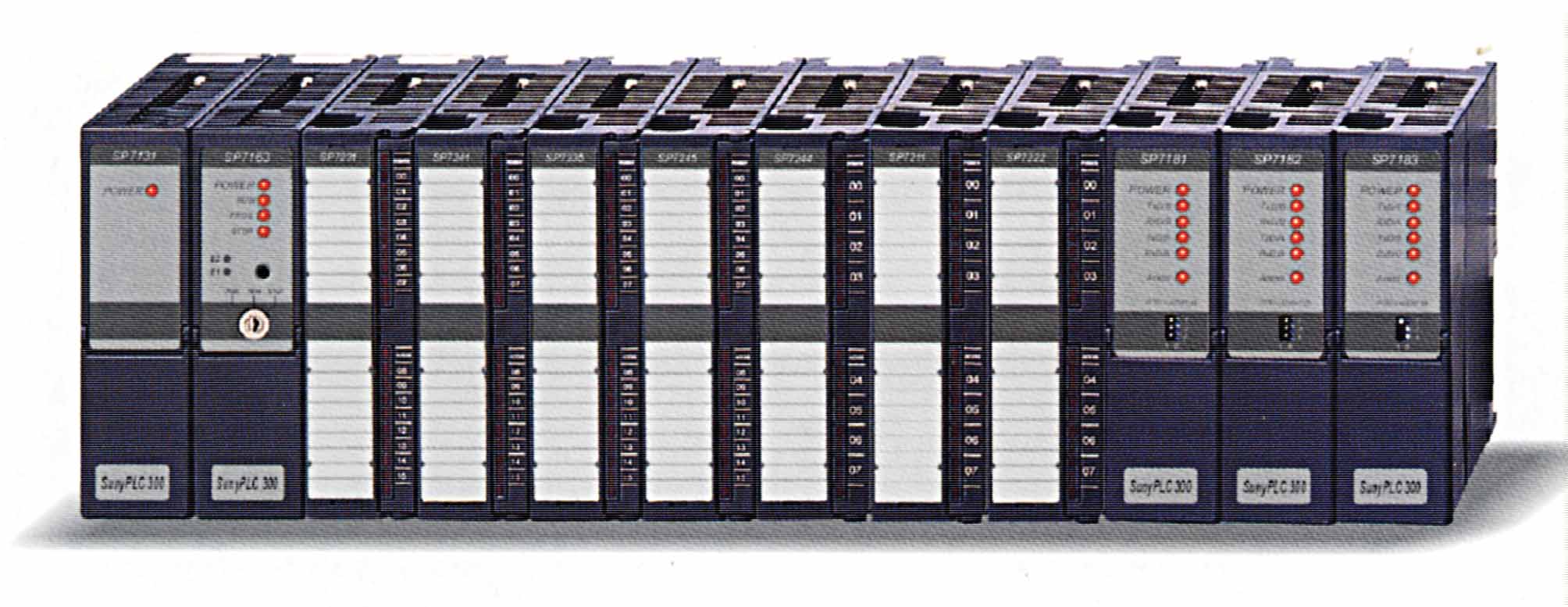
|  |
| --- |
| 信息抽取的系统解决方案 | yingrui |



|  |  |
| --- | --- |
| yingrui | Websiteschema与网络信息抽取 |

目录

[**前 言** 3](#_Toc315552721)

[**第一部分  4篇论文** 4](#_Toc315552722)

[1. Web信息抽取系统 5](#_Toc315552723)

[2. HTML文档的标题抽取及其对网页检索的应用 14](#_Toc315552724)

[3. VIPS基于可视化信息的网页分块算法 20](#_Toc315552725)

[4. 网页布局的数值化分析方法与自动信息抽取 26](#_Toc315552726)

[**第二部分 工程篇** 31](#_Toc315552727)

[5. 软件集成框架与功能块 32](#_Toc315552728)

[6、信息抽取系统的架构 37](#_Toc315552729)

[7、功能块的实现 41](#_Toc315552730)

[**第三部分 技术篇** 42](#_Toc315552731)

[8、识别网页结构与配置 43](#_Toc315552732)

[9、采集程序基本功能块 44](#_Toc315552733)

[10、虚拟设备与URL队列 45](#_Toc315552734)

[11、站点、起始URL、任务和调度 46](#_Toc315552735)

[12、配置一个网站 47](#_Toc315552736)

[附A：Extjs+Spring+DWR+Mybatis与元数据服务器 48](#_Toc315552737)

[附B：对HBase的封装 49](#_Toc315552738)

[附C：XPath与DOM 50](#_Toc315552739)

**前 言**

信息抽取技术作为信息检索、知识挖掘等应用的基础，其进展已经到了相当成熟的程度，国内外有很多产品和系统设计来进行信息采集和抽取。然而其市场状况却非常混乱，如何才能设计出一款稳定、简单的产品，可以真正将信息抽取技术应用到实际的工程项目中呢。

信息抽取在信息自动处理系统的位置？

信息抽取面对的难题是什么？

信息抽取未来的方向在哪里？

**第一部分  4篇论文**

这一段我通过介绍一些论文来介绍网络信息抽取技术，完全没有原创的观点，所做的贡献就是将前人的观点整理一下，希望对大家有用。

通过解读这几篇论文，可以让大家了解到当前学术界对信息抽取的研究热点、研究方向和研究方法等。当然我不会完整的翻译全部的论文，所以仅仅是摘取其中的若干片段，希望也能为初入门者打开窥探的视角。下面介绍的这篇论文是关键词“web information extraction”的第一个google返回结果，论文名称为《A Survey of Web Information Extraction Systems》。

### 1. Web信息抽取系统

什么是Web抽取系统，如何评价抽取系统？如何系统的认识一个信息抽取系统，这必将全面的影响我们原先对所谓的爬虫、蜘蛛的看法。

#### [*Web信息抽取系统综述*](http://in2.csie.ncu.edu.tw/~chia/pub/iesurvey2006.pdf)

***作者：***[***Chia-Hui Chang***](http://in2.csie.ncu.edu.tw/~chia/)*,****Mohammed Kayed****,****Moheb Ramzy Girgis****,****Khaled Shaalan***

***摘要：****Internet存储了大量有用的信息，但是这些信息的格式不同，这使得从不同来源抽取出相关信息变得困难。因而，信息抽取系统在将网页转换成机器可读数据的能力，主要是指程序的鲁棒性和灵活性，就成为了衡量该系统的主要指标。虽然已经开发出了很多方法用于网络信息抽取，但是作者有限的努力只能比较其中数种，而且这些方法因为抽取任务的不同，也只有很少一部分结果可以进行比较。本文调研了主要的网络数据抽取方法，并在方法的适用领域、自动化程度和采用技术等三个维度进行综合的比较研究。第一个维度的标准解释了为什么有的信息抽取系统不能处理一些结构特殊的网站，第二个维度的标准基于系统采用的技术，对抽取系统进行了分类。第三个维度的标准衡量了抽取系统的自动化程度。我们相信这些标准可以很好的衡量不同的信息抽取方法。*

***关键词：信息抽取****，****Web挖掘****，****抽取器****，****抽取器生成***

在摘要中明确提出了要从适应性，自动化程度和使用的方法来评价一个信息抽取系统，那么我们来看，一个信息抽取系统面临的是什么问题，他的产生背景是什么呢？

*万维网的爆炸性增长和流行，在因特网上产生了大量的信息。然而，由于网络资源的特异性以及缺乏固定结构，访问这些资源的主要方法被限制为浏览和搜索。一些智能的网络挖掘应用，例如价格比较机器人，需要对数据格式的维护付出进行大量的维护成本。为了能够将网页转换成结构化数据，人们已经在信息抽取（Information Extraction IE）领域付出了巨大的努力。不同于信息检索（Information Retrieval IR）仅考虑如何从文档集中找出相关文档，IE必须生成结构化数据用于其他数据处理，这对于许多Web挖掘和搜索应用都是决定性的。*

文中描述了Web IE是很多网络挖掘和搜索工具等应用的关键技术，就像目前的团购网站，新闻搜索，价格搜索等等。那么我们接着来看Prof.Chang是怎么定义一个IE任务呢。

*通常，一个IE任务由输入和其抽取目标两部分组成。输入可以是无结构的文档，可能是自由文本，也可能是网络上的半结构化文档，就像表格、项目清单或者列表。抽取目标可以被形式化为具有一定关联关系的k元组（k是一个记录的属性的数量），也可以被形式化为一个复杂对象，其具有树形结构或者嵌套有其它复杂对象。并且很多IE任务中，一个抽取目标的属性可能允许没有值或者有多值。更为复杂的问题还包括：在输入文档中，不同属性之间的排列组合顺序，以及一些印刷错误等。*

既然一个IE任务是将输入转换成结构化的数据，那么接下来我们将要定义什么是信息抽取系统和抽取器了，请看接下两段：

***抽取器****(Wrapper或者Extractor)被定义为：实现IE任务的计算机程序。抽取器最初被定义为信息集成系统（Information Integration System）的一个组件，他是一个特定的查询接口，用来以固定的方式访问不同的信息源。通常情况下，在一个信息集成系统中，抽取器是一个程序，它将各种信息源（如：数据库，网络资源）转换成通用信息源，使得信息集成系统可以直接访问各种数据而不需要改变其原有的查询机制。举例如下：当某信息源为Web服务器，一个抽取器必须通过HTTP协议查询此Web服务器并抽取和收集网页上的数据，最终将这些数据和其它信息源上的数据集成在一起。*

***抽取器生成系统****(Wrapper Induction WI System 或者 Information Extraction IE System)是一种被设计来生成抽取器程序的程序。一个抽取器通常表现为一个模式匹配的过程，或者是一种有限状态机，它依赖固定的规则执行IE任务。针对特定任务开发设计的一个WI系统，主要依赖于任务的数据类型，领域特点和应用场景。为了最大化系统的重用性和最小化系统的维护成本，设计一个可以训练的WI系统一直是信息理解、机器学习、数据挖掘等研究领域的重要课题。*

现在我们知道了，一个抽取器生成系统，又称信息抽取系统，主要是被设计来生成采集器，而采集器通常是依照规则运行的程序，所以生成的采集器，通常就是生成一些规则，或者配置。因为这篇论文是介绍Web IE系统的，Prof.Chang接着又说明了Web IE系统和传统IE系统的不同特点，那让我们接着来看一看吧。

*Web IE任务和传统的IE任务有很大的不同，传统的IE任务致力于从自由文本中抽取出信息，然而相反的，Web IE系统处理的在线文档通常都是半结构文档，并且绝大部分是由服务端程序自动生成的。其结果是，传统IE系统主要借助NLP技术实现，例如词法分析和语法分析技术，而Web IE技术通常应用机器学习和模式挖掘等技术去查找文档中的符号标记或者层次结构等模板信息。*

我们进一步知道了Web IE系统的特点，但是我个人相信，传统IE系统中的使用的NLP技术，对Web IE来说，也是同样需要的，例如从新闻的正文中抽取出作者、来源等信息。接下来让我们了解一下在2006年，信息抽取系统的状况吧。当时许多基于机器学习和模式挖掘技术的IE系统都已经被开发出来了。

[*消息理解会议*](http://en.wikipedia.org/wiki/Message_Understanding_Conference)*（Message Understanding Conference MUC）最初推进了IE系统的发展。其时有5个主要任务与text IE有关，包括命名实体识别、指代消减和指定元素、关系和情景识别。MUC在IE领域发展的时候，推动了研究人员将IE方法分为了两种不同的方法：MUC方法和后MUC方法。其代表分别有，MUC方法：AutoSolg、LIEP、PALKA、HASTEN、CRYSTAL；后MUC方法：WHISK、RAPIER、SRV、WIEN、SoftMealy、STALKER。*

