抽取的数据之外，还包含一些描述性的文字，比如“价格”。而这些描述性的文字并不是我们真正想要的信息，应当被过滤掉。因此，还必须开发基于文本的模式语言来描述这种模式并抽取其中的数据。

2) Focused Crawling

未来基于主题的信息服务将会得到广泛的发展。搜索将会朝着专业化的方向发展。而要获取互联网上所有的与主题相关的信息，首先就需要 Focused Crawling，然后才能进行信息抽取。虽然我们已经开发了招聘信息搜索系统，但是其方法还不是很通用，不能胜任很多其它的任务。还需要进一步开发通用的 Focused Crawling 方法和架构。

2 开发基于信息抽取的各种应用

1) 信息助手

每个人每天都会上固定的网站看一些东西，比如体育新闻，比如娱乐新闻。上的网站经常有多个，因为要关心不同方面的消息。而另一方面，在一个网站上用户不会察看所有的消息，而是只是看固定它所关心的那个部分。用户每次都必须重复一些工作，不断打开浏览器，不断连接页面，不断拖动滚动条一定为自己想看的内容。这些工作虽然很简单，但是很繁琐，很耗时。我们希望开发一个系统自动的跟踪用户的上网模式，自动抽取用户想看的那部分信息，自动将内容下载到本地，以方便用户查看新闻。就像 foxmail 和 outlook 这样的邮件客户端一样，消息被定时被下载到本地，而不需要连接网站，那样会很浪费时间。当然，所涉及到的技术是完全不一样的。电子邮件客户端要做的只是遵照标准，定时获取就行了。但是，我们的问题是，

信息完全从互联网上获取，没有标准的格式和协议。这要依靠信息抽取。

要获取哪些信息并不是固定的。每个用户都不一样。每个用户都有自己信息浏览模式和感兴趣的信息。而这些东西并不是像邮件服务器事先放好在某一个地方的。首先，用户经常上哪些网站？二，在网站上，用户经常看那一部分内容？这需要进行客户端的上网模式挖掘。

2) 新闻聚合

新闻网站很多，有综合的新闻网站，也有特定主题的专业新闻网站。综合的新闻网站新闻多而全，但专业新闻网站的新闻少而精。即使都是综合网站，每个网站也都有各自的特色。新闻经常会被转载，或者经常各处都出现相同的新闻，新闻聚合将可以帮助找住最流行的新闻。

并不是所有的领域都有对应的新闻网站。有很多专业新闻网站的消息都是转自其它的覆盖面更大的新闻网站。新闻聚合可以帮助自动建立特定主题的新闻网站。

任何新闻网站，人们不会去关心他的所有新闻，他只关心他所感兴趣的那一部分。简单的分类并不能满足用户特定的爱好。通过挖掘用户喜好，结合新闻聚合，可以自动建立用户感兴趣新闻网站。

要进行新闻聚合首先就需要将所有网站的新闻每天自动抽取下来。

3) 企业竞争情报系统

竞争情报对企业来说无疑是很重要的信息。而互联网的广泛应用使得大量的竞争情报可以直接从网上获得。但人为的收集情报是一件费时费力的工作。企业竞争情报系统就是自动

的从互联网上获取特定企业所关心的信息，并及时通知企业。企业竞争情报有各个方面，比如关于企业的最新新闻，比如竞争对手的最新产品信息，最新招聘信息，最新专利等等。而这些信息的获取和监视都要依赖于信息抽取。

4) 专业（垂直）搜索引擎

通用搜索引擎虽然搜索面广，但是搜索结果并不精确，因为有大量不相关领域的信息。专业搜索引擎只索引特定领域的信息，从而提供更精确的搜索结果。同时，还可以根据专业的特点，提供专业的搜索服务和信息表示。虽然通过分类可以过滤出特定领域的文档，但并不能进一步提供专业的搜索服务。还需要进行信息抽取，将各种特定领域的相关信息抽取出来，这样才能各种专业的信息服务，这也是通用搜索引擎难以达到的深度。

5) 信息集成

互联网上有各种搜索服务。信息集成就是将各种搜索服务集成起来，提供一个单一的搜索界面给用户，用户就不需要依次使用各个搜索引擎。虽然 Web Service 提供了一个很好的架构，但是仍然有大量的网站并不提供 Web Service。这就需要自动提交表单并抽取结果页面的数据。

参考文献

[AM03] Arvind Arasu, Hector Garcia-Molina: Extracting Structured Data from Web Pages. ICDE 2003: 698-698, 2003.

[BLP01] David Buttler, Ling Liu, Calton Pu: A Fully Automated Object Extraction System for the World Wide Web. ICDCS 2001: 361-370.

[CL01] Chia-Hui Chang and Shao-Chen Lui. IEPAD: Information extraction based on pattern discovery. In Proc. of 2001 Intl. World Wide Web Conf., pages 681–688, 2001.

[CM03] Valter Crescenzi and Giansalvatore Mecca. On Automatic Information Extraction from Large Web Sites. Technical Report DIA-76-2003.

[CM99] Mary Elaine Califf, Raymond J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. AAAI/IAAI 1999: 328-334.