*Hsu和Dung将抽取器分为4类，包括：使用通用编程语言编写的抽取器、特殊设计的领域编程语言或工具、启发式抽取器和抽取器生成系统。Chang依照这种分类，再结合用户观点对相应WI系统进行比较，再根据这些系统的自动化程度，也归纳了4种分类：程序员依赖的信息抽取系统、需要标注语料基于机器学习的信息抽取系统、不需要标注语料的无监督信息抽取系统，半监督的信息抽取系统*

*Muslea维护了RISE网站（Repository of Online Information Sources Used in Information Extraction Tasks 在线信息抽取任务的信息源数据库）。按照IE任务的输入文档和抽取器规则，将信息抽取系统分为3类：第一类包括所有按照符号标记为模式对自由文本进行抽取的系统。第二类为以剔除规则为主的，并主要针对HTML文档的信息抽取系统。第三类也是针对在线文档，但是同时包含剔除规则和符号标记。*

PS：我忍不住要在这里插上一句，我们发现，居然有人按照抽取器规则和输入文档类型就对抽取器归了3类。到目前为止，我们是否可以认为：所有的抽取规则仅限于基于符号标注和剔除规则两种呢？剩下的抽取器就是那些自己写代码，一个程序针对一种网页的抽取器呢？

*Laender提出了按照抽取系统生成抽取器的技术来对信息抽取系统进行分类。其中包含几个维度有：开发抽取器使用的语言（例如：Minerva，TSIMMIS，WebOQL）、与HTML语言相关（例如：W4F，XWrap，RoadRunner）、基于NLP技术（例如：WHISK，RAPIER，SRV）、以生成抽取器为主的（例如：WIEN，SoftMealy，STALKER）、基于模型的（例如：NoDoSE，DEByE）、基于本体的（例如：BYU）。Laender从7个方面对这些系统和工具进行了对比：自动化程度、支持复杂对象、文本内容、GUI友善度、XML输出、支持非HTML信息源和适应能力。*

*Sarawagi将HTML抽取器依照抽取任务的不同将IE系统分为3类。第一类是记录级别的抽取器，抽取系统发现网页上每条记录的边界，然后由页面上抽取出记录列表。第二类是页面级别的抽取器，每个目标页面有多种记录。第三类是网站级别的抽取器，目的是将整个网站转录入一个本地数据库。*

*Kuhlins和Tredwell按照商用和非商用能力对IE系统进行了归类，并按照系统的输出方法、接口类型、网络采集能力和界面化人机接口对抽取系统进行了对比。*

*本篇综述将按照3个主要维度评价IE系统。第一是依据输入文本的不同，主要考虑的是IE任务的难度。第二是依据抓取器使用的技术进行分类，例如使用正则表达式、逻辑规则、有限状态机、概率模型等等。第三是根据程序员的参与程度，学习曲线或者无监督自学习等方面，主要考虑系统的自动化程度。*

其后则整篇文章都在介绍各种被Prof.Chang选择的抽取器，我想打乱一下原文的讲解顺序，首先将其针对输入类型而进行对比的结果翻译并列举出来，然后再选择几个有代表性的抽取器进行稍微详细的介绍。

表1：基于IE任务适用范围的对比分析

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **名称** | **页面类型** | **非HTML页面** | **数据等级** | **抽取目标数据模型的限制** | | | **字段的格式变化** | | **能否进一步抽取属性** |
| 空值/多值 | 字段顺序不定 | 嵌套复杂对象 | 一个字段多种格式 | 多种字段统一格式 |
| **手工配置** | Minerva | 半结构化 | 支持 | 记录级别 | Yes | Yes | Yes | Both | By Order | 支持 |
| TSIMMIS | 半结构化 | 支持 | 记录级别 | Yes | No | Yes | Disj | By Order | 不支持 |
| WebOQL | 半结构化 | 不支持 | 记录级别 | Yes | Yes | Yes | Disj | By Order | 不支持 |
| W4F | 结构化 | 不支持 | 记录级别 | Yes | Yes | Yes | SP | By Order | 支持 |
| XWRAP | 结构化 | 不支持 | 记录级别 | Yes | No | Yes | SP | By Order | 支持 |
| **监督学习** | RAPIER | 自由文本 | 支持 | 字段级别 | Yes | -- | -- | Disj | More constraints | 支持 |
| SRV | 自由文本 | 支持 | 字段级别 | Yes | -- | -- | Disj | More constraints | 支持 |
| WHISK | 自由文本 | 支持 | 记录级别 | Yes | Yes | No | Disj | By Order | 支持 |
| NoDoSE | 半结构化 | 支持 | 页面/记录 | Yes | Limited | Yes | No | By Order | 不支持 |
| DEByE | 半结构化 | 支持 | 记录级别 | Yes | Yes | Yes | Disj | More constraints | 不支持 |
| WIEN | 半结构化 | 支持 | 字段级别 | No | No | Limited | No | By Order | 不支持 |
| STALKER | 半结构化 | 支持 | 字段级别 | Yes | Yes | Yes | Both | More constraints | 不支持 |
| SoftMealy | 半结构化 | 支持 | 字段级别 | Yes | Limited | Multi Pass | Disj | By Order/Single Pass | 支持 |
| **半监督** | IEPAD | 模板化 | 有限 | 记录级别 | Yes | Limited | Limited | Both | By Order | 支持 |
| OLERA | 模板化 | 有限 | 记录级别 | Yes | Limited | Limited | Both | By Order | 支持 |
| **无监督** | DeLa | 模板化 | 有限 | 记录级别 | Yes | Limited | Yes | Both | By Order | 不支持 |
| RoadRunner | 模板化 | 有限 | 页面级别 | Yes | No | Yes | No | By Order | 不支持 |
| EXALG | 模板化 | 有限 | 页面级别 | Yes | No | Yes | Both | By Order | 不支持 |
| DEPTA | 模板化 | 不支持 | 记录级别 | Yes | No | Limited | Disj | By Order | 不支持 |

这篇综述中按照IE系统处理的输入数据，将这些输入数据分为：结构化、半结构化和自由文本三类，这对应其在综述中评价IE任务的难度为：简单、中等和困难。

*其中，手工配置和有监督的抽取系统主要是被设计来抽取多个不同网站的数据，而半监督和无监督抽取系统则主要被设计来对有固定模板的结构化数据进行抽取。后者的这些IE系统都必须针对使用固定模板的网站，而前者则主要包含一些由符号标记构成的抽取规则。通过融合了结构化数据中的一些特征，无监督IE系统体现了相当高的自动化程度，但是其对非结构化数据的扩展能力却受到了限制。*

以下虽非原话引用，但也是基本传达了综述中提到的基本意思：

**非HTML页面**：是指对HTML类型的输入数据的支持，主要指自由文本。我们看到目前自动化程度较低的系统大都可以同时支持非HTML类型的数据，而自动化程度较高的系统很难处理自由文本。

**任务等级**在这里被分为四个级别，它们由低至高为：字段级别、记录级别、页面级别和网站级别。这和之前所说了多个一个字段级别，这意味着一个抽取任务的目标仅是一个字段而已。

**抽取目标的数据模型**按照复杂度被分为4种，分别是：允许空值字段、允许多值字段、不确定字段顺序、嵌套复杂数据模型。因为允许空值和多值通常都能够同时支持，所以在表1中被合并为同一列。

由于数据**字段的格式**可以是变化的，这里将一个字段的格式分为：一个字段可能有多种格式和多个字段合用一种格式两类。这第一类字段，因为其格式不定，所以多数系统采用设定分割符的方式进行抽取，在表中使用Disj表示，而有的系统又忽略了格式问题，因为虽然格式不同，但是不同字段的顺序可能是一样的，所以如果依赖字段的顺序来解决这一问题的系统，在表中使用SP表示，如果两种都采用的则使用Both表示。第二类字段通常就可以按照顺序进行抽取了，但是因为有的数据可能会缺失，所以有的系统可以添加更多的限制条件来保证抽取结果的正确性，这里使用Add Constraints表示可以配置更多限制条件，By Order表示完全依赖字段的顺序。

**能否进一步抽取属性**是因为有的字段的值还需要进一步转换，例如日期字段，作者字段。更详细一些，某标签的文本可能是“作者：张三”，这时候，系统需要能够对这个属性做进一步抽取，仅抽取出张三即可。

表2：基于使用的技术进行的对比分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **系统名称** | **扫描方法** | **抽取规则** | **应用特征** | **学习算法** | **数据结构确认方式** |
| Minerva | Single | Regular exp. | HTML tags/Literal words | None | Manually |
| TSIMMIS | Single | Regular exp. | HTML tags/Literal words | None | Manually |
| WebOQL | Single | Regular exp. | Hypertree | None | Manually |
| W4F | Single | Regular exp. | DOM tree path addressing | None | Tag Level |
| XWRAP | Single | Context-Free | DOM tree | None | Tag Level |
| RAPIER | Multiple | Logic rules | Syntactic/Semantic | ILP(bottom-up) | Word Level |
| SRV | Multiple | Logic rules | Syntactic/Semantic | ILP(bottom-up) | Word Level |
| WHISK | Single | Regular exp. | Syntactic/Semantic | Set covering(top-down) | Word Level |
| NoDoSE | Single | Regular exp. | HTML tags/Literal words | Data Modeling | Word Level |
| DEByE | Multiple | Regular exp. | HTML tags/Literal words | Data Modeling | Word Level |
| WIEN | Single | Regular exp. | HTML tags/Literal words | Ad-hoc(bottom-up) | Word Level |
| STALKER | Multiple | Regular exp. | HTML tags/Literal words | Ad-hoc(bottom-up) | Word Level |
| SoftMealy | Both | Regular exp. | HTML tags/Literal words | Ad-hoc(bottom-up) | Word Level |
| IEPAD | Single | Regular exp. | HTML tags | Pattern Mining, String Alignment | Multi-Level |
| OLERA | Single | Regular exp. | HTML tags | String Alignment | Multi-Level |
| DeLa | Single | Regular exp. | HTML tags | Pattern Mining | Tag Level |
| RoadRunner | Single | Regular exp. | HTML tags | String Alignment | Tag Level |
| EXALG | Single | Regular exp. | HTML tags/Literal words | Equivalent Class and Role Differentiation by DOM tree path | Word Level |
| DEPTA | Single | Tag Tree | HTML tags tree/HTML tags | Pattern Mining, String comparison, Partial tree alignment | Tag Level |

**扫描次数**是指系统在抽取的过程中需要对输入文档进行扫描的次数。大多数系统都是一次扫描，抽取出多个相应的记录和字段。但为什么有的系统需要多次扫描呢？这是因为多次扫描更加灵活，更能适应复杂对象的抽取，在SRV和Papier中每个字段适用一个抽取规则，如果要从一篇文档中抽取多个数据，则必须要对同一篇文档扫描多次。当然，扫描一次要比扫描多次更加快速。

**抽取规则**有好几个类型。主要是基于正则表达式的，正则表达式非常适合于半结构化数据，尤其是有固定模板的结构化数据。有的系统还使用了一阶逻辑，这是因为在自由文本的抽取中，需要使用结合很多特征来作判断，于是一阶逻辑就成为了识别分割符、切割位置等信息的判别方法。另外Context-free和Tag-tree两种抽取规则还请自行参考相应的系统。

**使用特征**：由于早期的IE系统都被设计来抽取非结构化文档，所以大多系统都使用了HTML标签和字符串作为切割规则。在基于模板的网页中，使用DOM树的XPath现在已经成为能够代表信息的特别的特征。在基于自由文本的IE系统中，词性标注、Word-Net等语义计算方法，都被采用作为更多的特征。另外还有一些系统将通过训练获取的统计信息作为特征使用，用来生成抽取器。

**学习算法**：一些手工编写抽取器的系统是不需要抽取器的，另外的算法我们在这里也不做介绍。

**数据结构的确认方式**：一些手工编写抽取器的系统也是不需要为每个抽取信息确定字段的，一些系统按照文档的标签进行抽取，就是tag-level的抽取规则，也就是通过标签的位置来确定了数据的结构。另外还有一些系统通过字符串作为抽取器的分割符，这些是word-level的。

表3：对比IE系统的自动化程度

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **系统名称** | **对用户的要求** | **数据采集** | **输出格式** | **适用范围** | **限制** |
| Minerva | 需要编程 | 不支持 | XML | 高 | 无限制 |
| TSIMMIS | 需要编程 | 不支持 | Text | 高 | 无限制 |
| WebOQL | 需要编程 | 不支持 | Text | 高 | 无限制 |
| W4F | 需要编程 | 支持 | XML | 中 | 无限制 |
| XWRAP | 需要编程 | 支持 | XML | 中 | 无限制 |
| RAPIER | 需要标记 | 不支持 | Text | 中 | 无限制 |
| SRV | 需要标记 | 不支持 | Text | 中 | 无限制 |
| WHISK | 需要标记 | 不支持 | Text | 中 | 无限制 |
| NoDoSE | 需要标记 | 不支持 | XML, OEM | 中 | 无限制 |
| DEByE | 需要标记 | 支持 | XML, SQL DB | 中 | 无限制 |
| WIEN | 需要标记 | 不支持 | Text | 中 | 无限制 |
| STALKER | 需要标记 | 不支持 | Text | 中 | 无限制 |
| SoftMealy | 需要标记 | 支持 | XML, SQL DB | 中 | 无限制 |
| IEPAD | 后标记，模式选择 | 不支持 | Text | 低 | 需要训练数据 |
| OLERA | 部分标记 | 不支持 | XML | 低 | 无限制 |
| DeLa | 模式选择 | 支持 | Text | 低 | 需要训练数据 |
| RoadRunner | 模式选择 | 支持 | XML | 低 | 需要训练数据 |
| EXALG | 模式选择 | 不支持 | Text | 低 | 需要训练数据 |
| DEPTA | 模式选择 | 不支持 | XML | 低 | 需要训练数据 |

**对用户的要求**：一些手工编写抽取器的系统需要用户具备一定的编程能力，而其他的系统相应的需要用户参与语料标记、模式选择，或者两者同时需要。

**数据采集**是指有的系统只专注于信息抽取，而不提供数据采集的支持。**输出格式**一列表明了抽取程序是如何和其他系统集成的。**适用性**表明了程序的应用范围。通常是指程序可以适用各种类型的输入文档。

**限制条件**是指有的系统需要目标页面做为训练数据。

#### 小结

通过这篇综述，我们基本知道了一些2006年的情况。除了信息抽取、抽取器、抽取系统的基本概念之外：

* 原来研究信息抽取的大部分人都不想怎么采集数据；
* 原来一个抽取器可以自己写，也可以通过标记数据，训练出来一个具有启发性的程序，甚至通过输入目标网页，抽取系统可以自己训练抽取器；
* 原来抽取规则主要是正则表达式，其次是DOM Tree；
* 原来好多程序都是通过分割符来进行抽取信息的；
* 原来每个系统都有自己的适用范围，不是什么算法都能随便用的。

### 2. HTML文档的标题抽取及其对网页检索的应用

微软在Bing上的投入我们大家都是有所耳闻的，那么微软研究院下属的牛人们都在做些什么呢，他们怎么应用Web IE技术并有什么贡献呢？连续的两篇论文将为大家介绍Web IE技术上两类分析方法。

#### [*HTML文档的标题抽取及其对网页检索的应用*](http://research.microsoft.com/apps/pubs/default.aspx?id=69200)

***Yunhua Hu 1****,****Guomao Xin 2****,****Ruihua Song****,****Guoping Hu 3****,****Shuming Shi****,****Yunbo Cao****,****Hang Li******Microsoft Research Asia, 1 Xi'an Jiaotong University, 2 Peking University, 3 University of Science and Technology of China***

***摘要：****本文主要关注的内容是自动由HTML文档中抽取出正确的标题。HTML标题本应被正确的定义在title字段中，但实际上通常不是这样，所以只能由HTML的body中尝试抽取出标题。这个问题在之前的研究中并没有被很好的研究过，在这篇论文中，我们采用一种有监督的机器学期算法来解决这个问题。论文提出了一种针对HTML标题规范，并综合了字体、位置、大小等格式特征来进行抽取。我们的方法要优于直接使用最大字体作为标题这种方法（实验结果显示F1值提高了20.9%-32.6%）。在应用上，我们结合网页检索技术，并使用TREC的数据来进行评价。我们基于标题抽取提出了一种新的检索方法，实验结果显示：如果同时使用抽取出来的标题和原来的title字段作为标题，则检索结果也提高了23.1%-29%。*

***关键词：信息检索****，****HTML文档****，****元数据抽取***

作者们也许都是微软亚洲研究院的实习生，可能是团队合作，所以作者的数量很多。虽然作者和本文关系不大，写完本书之后，还是必须要请第一作者授权与我的。之所以先放下本书的内容却提到作者，是希望我们国内的某些搞学术的人，能尊重别人的学术成果。

从摘要上我们可以看到，标题抽取技术被应用到了提高搜索结果准确性上面了。看来他们对搜索市场是真的急了，不知道这篇文章上的方法，有没有应用到bing上，我个人觉得没有。但是其使用的方法，确是我们在前面提到的基于概率模型的机器学习算法。

文章描述的实验过程是这样的：首先人工从一些网页中标记出标题，然后将这些网页作为训练材料进行训练，得到一个分类模型，最终使用这个模型去对其他网页进行标题抽取。

这篇文章一共定义了245种特征，很多吧！那么他们是怎么选定这种方法的呢？

*Web IE目前由数据引导着发展。Breuel提出了基于DOM树解析网页的方法。以及Song等人提出了基于Block的方法，他们各人都开发出了各自的实现。*

*Web IE可以是与领域相关的，也可以是领域无关的。在某个特殊的领域，可以假设网页的结构是相似的，并且这些页面相似的结构可以被程序学习并用来抽取。Kosala等人观察到在同一个资讯网站的新闻都共享同一个模板，并且他们已经尝试了利用这一特点对新闻进行抽取。*

*通常的，信息抽取技术主要有两类，一类是基于规则的方法，一类是基于机器学习的方法，并且现在基于机器学习的方法已经被广泛采用了。*

*根据我们目前调研的情况看，并没有人专门针对标题抽取展开过研究，尤其在开放式环境中采用基于格式信息进行抽取的方法。我们的工作使用的方法类似于基于DOM tree信息抽取，不同于基于Block块的抽取方法，因为虽然定位重要的Block很有意义，但对标题抽取的影响有限，因为有的标题可能出现在不起眼的块里面。*

最终他们是要结合字体、大小、排列等格式信息，对网页的DOM树进行抽取。我们再简单的介绍一下他们的思路，也许对大家有所帮助。因为假定标题是出现在普通的标签中的，所以本文使用的方法只基于一些格式信息，来应用在标题抽取中。

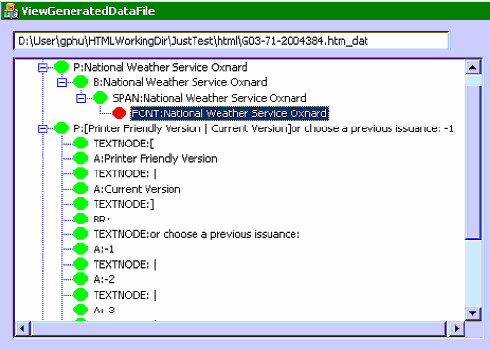
首先要考虑如何定义标题，这篇文章给出了标题是一篇文章中最明显的部分，并给出了一些相关规则

* 在数量上，一篇文档可以拥有两个标题，一个标题或者没有标题。
* 在位置上，标题只能出现在文档的顶部(处在文档的顶部1/4以内)，标题不能太短(宽度不小于文档的1/4)。
* 在外观上，标题的字体通常是最大或是第二大，并且被标记成注目的格式，例如字体、颜色、背景以及文本的长度
* 在邻居上，标题由连续不断的行组成，标题不可能是章节的标题 (第一章 xxx)，如果存在两个标题，两个标题通常被不同的字体、颜色的标签、链接或图片分隔。
* 在内容上，标题不可能是链接，标题不可能出现某些特定字符 (如：最新更新等)，标题可以直接跟在如下字符之后“标题：”。
* 其他，不考虑标题是图片的情况。

在本文中，采用了一种机器学习算法来抽取标题。此方法包括两个步骤：训练和抽取。这两个步骤都使用相同的数据预处理方法。整个数据预处理过程就是将输入文档转换成一个序列，序列中的每个包含有效字符的元素被称为一个单元unit。不包含文字的标签为空单元，在预处理之后将被过滤。例如：

* Unit1:[ text="Microsoft Corporation", align="center", bold="false", italic="false", isH1="false",... ]
* Unit2:[ text="Windows Operating System", align="center", large\_font\_size="false", italic="false", isH1="false",... ]
* Unit3:[...]
* ...
* Unit5:[ text="Overview", align="left", large\_font\_size="false", italic="false", isH1="false",... ]
* ...

一个unit序列其中包含的内容完全来自DOM树的内容，DOM树的内容如下：

  
图1：DOM树

*在学习的阶段，程序的输入是一个unit序列，每一个序列对应一篇文档。在训练过程中，已标注好标题和其他的unit，作为训练数据集被用来构建一个分类器模型，这个最终生成的模型将用来决定一个单元是否是标题。*

*在抽取阶段，程序的输入仍然是一个unit序列，每一个序列对应一篇文档。我们应用分类模型对每一个unit进行分类，并为每一个分类结果都标记一个类似隶属度的数值。*

*在之后的处理中，我们再使用一些启发式方法由分类结果中抽取出标题。程序的最后输出就是一篇文档的标题。更详细一些，就是将隶属度最高的且连续的单元抽取出来做为主标题，如果仍有标签被分类为标题，那就将隶属度次高的连续单元抽取出来做为第二标题，并且这些单元的隶属度都大于0。*

*我们将在一个通用的框架内描述模型，模型的输入是一个由多维向量构成的序列X1...Xn，并且每一个Xn对应相应的标记Yn，也即对应有一个标记的序列Y1...Yn。其中每一个多维向量Xn代表原始数据，例如一个unit单元。每一个标记代表是或者不是标题。*

*在本文中，我们采用了一种感知机模型作为分类器，Perceptron with Uneven Margin。这种感知机模型能够很好的处理正向训练数据和负向训练数据在数量上极不对等的情况。*

啊哈！我们看到了使用感知机作为分类器，这是一个经典的分类器模型，可以追述到上世纪50年代了，但是Uneven Margin我还不知道该怎么翻译，大概意思他也已经说明白了。因为一篇文章如果只标记“是标题”和“不是标题”两类的话，“不是标题”这一类的样本数据一定大大多于“是标题”的样本。可能Uneven Margin可以翻译为不对等数据集。

接着我们再介绍一下他是怎么选择特征的就好了，这篇文章主要是介绍训练一个分类器的步骤。我想再介绍一下特征的选择，目的就达到了。

*虽然我们主要是采用格式信息来进行抽取，但是标题的抽取也并非如此简单，因为标题可以出现在任意标签中，所以要尽可能选择有效的特征。我们最终考虑使用了如下特征：*

1. *Rich format information*
   * 1. *字体大小，并分为7个级别*
     2. *字体权重，是否加粗*
     3. *字体来别，新罗马、Arial等*
     4. *字体风格，普通或者斜体*
     5. *字体颜色*
     6. *背景色*
     7. *位置，居中、靠左、靠右等*
2. *Tag information*
   * 1. *H1，H2，H3，H4，H5，H6*
     2. *LI：是否是列表项*
     3. *DIR：是否是路径列表*
     4. *A：链接*
     5. *U：是否下划线*
     6. *BR：行分隔*
     7. *HR：水平*
     8. *IMG：是否图片*
     9. *Class name：'sectionheader'，'title'，'titling'，'header'*
3. *Position information*
   * 1. *标题处在Body的开头部分*
     2. *宽度不能太小*
4. *DOM tree information*
   * 1. *兄弟节点的数量*
     2. *相关的父节点、兄弟节点之中，是否包含字体大小的变化*
     3. *相关的向前叶子节点和向后的叶子节点，没有字体大小的变化*
5. *Linguistic information*
   * 1. *文本的长度*
     2. *字母的长度*
     3. *忽略日期、电话、传真、邮件、作者等字符*
     4. *位置标记，包含摘要、介绍、简介、标题等字眼*