[Dav00] Brian D. Davison. Recognizing Nepotistic links on the Web. Proceeding of AAAI 2000.

[DOM] Document Object Model, W3C Recommendation October, 1998. http://www.w3.org/DOM/.

[ECJ+99] David W. Embley, Douglas M. Campbell, Y.S. Jiang, Stephen W.Liddle, D.W. Lonsdale, Y.-K. Ng, Randy D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. Data and Knowledge Engineering 31, 3, (1999), 227-251.

[Eik99] Line Eikvil. Information Extraction from World Wide Web A Survey. Norweigan Computing Center Technical Report 945, 1999.

[EJN99] David W. Embley, Y. S. Jiang, Yiu-Kai Ng: Record-Boundary Discovery in Web Documents. SIGMOD Conference 1999: 467-478.

[FLM98] Daniela Florescu, Alon Levy, Alberto Mendelzon. Database Techniques for the World Wide Web: A Survey. SIGMOD Record 27:2 pp. 59-74, 1998.

[Fre00] Dayne Freitag: Machine Learning for Information Extraction in Informal Domains. Machine Learning 39(2/3): 169-202 (2000).

[HD98] Chun-Nan Hsu, Ming-Tzung Dung: Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. Information Systems 23(8): 521-538 (1998).

[HTML] HyperText Markup Language, W3C Recommendation, December 1999. http://www.w3.org/TR/html401/.

[Kus00] Nicholas Kushmerick: Wrapper induction: Efficiency and expressiveness. Artificial Intelligence 118(1-2): 15-68 (2000)

[Kus99] Nicholas Kushmerick. Learning to remove Internet advertisements. Agent-99, 1999.

[LGZ03] Bing Liu, Robert L. Grossman, Yanhong Zhai: Mining data records in Web pages. KDD 2003: 601-606.

[LH02] Shian-Hua Lin and Jan-Ming Ho. Discovering Informative Content Blocks from Web Documents. KDD-02, 2002.

[LPH00] Ling Liu, Calton Pu, Wei Han. XWRAP: An XML-enable Wrapper Construction System for Web Information Resource. In Proceedings of the 16th IEEE International Conference on Data Engineering (San Diego, California, 2000), pp. 611-621.

[LRST02] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, Juliana S. Teixeira. A Brief Survey of Web Data Extraction Tools. SIGMOD Record, Vol. 31, No. 2, pp. 84-93, June 2002.

[MJ03] Jussi Myllymaki, Jared Jackson. Robust Web Data Extraction with XML Path Expressions. IBM Research Report, 2003.

[MMK01] Ion Muslea, Steven Minton, Craig A. Knoblock: Hierarchical Wrapper Induction for Semistructured Information Sources. Autonomous Agents and Multi-Agent Systems 4(1/2): 93-114 (2001).

[Mus99] Ion Muslea. Extraction Patterns for Information Extraction Tasks: A Survey. Proceedings of the American Association for Artificial Intelligence (www.aaai.org), 1999.

[SA99] Arnaud Sahuguet, Fabien Azavant. Wysiwyg Web Wrapper Factory. WWW8, 1999,.

[Sod99] Stephen Soderland: Learning Information Extraction Rules for Semi-Structured and Free Text. Machine Learning 34(1-3): 233-272 (1999).

[TIDY] HTML Tidy. http://www.w3.org/People/Raggett/tidy/.

[XALANJ] Xalan-Java. http://xml.apache.org/xalan-j/.

[XHTML] XHTML: The Extensible HyperText Markup Language, W3C Recommendation, January 2000. http://www.w3.org/TR/xhtml1.

[XML] Extensible Markup Language (XML), W3C Recommendation, February 1998. http://www.w3.org/TR/REC-xml.

[SCHEMA] XML Schema Part 0: Primer, W3C Working Draft, April 2000. http://www.w3.org/TR/xmlschema-0/.

[XPATH] XML Path Language (XPath), W3C Recommendation, November 1999. http://www.w3.org/TR/xpath.html.

[XSLT] XSL Transformations (XSLT), W3C Recommendation, November 1999. http://www.w3.org/TR/xslt.html.

[YLL03] Lan Yi, Bing Liu, Xiaoli li. Eliminating Noisy Information in Web Pages for Data Mining. KDD-03, 2003

[YMT+03] Guizhen Yang, Saikat Mukherjee, Wenfang Tan, I.V. Ramakrisnan, Hasan Davulcu. On the Power of Semantic Partitioning of Web Documents. In IJCAI'03 Workshop on Information Integration on the Web, August 9-10, 2003.

[YR02] Ziv Bar-Yossef and Sridhar Rajagopalan. Template Detection via Data Mining and its Applications. WWW 2002, 2002.

攻读硕士学位期间发表论文

1. 朱明、周津、王继康，一种新的基于击键特性的用户识别方法，计算机工程 Vol. 28, No. 10，2002
2. 尹大成，周津，朱明，电信综合网管数据模型有效实现方法研究，计算机应用与软件，已录取。
3. 周津，朱明，王胜，基于 XML 的网页信息自动抽取，计算机应用。已录取
4. 朱明，王胜，周津，基于 Web 企业竞争对手情报自动搜集平台，微计算机应用 Vol. 25, No. 1, 2004。