*根据这些信息，我们创建了四类特征，它们分别是：位置特征、外表特征、邻近特征、内容特征。这些特征的总数有245种，下表针对每类特征都显示了一些例子。*

表1：样例特征

|  |  |
| --- | --- |
| **类型** | **特征简介** |
| 位置特征 | 单元位于页面上部的20％，40％或其他位置 |
| 位置特征 | 单位的宽度小于页面宽度的10%，20%，30％或者40％ |
| 位置特征 | 是DOM树上的第一个有效单元 |
| 外表特征 | 是H1,...H6，或是其他非H\*标签 |
| 外表特征 | 是第一或第二或第三个H\*标签 |
| 外表特征 | 是DOM树上的第一级或者第二级节点，并且是H\*标签 |
| 外表特征 | 是DOM树上唯一的H\*标签 |
| 外表特征 | 字体大小是最大或者第二大 |
| 外表特征 | 单元字体大小的百分比小于0.02等 |
| 外表特征 | 单元居中、靠右、靠左或其他 |
| 外表特征 | 是否粗体 |
| 外表特征 | 是否斜体 |
| 外表特征 | 是否下划线 |
| 外表特征 | 单元内部的颜色对比度 |
| 外表特征 | 单元内部颜色和背景色的对比度 |
| 外表特征 | 单元内部字体设置的统一度 |
| 外表特征 | 在列表元素内 |
| 外表特征 | 开始于新行 |
| 内容特征 | 所有字符大写或者首字符大写 |
| 内容特征 | 数字字符的数量小于8，8到64之间，等等 |
| 内容特征 | 内容开始字符为：subject，introduction，title，overview等等 |
| 邻近特征 | 前一个单元为HR或者BR |
| 邻近特征 | 字体大小大于父节点、前一个叶子节点、下一个叶子节点和兄弟节点。 |
| 邻近特征 | 前一个节点是靠左，但当前节点居中 |
| 邻近特征 | 和兄弟节点有一样的字体大小等等 |

*这些特征的数量如下表：*

表2：特征类型和数量

|  |  |
| --- | --- |
| **类型** | **特征数量** |
| 位置特征 | 8 |
| 外表特征 | 97 |
| 邻近特征 | 100 |
| 内容特征 | 15 |
| 其他特征 | 25 |

#### 小结

虽然这篇文档的抽取方法很简单，但是这篇文章在格式信息的归类上，做出了很细致的工作，告诉了我们从格式的角度如何看待每一个DOM树上的节点。

### 3. VIPS基于可视化信息的网页分块算法

VIPS在一段时间内，被我和我的同事们寄予厚望，希望能借助VIPS算法，自动发现新闻网站上的每个栏目，以及栏目下的每一篇文章。这里要描述一个题外话，为什么要自动发现网站栏目呢，是因为新闻搜索，如果能知道新闻属于何种栏目，那在用户检索的时候，就能够引导用户进行垂直检索，进行更深入的挖掘。不过后来发现这好像不是很现实，而且还有另外的办法解决这个问题。

#### VIPS：基于可视化信息的网页分块算法

***Deng Cai, Shipeng Yu, Ji-Rong Wen, Wei-Ying Ma***

***摘要：****本文基于网页视觉特征提出了一种新的针对网页内容结构的分析方法。采用这种方法，许多网络应用程序，诸如信息检索、信息抽取等都能从中受益。本文提出的检测网页内容结构的方法是一种自顶向下，与标签树无关的方法。通常用户是根据网页视觉上的设计，来理解网页的结构，而这种方法也正是如此。相比其他目前已经存在的技术，例如DOM树，这种方法是独立于HTML文档的，即使当HTML文档结构非常复杂的情况下也能很好的对网页的内容进行组织。本文还就此设计了若干实验来验证其优点。*

***关键词：网页切分，基于视觉信息的网页切分***

可能有很多人都喜欢这个算法，光看摘要就觉得很Cool。因为这模拟了人在看到某网页时，对网页内容的分析过程。同时还使用实验证明了对搜索结果有很大的改善，我们接着来看VIPS是如何实现的。

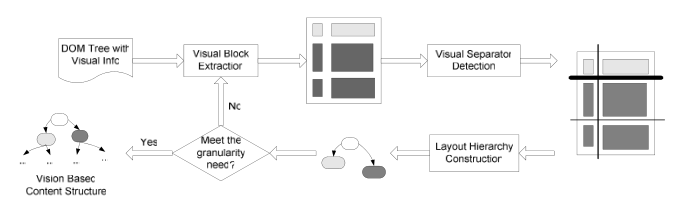
*当今因特网已经成为人们最大的信息源。绝大部分信息检索系统将网页作为系统的最小处理单元来对待，殊不知单个网页也有可能包含多个主题。一个网页通常包含与主题无关的内容有：导航、装饰、交互控件、声明等。另外一个网页也经常包含多个独立的主题。因此，对网页的内容结构进行分析，可以潜在的改善信息检索的性能。*

*许多Web应用都可以对网页的内容结构进行分析。例如在网络信息获取领域，为了克服浏览器和关键字查询的限制，许多研究人员就尝试在数据库技术的基础上，通过创建抽取器来组织网络数据[Adelberg et al. 1998][Ashish et al. 1997a][Ashish et al. 1997b][Hammer et al. 1997]。在生成抽取器的过程中，一个必要的过程就是将Web文档分拆成不同的数据块。之前的研究使用了一种ad hoc的方法来处理不同类型的网页。如果我们能够得到网页的语义内容结构的话，抽取器一定可以更容易的生成，使得信息可以更容易获取。另外，链接分析技术[Page et al. 1998][Kleinberg 1998]近来也被研究人员广泛关注。传统的链接分析方法对待每一篇链接都是公平的。链接分析算法的基本假设是：如果在两个页面A、B之间存在一个链接，那么意味着页面A和页面B存在着某种关联。但大多数情况下：一个由A指向B的链接仅仅代表了页面A中的某个部分和页面B中的部分。同时，如果大量的噪声链接出现，将会在HITS算法[Kleinberg 1998]中导致topic drift problem。近来主要研究集中在主题提取[Chakrabarti 2001][Chakrabarti et al. 2001]和主题采集[Chakrabarti et al. 2002]上。然而这些工作主要基于网页的DOM树来完成，而根据我们的实验，DOM树并不具备足够的能力来对网页进行精确的内容组织。*

首先发个牢骚：在提到链接分析技术的时候，没有李彦宏什么事，是个问题。剩下的简介部分就不翻译了，无非是说VIPS如何厉害。在这里我们得出了VIPS两种典型的应用场景。一是可以辅助对内容进行精确的抽取，二是在做超链分析的时候可以提高算法的性能。

*本文提出了VIPS算法用于抽取网页的内容组织结构。此种结构是具有层级关系的树形结构，其中每一个节点我们称为visual block。每一个节点都有一个值DoC，叫做相干度(Degree of Coherence)，DoC意味着在每一个visual block内部的内容节点间，在视觉特性上有多么的相关性。VIPS算法最大可能的应用了页面布局的特征：他首先根据DOM树抽取出合适的blocks，然后试图去找出这些visual blocks之间的切割线；这里切割线的定义是指在视觉上能够水平或垂直的切分visual blocks的矩形；最终，依据这些切割线，这些visual blocks又被组织成树形结构。在这个过程中，VIPS应用了自顶向下的分析方法，因此是非常高效的。*

因为是篇长论文，所以在对VIPS的讲解中，我将仅介绍VIPS是如何计算DoC以及切分block。以下是VIPS算法流程图：



图：VIPS算法流程图

1. 获得携带可视化信息DOM树；
2. 抽取可视化块；
3. 检测可视化块之间的切割线；
4. 根据切割线组织可视化块之间的父子关系；
5. 检测每一个块的DoC是否都满足设置的阈值，如果不满足则切割此块，并跳转到第二步继续执行。

*对每一个可视化块而言，DoC衡量着这个可视化块的相干性或说连贯性，或者说这个可视化块内部包含的节点在多大程度上，代表了相同的语义信息。DoC具有以下属性：*

* *Block它的DoC的值越大，那么这个bolck内的节点内容就越是一致；*
* *在由block组成的树中，子block的DoC一定不能小于父节点的DoC。*

*在VIPS中，DoC的值是1至10之间的整数，当然也可以使用其他类型和范围。*

*在可视化块的抽取过程中，旨在找出指定页面下有多少可视化块。通常来说，DOM树上的每一个节点都代表了一个可视化块。然而，一些节点所占页面比例非常的大，例如<TABLE>和<P>，它们常常被用于组织内容，所以并不一定被用来代表一个独立的可视化块。在这样的例子中，这些DOM节点应该被切分成若干个可视化块。更甚者，因为HTML语法的灵活性，许多Web页面并不完全遵照W3C的HTML标准，所以DOM节点之间的关系总是不能由标签本身正确反应出来。*

*对于每一个抽取出来的DOM节点，可以代表一个可视化块，这个可视化块的DoC值可以根据其视觉特征来计算。可视化块的抽取过程是一个迭代的计算过程，直到每一个可视化块都不能再分。*

*Algorithm DivideDomtree(pNode, nLevel)*

*{*

*IF(Dividable(pNode, nLevel) == True ) {*

*FOR EACH child OF pNode {*

*DivedeDomtree(child, nLevel);*

*}*

*} ELSE{*

*Put the sub-tree(pNode) into the pool as a block;*

*}*

*}*

*通常一个DOM节点是否可分是根据以下几个方面来决定的：*

* *DOM节点自身的属性。例如：节点的标签名，背景颜色、形状和大小等。*
* *其子节点的属性。除了子节点的标签名、背景颜色、大小外，还有其子节点有多少类型也要考虑。*

*根据WWW HTML说明文档4.01，我们将DOM节点分为两类：内联节点(inline node)和分段节点(line-break node)：*

* *内联节点：包含内联文本HTML标签的DOM节点，这些节点的影响着文本的样式，它们通常应用在一串字符的显示设置上。常见的有：<B>、<BIG>、<EM>、<FONT>、<I>、<STRONG>、<U>等；*
* *分段节点：其他非内联节点。*

*基于DOM节点和其子节点在浏览器上的样式，本文给出了如下几个定义：*

* *有效节点(Valid node)：在浏览器上被显示出来的节点，或者说此节点被渲染之后在屏幕上所占面积不为0。*
* *文本节点(Text node)： DOM节点中的文本节点，不包含HTML标签。*
* *虚拟文本节点(Virtual text node)：此定义为嵌套定义，*
  + *内联节点且仅包含文本节点作为其子节点*
  + *内联节点且仅包含文本节点和虚拟文本节点作为其子节点*

*根据刚才列出的算法DivideDomtree的伪码，用来判断一个节点是否可以再被分为其他可视化块的依据为：*

* + *标签：*
* *<HR>标签通常用来切分页面主题，如果一个DOM节点包含这样的子节点，则多半是可分的(Dividable)；*
* *如果一个内联节点的子节点中有分段节点，则此节点可以切分。*
  + *颜色：通常情况下如果一个节点和其子节点的背景颜色不一样，则应该被切分。但同时，在本次切分过程中，如果其子节点后代和子节点的背景颜色不一样，也不再切分了。*
  + *文本：如果一个DOM节点的后代多数是文本节点和虚拟文本节点，则不应该被切分。*
  + *大小：通常我们针对不同HTML标签预定义一个相对大小的阈值（通常是节点面积和整个页面面积的比值），如果小于阈值，则不应切分此节点。*

*基于这些依据，我们生成了一些启发式规则来判断是否一个节点应该被切分。如果一个节点不可以被切分，则此节点被抽取出来，并计算此节点的DoC。我们按照启发式规则的优先顺序在表1列出：*

表1：应用于可视化块抽取的启发式规则

|  |  |
| --- | --- |
| **规则** | **描述** |
| 规则1 | 如果DOM节点不是文本节点，并且没有有效的子节点，则此节点不能被切分，而且要被抛弃。 |
| 规则2 | 如果DOM节点仅有一个有效的子节点，并且此子节点不是文本节点，则应该切分此节点。 |
| 规则3 | 如果此节点是页面的第一级子节点，如果此节点没有兄弟，则切分此节点。 |
| 规则4 | 如果此节点的所有子节点都是文本节点和虚拟文本节点，不要切分此节点。如果其子节点的字体大小是相同的，则此节点的DoC应设置为10，否则为9. |
| 规则5 | 如果其子节点中有一个子节点为分段节点，则切分此节点 |
| 规则6 | 如果子节点中有一个节点是<HR>，则切分此节点 |
| 规则7 | 如果此节点和其子节点的背景颜色不同，切分此节点。同时不再切分其子节点，并依据其子节点的HTML标签等信息，设置其DoC为6至8。 |
| 规则8 | 如果此节点至少包含一个文本节点和虚拟文本节点，并且此节点的相对面积大小小于预先设定的阈值，此节点不可再分，并根据其HTML标签的名称设置其DoC为5至8。 |
| 规则9 | 如果此节点的子节点中，其相对面积大小无一超过相应的阈值，则不要切分此节点。并根据其HTML标签设定其DoC。 |
| 规则10 | 如果上一个兄弟节点没有被切分，则不切分此节点。 |
| 规则11 | 切分此节点 |
| 规则12 | 不切分此节点，并根据其HTML标签设定其DoC。 |

*针对不同的节点，我们定义了不同的规则链，表2展示了我们应用的一些规则链：*

表2：应用不同节点的规则链

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **规则** | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** | **R8** | **R9** | **R10** | **R11** | **R12** |
| 内联文本节点 | √ | √ | √ | √ | √ | √ |  | √ | √ |  | √ |  |
| <TABLE> | √ | √ | √ |  |  |  | √ |  | √ |  |  | √ |
| <TR> | √ | √ | √ |  |  |  | √ |  | √ |  |  | √ |
| <TD> | √ | √ | √ | √ |  |  |  | √ | √ | √ |  | √ |
| <P> | √ | √ | √ | √ | √ | √ |  | √ | √ |  | √ |  |
| 其他 | √ | √ | √ | √ |  | √ |  | √ | √ |  | √ |  |

我们可以由此可以简单概括出VIPS的方法：主要是根据DOM节点的一些属性，来决定这个DOM节点和其子节点，是否描述了相同的一类数据或者主题。VIPS在计算的时候，需要DOM节点的一些属性，和浏览器渲染之后的背景颜色和大小两个属性来一起判断此DOM节点和其子节点，在多大程序上描述或代表了相同的主题和信息。

他在后来的实验环节，通过对雅虎上14类网站的600多个页面进行了切分，证明了VIPS的准确率可以达到93%。并且将VIPS应用在信息检索领域，性能也有所提升，我个人并不完全认同其方法，可能在实际开发中过于复杂，难以应用到实际情况（也不知道Bing用了没有）。其基本流程如下：

1. 通过普通的方法检索到最相关的80篇文档。
2. 并不直接将最相关的文档返回给用户，而是对其使用VIPS算法进行分析，并找出最相关的20个可视化块。
3. 在这20个可视化块中进行分析，组织一个新的扩展查询
4. 查询系统并将此次得到的结果返回给用户

#### 小结

所以愚见是：VIPS还是很适合信息抽取系统的，可以方便机器判断那些地方可能有用户关心的数据，要用在信息检索领域，也不是不可以，就是对小公司来说，成本比较高，至少按照他的实验流程来看，是这样的。

### 4. 网页布局的数值化分析方法与自动信息抽取

这是我在2009年写的一篇文章，实际情况是未完成的，因为实验部分实在是没有时间很好的完善，而且新颖的地方并不多，基本的方法是遵循RoadRunner和Webstemmer演变来的。但是这基本代表了Websiteschema的基本思想，是Websiteschema的雏型。

#### 网页布局的数值化分析方法与自动信息抽取

***摘要：****综合目前对信息抽取技术的分析，本文借鉴在搜索引擎中，将文档集转换成向量空间的思路上，提出将网页布局数值化的方法，用以实现网页信息抽取的自动化。在本文构建的向量空间中，DOM树中的每一个XPath就是一个维度。通过构建向量空间，并为相同网站的不同页面进行聚类，通过聚类分析出相似的网页，然后采用目前一些已经较为成熟的方法，实现自动化抽取。本文介绍了网页布局的数值化方法，实现了对网易新闻的聚类分析，以及介绍了一个在此基础之上的信息抽取系统的简单实现。通过比较实验结果，证明了基于网页布局的向量空间，为实现无监督的机器学习提供了基本、有效的分析数据。另外作者认为其更重要的一个作用，是提示了大家不能忽略对目标网站页面类型的分析，其应该成为信息抽取的基本步骤，处于信息抽取的预处理环节中。*

***关键词：Web信息抽取****，****网页布局****，****聚类***

当时因为RoadRunner需要选择语料，所以Webstemmer就将选择语料这一步都省略了，通过聚类来自动发现新闻网页。但是webstemmer的计算方法速度相对较慢，且聚类结果一旦扩展到其他类型的网站，则难以获得很好的效果。于是萌生了改进Webstemmer聚类算法的想法。

*本文主要提出一种为页面布局数值化的方法，达到为网页进行聚类分析的目的，从而为进一步进行信息抽取提供有利的指导，本文主要讨论布局数值化的方法，并简单介绍一些在此基础上实现信息抽取的应用实现。*

那么为什么要对网页首先进行聚类呢，是根据什么进行聚类呢？

*目前网页信息抽取方法的情况，Hu[2]等人指出主要有基于DOM树和可视化Block两种研究方向，同时两种方法都互有一定的局限。基于DOM树的方法主要因为HTML的语法弹性很大，标签的语义信息很难明确[3]。同时基于Block的方法在作信息抽取的时候，很多有用的信息并非出现在明显的位置，以至于无法轻易的找到包含有用信息的Block[2]。在早些的论文[4]也指出，网页内容的抽取工作主要有几个共同的特点：*

1. *要么需要预先人工标定的学习样本*
2. *要么需要大量的先验知识*
3. *一个时刻只能处理一个页面*

*文献[2][4]中都指出：在内容丰富的网站中，大部分页面都可能通过后台数据库、脚本或程序发布，并提出了通过比较相似结构的文档，生成抽取器的方法。根据观察，即使是一个较大的新闻网站，其页面模板的数量也并不多，如果我们能够自动的归纳出网站上页面所属的类别，那么通过一些成熟的方法，可以更快捷的实现自动化抓取。*

*文献[6]的作者提出了一个非常实用的方法，并称为Webstemmer，并由文献[7]为Webstemmer总结并指出了Webstemmer适用的前提条件：*

1. *大多数网页都有共同的布局结构*
2. *页面中主要文本的位置固定*
3. *即使文本中的内容变化，页面的结构不变*
4. *页面中的广告、导航等信息常常不变*

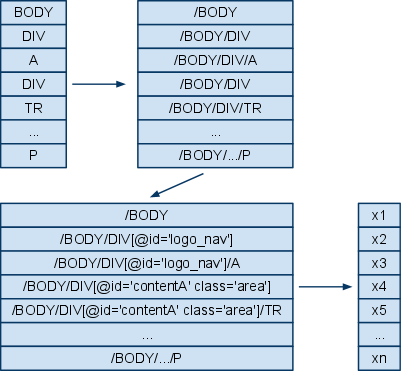
*Webstemmer设计的无监督聚类分析方法，很好的解决了为网站进行自动聚类分析的问题，为文献[4]中需要人工指定学习类别，实现了自动化的方法。*

*Webstemmer为实现无监督的网页抽取，取得了很大的成功。但实际使用中，Webstemmer仍然有一些限制和不足。例如在聚类和分类的上有一定的性能问题；相似性计算的算法太过简单，不适合复杂的Web页面；不便于管理和调整；对页面布局过于敏感；正是这些问题催生了对Webstemmer算法进行修改的愿望。*

我们提出按照网页的布局对同一网站内的页面进行聚类，这和Webstemmer的想法是一致的，但是我们改进了其聚类算法。

*本文拟提出将HTML页面分解为由绝对XPath代表的单元，构建以绝对XPath为轴的，描述HTML文档结构的向量空间。通过使用余弦度量的方法确定向量之间的相似度，最终实现对不同页面相同结构的分析。*

*考虑在同一个页面中，文章的结构可以考虑成如图1所示的标签的序列。标签的序列由左至右变为XPath的序列，再添加属性信息，最后变成一个n维的向量。*

*  
图1：HTML Tag、XPath、维度之间的转换*

*可以简单的认识到，一定程度上，简单标签的序列看起来就已经可以代表了HTML页面的结构。参考Webstemmer的设计，本文选用XPath作为这个序列的Items，同时假定XPath描述的位置和其节点在页面上的位置相关。因为经过观察，相同XPath的节点，其在页面上出现的位置也大致相当。*

*事实上，由于HTML语言本身的语法弹性太大，还需要在XPath中加上部分属性才能更好的使Xpath具有代表性。****这里提出一个假设：在同一个网站内，带有属性信息的绝对路径 (XPath)，和页面上拥有不同语义信息的标签，具有一一对应关系。****通过实践测试，使用align、font、class、id等几个分别代表位置、语义特征和外形的属性保存在XPath中即可。*

通过第一个假设，并利用XPath的唯一性，将一篇文档，转换成一个向量。但是并没有提及如何得到向量上每一个维度的值如何计算，接下来就介绍了怎么统计每一个维度的权重，而这个权重就代表了每一个维度的值。

*在将文章分解为由XPath组成的序列之后，我们就可以对两篇文章进行简单的比较了，如果两篇文章中XPath序列一致，那么这两篇文章必然是相同结构的。如果XPath序列相似，那么两篇文章也有可能是相同结构的。*

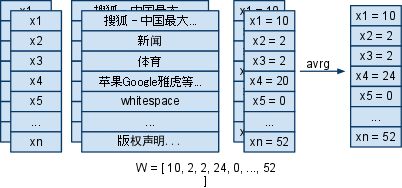
*似乎这是一个很好的办法，但是如果这样，同一网站的视频新闻和普通新闻就有可能会因为具有相同的广告，而被认为是相同结构的，在某些特定的要求下，这是不愿意接受的。*

*在Webstemmer中也提到了权重的概念，但其只是介绍了两篇文章相似度计算的方法。本文继续借鉴这种思想，但与Webstemmer不同。因为本文提出的方法首先以XPath为轴，构建了一个向量空间，从整个网站的大局来考虑。这样在程序设计上更加简单，对页面布局多样、负责的特点，也有较好的适应性。*

***本文提出的第二个假设：传递信息越多的标签，权重越大。****简单的说，标签中包含的平均文本数越多，权重越大。在这里，权重表示的是在向量空间中，处于某一维度的位置。*

*在实际操作中，首先采集一定数量的页面，成为样本页面，将每个页面解析为以XPath为坐标，以该XPath包含文本数量为值的向量。*

*在由这些向量构成的向量空间中，我们通过统计采集的样本，计算出每个XPath拥有的平均权重，形成权向量W。这里的平均权重，其实也是一个期望的概念。*

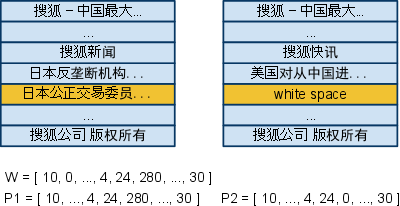
*  
图2：各维度权重的统计*

现在我们知道了一篇文章可以被根据其包含的XPath被转换成一个向量，这个向量的每一个维度就是一个XPath，每个维度内的值就是统计出来的平均权重。下面还举例说明了一下相似度计算：

*任何页面，只要拥有样本集里面的任意一条XPath，那么这个页面就在这个XPath代表的维度上，使用经过统计之后得到的权值Wi，做为该页面在这一维度的特征值。通过对样本页面使用这样的方法重新计算，得到若干向量，最后通过余弦度量的方法[3]，计算样本页面中每个页面的结构相似度了。*

*如果两个向量之间，同时包含一些最主要的维度，表明他们之间在几个权重较大的维度上拥有相同的值，其夹角变小，所以两个向量也就更相似了，体现了权重的作用。*

*在下图所示，快讯频道可能没有正文，这两个页面即使别的标签都一样，但因为正文的标签不同，所以两个页面之间差别仍会很大。*

*  
图2：各维度权重的统计*

*通过构建相似矩阵，可以实现聚类分析，分别得到几个类别的样本点。今后采集到新的页面，只要将其转换为向量空间中的一个点 (其中每个维度的值仍然使用统计之后的平均权重)，就可以计算出它属于那个类别了。*

其实只要构建了向量空间，就可以使用更通用的方法去进行聚类了，可以使用基于距离的计算方法，也可以使用基于相似度的计算方法。

#### 小结

这种方法可以帮助我们对网站内的网页进行分类，但不能解决全部的问题，并且这篇文章提出的一些假设，只是在大多数情况下适用。真实世界中不同网页之间的差异性，残酷的打破了希望以统一的方法对信息进行抽取的美梦。

**第二部分 工程篇**

敝人是学习自动化出身的，记得一段有关分散控制系统的描述，这是自动控制系统由集中管理模式向总线控制系统或网络控制系统发展的一些典型驱动。

“针对生产过程的改变，对可编程逻辑控制器进行相应的改变时，开发人员必须将所有需要的新功能重新集成在原来的控制器上。并且在开发一个控制系统的时候，每增加一个新的功能就意味着需要附加连线，伴随用户的需求，数据输入和输出的连线也随着功能的增加而增加。这使得系统的连线越来越复杂，也越加难以管理。而且当系统复杂到一定程度的时候，对系统的重新配置，变成了一件复杂而且代价高昂的工作。因为这几乎和重新设计系统是一样的。”

“现在，工业控制系统已经发展成：在一个开放式的网络中，原有的执行器、传感器、控制器都已经被抽象成了标准的数据结构。这些具有标准数据结构的软件模块就是功能块。”

“现代自动控制系统将能够适应不同的环境，能够在分布式网络中运行，基于模块的开放式系统能够整合友好的人机接口和高度智能化的软硬件设备。所以未来的控制系统将是在单元化、分布式控制系统发展的基础上，演变成由智能单元构成的社会化控制系统。”

这似乎正是我们很多程序员面临的困境，面对软件需求的不断增加，原来设计的接口变得越来越胖。本来想着写两个模块，多数时候，就变成一个了。

回顾软件开发的几个时期，按照出现时期排列可以分为：面向对象编程、组件化编程、SOA、云计算。一开始我们认为找到了描述世界的方法，就是对象。但是这仅仅是一种描述对象的方式。随着系统的发展，异构系统以及高类聚，低耦合的思想开始让大家寻找解决办法，于是DCOM、CORBA等基于组件编程的技术出现，最后发展到SOA。虚拟云的出现，更是促使大家以崭新的角度设计一个软件系统。单机版的应用程序似乎已经变成小孩子的玩意，而软件系统才是所谓精髓和核心了。

未来的软件系统必将由开放的、可扩展的模块组成，目标不再是实现某一特定需求，而是要为用户提供个性化的配置平台。

信息抽取系统也是如此，面对日新月异的网络信息，如何能够应对数据的快速变化，不依赖模块化的设计无法做到。这一部分依赖模块化设计，详细描述了信息抽取系统在调度、配置、运行等一系列过程中，应如何解决：简易、准确、快速、扩展、自适应等问题。真正将学术领域的研究成果，快速应用到实际的工程项目中。

### 5. 软件集成框架与功能块

首先提一个问题：“您认为10岁左右的小孩子可以造出一个机器人吗？”答案当然是肯定的，那么您是否感觉到一丝惊讶？为什么制作机器人会如此简单呢？我想这应该主要归功于 “软件集成和重用”领域持续的研究和实践的结果。

要探讨“软件重用”这一领域，作者显然不具备资格来探讨这个问题。但可以通过查找相关的资料来简单了解一下，比如从当下程序员最喜欢的百度开始。

“软件重用，是指在两次或多次不同的软件开发过程中重复使用相同或相似软件元素的过程。软件元素包括程序代码、测试用例、设计文档、设计过程、需要分析文档甚至领域知识。通常，可重用的元素也称作软构件，可重用的软构件越大，重用的粒度越大。”[[百度百科](http://baike.baidu.com/view/5081070.htm)]

软件重用是一个具备重大实践价值的研究领域，有很多人参与研究，本文如果讨论这个肯定会招人耻笑。但软件重用的概念在实际工程中并没有得到足够的重视，所以不才贸然的整理一下有关软件集成框架相关的资料。

#### 5.1.软件重用

其实无论什么框架，都是软件重用的实践。所以我整理了一篇软件重用相关的综述，主要是讨论软件重用的形式和特点。这篇综述就是《软件重用》（CHARLES W. KRUEGER）。

*简单来说，软件重用就是在构建新的软件系统的时候使用已经存在软件制品。这些被重用的制品的类型并不局限于源代码，而是包括设计、模块、接口、文档、变换等等。*

*集成：重用技术通常都包含一个集成框架。软件开发人员使用此框架将一系列精挑细选的软件制品组合成一个软件系统。一种模块连接语言就是典型的软件集成框架。*

*软件重用可以分为8种方法，它们分别是：*

*高级编程语言 设计和代码片段 源代码库 软件模式*

*应用生成器 更高级语言 软件变换 软件架构*

*对每一种方法，都从四个方面来讨论，这四个方面分别是：抽象、选择、说明和集成。*

*抽象：说明什么制品可以被重用，并且这些制品被如何抽象的。*

*选择：说明如何选择可重用的制品。*

*说明：制品是如何被推广和说明的。*

*集成：制品是如何被集成，并成为新的软件系统的。*

*高级编程语言：例如C、Ada、Lisp、Smalltalk和ML并没有被专门设计用于软件重用。*

*设计和编码片段：许多程序员都能很有效的重用一些设计模式和代码片段。这些从已有软件系统里面整理出来的代码也可以被用来做为新系统的一部分。*

*源代码库：构建大规模软件库作为软件重用的理念，指引了当下许多工业级别的软件库构建。模块连接语言提供了集成组件的框架。集成可重用的组件和整合代码一样重要。一旦采用了模块连接语言，在模块的连接中，命名必然会成为一个问题。*

*面向对象的组件重用：在引入了继承之后，就是一种典型的软件重用。组织好对象的关系和设计，能够帮助开发人员快速的选择需要重用的对象。*

*软件模式：暂时翻译为模式，是一些标准的算法、数据结构。这些通用的数据结构在开发人员接收基本教育的时候就开始接触，它们在软件重用的时候，使得开发人员能够快速找到相应的制品。*

*应用生成器：这类似一个简单的程序编译器，可以自动生成可执行的代码。针对某一特定领域，使用应用生成器能够带来比扩展代码更好的重用性。*

*更高级语言：是一种在普通高级语言上更高层次的一种抽象，可以有更加自然的语言描述，能够表达比普通高级语言更多的抽象。因此降低了开发人员的认知距离。*

*变形系统：软件开发人员使用高级语言描述软件系统的行为，变形系统在不改变系统行为的情况下，改进了系统的执行效率。*

*软件架构：可重用的软件架构是一种大颗粒度的软件框架和子系统，并被应用在软件架构的设计中。典型例子就是数据库。*

*总结：理想化的软件重用技术一定会使用以上所有的方法来缩短开发者的认知距离。因此，开发者在使用该技术的时候，会根据需求迅速的选择、配置和集成，生成最终的软件制品。*

有兴趣的读者可以自行阅读这篇论文。我们的重点是要介绍一种在工业控制领域中被定为标准的“软件集成框架”。这就是IEC 61499，一种基于事件的软件集成框架。通过简单的介绍之后，我们再来比较一下此种框架与软件重用8大方法之间，有什么联系。

#### 5.2.IEC 61499功能块

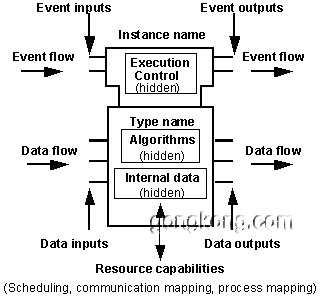
Websiteschema目前仅借鉴了IEC 61499中基本功能块的定义，所以我们仅介绍基本功能块模型。

*IEC 61499中，软件封装和重用的基本单元是功能块。功能块是一个软件功能单元，它由一个由其类型决定的专有数据结构和相关操作组成。IEC 61499基本功能块由事件输入和输出、数据输入和输出、执行控制表ECC、算法和内部数据组成，如图所示。执行控制表是一个事件驱动的状态机，它决定状态机的状态转换规律、状态机当前状态与输入事件的关系、执行算法与进入新状态时发出事件的关系。算法决定功能块的功能特性。特定事件发生时，其变化反映在相应的事件输入上，它驱动相应算法执行，算法读取输入数据，根据输入数据和内部数据产生内部数据和输出数据的新值，最后发出一个事件并把它输出到事件输出上。*

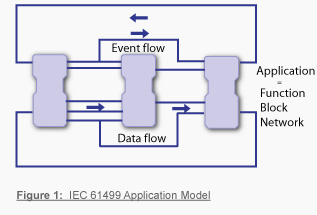
*功能块的最大特征在于其封装性，具有黑盒子特性。对于功能块外部来说，算法、执行控制表和内部数据都是不可见的，而且使用功能块时一般只需知道其外部接口。*

*来源于：百度百科*

基本功能块如图所示，功能块在接受到特定事件之后，将执行某特定算法并触发特定输出事件。或者根据特定的状态机模型（ECC），执行和触发相应算法和输出事件，输出数据在输出事件触发的同时被传递到指定的功能块。



下图描述了一个由功能块网络构成的应用，软件系统的功能由功能块决定。这是一种基于消息的计算模型，理论上是可以实现所有的应用需求。



#### 5.3.从软件重用的角度论功能块

功能块是非常简单直观的模型，现在我们以“软件重用”的8种方法，作为8个角度来试图探讨：“功能块原来是秉承了软件重用的思想”。

1、基于高级编程语言设计：功能块的实现，必定是基于高级语言的来实现的。这不仅仅是功能块的实现，目前出现的很多新的语言，都是如此。

Java，在C、C++语言基础之上实现，可以通过JNI与C调用native程序库。

Scala，在Java和C#语言之上实现，可以实现与Java和C#之间无缝结合。

Websiteschema中使用Java语言实现基本功能块。

2、程序片段：每一个功能块，都有可能来自其他开放源码软件的片段，这主要取决于开发人员的经验。

3、代码库：因为使用 Java，其可供选择的第三方API是非常丰富的，而且通过maven我们可以快捷的检索并获取第三方代码库。

4、软件模式：这主要取决于功能块的设计。例如我们要设计出来一些在数据结构中常用的功能块，这就是软件模式的一种重用。

5、应用生成器和更高级语言：应用生成器有点领域语言的意思，这大概可以在一个功能块里面实现。而所谓的更高级语言，其实功能块网络也可以看作是一种更高级的语言，它使用网络的方法描述算法，这种图形化的设计方法，极大的降低了开发人员的认知距离。

6、软件变换：功能块执行的优化其实并不像是“软件变换”的字面意思，作者觉得复合功能块到更像是软件变换。因为可以通过组合得到更多模块，使得开发人员事半功倍。

7、软件架构：通过设计特殊的功能块，我们可以将很多现在流行的软件模块结合至系统中，实现大规模软件系统的集成。

#### 小结

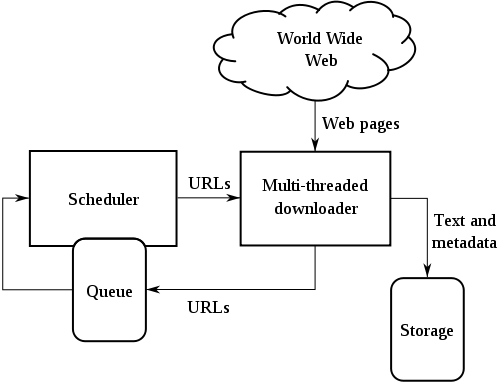
综合来看，功能块模型提供了一种模块连接语言、一种图形化编程语言。不仅仅改变的是一种编程方式，而且在软件系统的开发过程中，引入了一种新的角色。这在工业控制领域，称为组态人员。这些人员熟悉若干功能块，面对特定的任务，通过图形化设计工具，设计软件。而原来的软件开发人员则专注功能块的开发和测试，不去关注最终的业务需求。

但是功能块网络的运行更像是一些在后台运行的软件，目前来看要实现人机界面的设计，确实很难，需要投入大量的精力才能做出有新意又实用的功能。

### 6、信息抽取系统的架构

自动化信息处理系统应当包含对信息的收集、整理、存储、检索和传递等部分。信息抽取系统主要参与其中的搜集、整理、存储等部分，所以地位非常重要，而这就是信息抽取系统的基本业务需求。我们需要仔细的设计其架构，满足尽可能多的需求。

维基百科中关于crawler的架构我们可以仔细分析一下：



此图描述了一个通用的采集程序，这样一个程序通过调度器，可以多线程的采集网页，存储并产生新的任务。我们相信，调度、存储、多线程和队列是所有采集程序必不可少的。

从之前讨论过的软件重用来看，我们会重用这部分软件架构。

事实上，真正的采集系统没有这么简单。我们有时候需要精确的调度采集任务、有时候需要分布式的采集和存储数据，对数据的抽取也并不简单，不同类型的数据都有不同的要求。这些都是采集系统的基本要求，所以我们需要更多的抽象，来描述这个系统，例如第一部分提到的：采集和抽取。

#### 6.1、起始URL

任何采集程序都是由起始URL开始。这些URL在程序启动的时候被加入到URL队列中，于是我们需要这些起始URL除了是一个URL外，还有一个任务的概念，我们不应该单纯的使用配置文件来管理他们，至少Excel也是不恰当的。关系型数据库才是管理它们的最佳选择，当然您可能认为这可能有点大材小用，但是如果需要把您的程序做成基本可用，这是必须的，要不然上万个起始URL，怎么管理？

起始URL其实是一个任务，不仅仅是URL，仍需要记录任务在何时、以什么方式运行等信息。以及此URL本身代表的一些信息，例如网站域名、网站类型等等，而且这些数据都是变化的，因此一个结构化数据库是迫切需要的。

#### 6.2、数据存储

假设我们需要面对多达上亿的网页需要采集，随着时间增长，数据量可能更多，我们要考虑多强的服务器来存储呢？其实最有效的是考虑能够简单扩展的存储系统。并不是依靠专业存储，而是一种以低成本的方式进行扩展的分布式存储即可。

同时采集系统存储的多是文本数据，因此当下很多流行的分布式存储解决方案都可以考虑。这里面的代表有Big Table和Hbase等，一些可扩展的分布式数据库。

另外，如果你只需要采集，而使用搜索引擎、或者别的数据库系统来存储数据的话，也可以考虑关系型数据库暂存一段时间内的数据。

#### 6.3、分布式计算、任务调度与执行

任务是从调度器添加到URL队列中开始的，从这时起，程序就开始了一个互联网的遍历过程。此URL队列必然是一个庞大的队列，其数据结构并不复杂，但是其规模却是的可能惊人的（如果不能及时执行新增加的URL的话）。

对任务的分配上，我们当然需要能够将任务分配到足够大的集群中了，可以使用一种Master/Slave模式，当然对等模式也是可以的，但是其实现起来更加复杂。对等模式不仅仅需要维护整个系统中各个节点的状态，每个节点的之间也必须相互知道对方的情况。

以URL队列为调度器，让分布式子节点到队列中取得新任务，这样的实现方式相对简单得多，对系统的实现带来了很多方便。

通过消息队列来实现调度和分布式计算已经成为现代计算机系统的常用方法，我们可以找到很多消息中间件来作为URL队列，并基于此实现分布式应用。

#### 6.4、采集任务的多样性

面对不同的业务需求，我们发现，如果没针对一种应用，就需要编写新的采集程序，是非常不现实的。我们也清楚，现在的采集系统都可以针对不同的数据类型进行配置，这些配置多是以文本的方式实现的，同时正是应用生成器的应用。

如果以软件重用的观点来看，对配置人员而言：如何降低配置文件的认知距离是非常重要的。对开发人员而言：如何重用原有软件模块非常重要。

面对不断增长的网络数据类型，配置文件的变化是一定的，一旦出了新的版本，配置参数的变化是一定存在的，这时候认知距离，以及对原有软件模块的重用显得尤为重要。

图形化编程仅是一种解决方案，就像工业系统中的组态一样。配置人员和开发人员分离，使得任务得到很好的完成。另外一个我们之前举的例子是Lego机器人的软件系统。正是图形化配置，极大的降低了对参与人员与系统之间的认知距离。

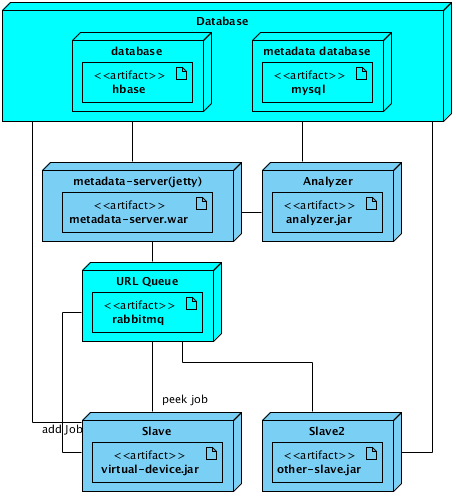
因为网络数据的多样性，对网页的分析必须和浏览器本身结合起来，也就是在配置的时候，很难离开浏览器，这是因为所见即所得的原则，我们可以直观的配置应用。

降低认知距离是产品设计的核心任务，我们通过增加组态人员这一角色，将传统的配置人员分为组态人员和参数配置人员，细化的角色，并都提供了相应的UI来降低他们的工作难度。

#### 6.5、Websiteschema的架构

下面的一张部署图展示，Websiteschema由关系型和非关系型数据库存储数据，由web应用实现对起始URL的调度，并通过消息服务器实现对采集任务的调度，最终由slave执行这些任务，而这些任务即是功能块网络。由一个基于firefox扩展的浏览器(analyzer.jar)实现配置。

颜色较亮的部分是重用的软件组件。其余部分是Websiteschema开发的程序。



#### 小结

Websiteschema扩展了简单的网络采集程序的架构，使其能够适应更大规模采集和更复杂业务需求。

该架构中大部分组件都是第三方软件产品，于是websiteschema实现的关键就在于设计自己的语言用来编写爬虫，这就是功能块网络。

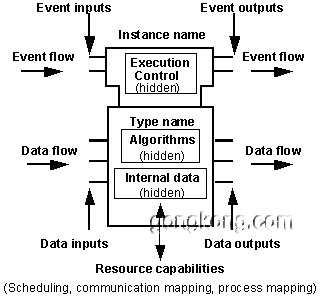
### 7、功能块的实现

几个常用功能块就足以构建基本的信息采集系统，体现了功能块网络的灵活性。其实针对特定的领域，通过简单的几个功能块，我们就能够实现很多功能，而且在特定场景下，功能块的描述性比配置文件更强，工作人员能够更快上手。

Websiteschema的项目lib-func-blk基于Java实现了基本功能块网络。通过注解，使得开发功能块和开发普通对象仅有一点点细微差别。

#### 7.1、基本功能块

基本功能块可以看作是一个组件，此类组件都有两种输入和两种输出。然后包含算法用来对相应的事件进行处理。其模型和IEC61499类似，如下图。



在package websiteschema.fb.annotation中，包含有若干注解，分别如下：

EI：表示一个组件有多少事件输入。关于EI的定义如下，EI仅关心有多少输入事件。但这里有一个trick，EI在实际定义的时候，必须定义每一个事件默认对应的算法，这样我们才能知道，到某个输入事件发生的时候，应该如何处理。

public @interface EI {

String[] name();

}

EO：表示一个组件包含多少事件输出。EO仅关心有多少输出事件。

public @interface EO {

String[] name();

}

DI：声明数据输入，是字段类型的注解。是用来定义接收数据的字段，目前要求声明了DI的字段都必须是public类型的。数据输入有名字、描述和相关事件。在功能块应用执行的时候，名字是必须的，但是描述和相关事件却不是必须的。描述主要用来做图形化编程的时候，对用户的一种提示。

public @interface DI {

public String name();

public String desc() default "";

public String[] relativeEvents() default {};

}

DO：声明数据输出，是字段类型的注解。用来定义输出数据的字段，目前要求声明的DO的字段都必须是public类型的。数据输出有名字、描述和相关事件。在功能块应用执行的时候，名字和相关事件是必须的。描述主要用来做图形化编程的时候，对用户的一种提示。

public @interface DO {

public String name();

public String[] relativeEvents();

public String desc() default "";

}

Algorithm：声明算法，是方法类型的注解。用来表示某个方法可以用来处理某个事件。目前要求声明Algorithm的方法都必须是public类型的。描述主要用来做图形化编程的时候，对用户的一种提示。

public @interface Algorithm {

public String name();

public String desc() default "";

}

EVT：声明某一参数为事件名，是参数类型的注解。因为可能有多个事件使用同一个方法来处理，那么这个方法上必然有一个参数，来表明事件的名称。

public @interface EVT {

}

Description：声明功能块的描述，在功能块应用运行时不是必须的。

public @interface Description {

String desc();

}

下面我们分别举例说明，在websiteschema中，一个基本功能块是如何编写的。

定时器功能块：这是一个描述定时器的功能块。首先定时器必须继承FunctionBlock，然后我们再看其相关注解。

Interval字段描述了定时器的间隔时间，默认是500毫秒。Times字段描述了定时器触发输出事件的次数，一定程度上可以当作计数器来看。Times字段的相关事件是EO，意味着当EO事件触发的时候，Times的值将会被同时传送到功能块网络上的其他功能块。

定时器功能块各有一个输入和输出功能块，它们分别是INIT和EO，当输入事件INIT被触发的时候，会执行Algorithm：INIT，这在EI的声明中确定的。

在算法INIT执行的时候，我们看到程序创建了一个Timer，timer被设置成每隔一段时间执行一次，每次执行的时候仅是触发EO事件。

@EO(name = {"EO"})

@EI(name = {"INIT:INIT"})

public class Timer extends FunctionBlock {

@DI(name = "interval")

public int interval = 500;

@DO(name = "times", relativeEvents = {"EO"})

public int times = 0;

java.util.Timer t;

java.util.TimerTask task;

public Timer() {

}

@Override

public void finalize() throws Throwable {

super.finalize();

t.cancel();

}

@Algorithm(name = "INIT")

public void init() {

t = new java.util.Timer(getName(), true);

task = new java.util.TimerTask() {

@Override

public void run() {

++times;

triggerEvent("EO");

}

};

t.scheduleAtFixedRate(task, 0, interval);

}

}

另外我们再看一个Merge功能块，它的功能是，一旦任何一个输入事件触发，都触发一次输出事件。这个功能块有四个输入事件，并且拥有四个输入数据，一旦对应的事件触发，则相对的数据将作为数据输出，发送到指定的功能块。

@EI(name = {"E1:M","E2:M","E3:M","E4:M"})

@EO(name = {"EO"})

public class Event4Data4Merger extends FunctionBlock{

@DI(name = "D1")

public String d1;

@DI(name = "D2")

public String d2;

@DI(name = "D3")

public String d3;

@DI(name = "D4")

public String d4;

@DO(name = "DO", relativeEvents = {"EO"})

public String dataout;

@Algorithm(name = "M")

public void DataMerger4(@EVT String ei){

if(ei.equals("E1")){

dataout = d1;

}

else if((ei.equals("E2"))){

dataout = d2;

}

else if(ei.equals("E3")){

dataout = d2;

}

else{

dataout = d4;

}

this.triggerEvent("EO");

}

}

#### 7.2、事件和数据连接的注意事项

功能块之间是如何实现连接，并互相传递数据的呢？因为功能块内部包含有一个RuntimeContext对象，此对象记录了功能块网络中所有的功能块实例。我们再通过Java反射，将数据读取和写入至对应的功能块。

每一个功能块对字段的定义主要包括类型和名称，当要传递数据的两个功能块之间，数据类型不同，这时候功能块应用就可能出错。例如功能块A要将一个字段date的数据，类型为Date的数据传给功能块B的字段time，但是time的类型是long，那么程序就会出错。

但是为什么说是可能出错呢？因为一旦数据类型不一致，程序会发现，并尝试转换，例如将String转换成Int。

基本对象之间比较容易转换，而其他对象您可能认为，通过JSON可以实现一定程序的自动化。但是websiteschema目前并未提供这样的机制，例如您可能希望程序可以自动将一个map转成一个对象，或者将一个JSON String转成一个对象。目前我们并不提供这样的隐式转换，目的是这样的转换可能会干扰到程序员调试过程中的判断，况且现在序列化技术、Java语言的动态性都有待提高，那么现在实现这样的功能也不会太好，大家编写功能块应用的时候注意一点好了。

private void setData(String name, String dataInput, Object data, RuntimeContext context) {

…

if (typo.equals(data.getClass())) {

field.set(obj, data);

} else if (String.class.equals(typo)) {

field.set(obj, data.toString());

} else {

field.set(obj, data);

}

return;

…

}

#### 7.3、应用-功能块网络

Websiteschema使用文本配置文件来描述功能块网络，之后我们都将称其为应用。Websiteschema稍微扩展了普通的配置文件，使其有了Zone的概念，其实并没有什么先进之处，仅仅是看起来更简单一些，同时支持以JSON格式书写对象。

[Timer]

FBType=websiteschema.fb.Timer

EO.EO={"Console":"PRINT","Threshold":"COMP"}

DO.TIMES={"Console":"STR","Threshold":"L"}

DI.INTERVAL=1000

其中每一个功能块是一个Zone，如果一个Zone配置的是功能块的话，必须包含参数FBType，功能块类型，其值是功能块的Class Name。之后就是配置功能块的EO、DO和DI，其写法有一点特别，就是由EO，加上点，再加上输出事件的名称，就代表了对某一输出事件了，DO和DI同理。

EO和DO的配置内容其实是一个Map[String, String]，对Map中的每一个键值对，表示一个功能块和对应的输入事件或输入数据。

DI的配置内容可以是简单数据类型，也可以是其他复杂数据类型的JSON String。在本例中DI.INTERVAL的值是一个简单类型Int。

现在没有使用Xml作为配置文件，是因为Xml配置文件阅读起来因为大量的其他字符的存在，没有配置文件简洁。今后如果考虑到使用Xml schema来检验配置文件的话，会考虑添加的。

一个应用的配置文件，必须的配置是：这表示应用启动的时候，触发的第一个事件，在本例中，这两个配置是指当应用启动的时候，自动触发功能块Start的COLD事件。

StartFB=Start

InitEvent=COLD

一个应用的实例是一个Application对象。其实现了IApplication接口，表明应用是可以单独运行的线程。同时应用拥有一个状态机，目前来说还很简单，仅有开始、结束和错误三种状态。

public interface IApplication extends Runnable, Callable<AppStatus> {

public long getTaskId();

public AppStatus getStatus();

}

如果要单独启动一个应用，我们可以使用以下几种方式。

Application app = new Application();

RuntimeContext context = app.getContext();

context.loadConfigure("fb/HelloWorld.app");

app.call();

或者以非阻塞的方式运行：

Application app = new Application();

RuntimeContext context = app.getContext();

context.loadConfigure("fb/HelloWorld.app");

Thread t = new Thread(app);

t.start();

又或者添加到线程池中运行：

Application app = new Application();

RuntimeContext context = app.getContext();

context.loadConfigure("fb/HelloWorld.app");

ApplcationService service = new ApplicationManager();

service.startup(app);

应用在开始执行功能块之前，会对应用进行一次检查，类似一个编译的过程，主要检查事件和数据连接是否都正确。

每一个应用都可以配置一个Spring容器，必须写在Zone：Bean下的SpringBean配置中，可以通过RuntimeContext的getSpringBean方法获取利用Spring加载的Java对象。

#### 7.4、Hello world

下面我们通过Hello World作为例子来看看如何配置一个应用。这个应用的配置比打印一个Hello World稍稍复杂了一点，但花点时间了解它应该是值得的。

首先我们配置了开始的事件是Start功能块的COLD。所以在一开始的时候，Start COLD事件会触发Timer功能块的INIT事件。

[Timer]

FBType=websiteschema.fb.Timer

EO.EO={"Console":"PRINT","Threshold":"COMP"}

DO.TIMES={"Threshold":"L"}

DI.INTERVAL=1000

接下来Timer功能块会初始化一个Timer并每隔1秒钟触发一次事件EO。这会触发功能块Console的PRINT事件，以及Threshold的COMP事件。并同时将Times传递到Threshold功能块的L字段。

当Console事件接收到PRINT事件的时候，将会在屏幕上打印“Hello World!”。但是这样这个程序就会一直运行下去了，不断的打印“Hello World!”了，不是吗？如果不添加Threshold功能块，则情况是会这样的。

[Console]

FBType=websiteschema.fb.STDOUT

DI.STR=Hello world!

Threshold功能块其实是一个比较功能块，L和R分别代表比较双方，如果L和R相等，则会触发EQ事件，现在EQ事件正连接到了Start功能块的STOP。通过改变Threshold功能块的配置DI.R，您还可以决定打印“Hello World!”的次数。

[Threshold]

FBType=websiteschema.fb.Compare

DI.R=1

EO.EQ={"Start":"STOP"}

整个应用的配置如下：

namespace=fb.test

StartFB=Start

InitEvent=COLD

[Bean]

SpringBeans=spring.xml

[Start]

FBType=websiteschema.fb.E\_RESTART

EO.COLD={"Timer":"INIT"}

[Timer]

FBType=websiteschema.fb.Timer

EO.EO={"Console":"PRINT","Threshold":"COMP"}

DO.TIMES={"Threshold":"L"}

DI.INTERVAL=1000

[Console]

FBType=websiteschema.fb.STDOUT

DI.STR=Hello world!

[Threshold]

FBType=websiteschema.fb.Compare

DI.R=1

EO.EQ={"Start":"STOP"}

#### 7.5、虚拟设备

我们通常不希望应用集中起来运行，或者说希望让应用能够作为任务，被调度在任意的机器上。这也符合传递计算不传递数据的思想，当然也没有那么好，因为目前websiteschema并没有让一个应用分布在多台机器上执行。现在的应用，以功能块为模型，更适合整合流程。

当我们要执行一个应用的时候，我们其实是希望将其调度至某个设备中执行。Websiteschema利用消息中间件RabbitMQ，将执行应用的调度信息发送至消息队列中，由消息的消费者来执行，通过这种方式设计的分布式框架。Virtual-Device是我们为这样一个设备起的名字，完全是一个应用的运行环境。

#### 7.6、没有OSGi就不是真正的模块化

目前功能块虽然支持OSGi，但是并不是完全支持，在使用Spring的时候，有一些问题截至目前仍未解决。但是不得不提到的是，作为Java模块化的准标准，确实是未来发展的趋势。

如果一旦将不同的功能块交给不同的团队开发的时候，问题就来了。因为当不同团队的开发者引入了相同的Jar包但是不同版本的时候，在集成运行的时候就会出现类加载冲突的问题。如果没有OSGi的封装，大规模的开发功能块势必会受到版本控制的问题。

因为受到Spring与OSGi兼容的问题，Websiteschema目前还是选择了Spring，当然未来还会投入更多的精力去解决两者之间的问题。

#### 小结

必须实现复合功能块，才能最大限度的体现软件重用。

OSGi与第三方软件的结合还需要更开放才可以。

**第三部分 技术篇**

本部分着重描述Websiteschema应用到的技术细节，来回答您的问题，例如：

* 如何识别网站中网页的类型，如何自动分析网页的结构，并抽取数据的？
* 网页采集过程中容易碰到的问题有哪些，可以怎么解决？
* 如何配置，以便抽取网页中特定的数据？
* 如何开发新功能和Websiteschema集成？
* 如何管理系统，识别目标数据的变化？

带您浏览websiteschema的源代码。

配置的问题（所见即所得），

合理的性能代价（选择合适的采集器），

管理和调度问题，

数据模型（Doc文档，Unit单元数据）

### 8、识别网页结构与配置

#### 8.1、网页结构数值化

基于XPath的假设

#### 8.2、聚类

#### 8.3、网页的类型

细览和概览

#### 8.4、标题、来源、日期的识别

### 9、采集程序基本功能块

几个常用功能块就足以构建基本的信息采集系统，体现了功能块网络的灵活性和健壮性。

#### 9.1、采集功能块

我们都知道，当设计一个函数去采集数据的时候，输入参数是一个URL，然后函数返回html字符串，或者DOM对象。事实上采集功能块的设计比这个复杂。我们首先设计了一个接口Crawler，此接口最重要的方法就是：

public WebPage crawlWebPage(String url);

然后我们编写一个功能块，对Crawler接口进行了封装，这样可以在应用执行的时候，可以配置不同的Crawler。

#### 9.2、内容和链接抽取功能块

一旦采集到内容，接下来应该就是抽取，抽取的对象通常包括几种对象：链接、内容和单元。这里的单元您可以理解成一行数据，一件商品信息，一个产品简介等。

内容抽取：Doc对象，Extractor接口。

链接抽取：Link对象，XPath和MustHave。

单元抽取：Map对象，基于XPath的单元抽取。

#### 9.3、数据存储功能块

将采集到的内容存储至数据库是采集的最终主题。

#### 9.4、URL队列功能块

遍历网站是采集程序的基本功能，对子链接的采集可以通过向URL队列中添加新的URL来实现。

#### 9.5、一个简单的爬虫

#### 小结

仅有几个功能块是完全不够的，仅仅说这个系统有了很好的框架，但是系统需要继续丰满。

### 10、虚拟设备与URL队列

#### 10.1、Rabbitmq消息服务器

#### 10.2、Message对象

#### 10.3、任务的状态

### 11、管理和调度

站点、起始URL、任务和调度

#### 11.1、站点的属性

#### 11.2、起始URL的属性

#### 11.3、Quartz和调度

#### 11.4、任务的配置

#### 11.5、报告和调度优化

### 12、配置一个网站

#### 12.1、前提假设

假设采集深度为1

#### 12.2、配置站点

#### 12.3、配置任务

#### 12.4、配置调度

### 附A：Extjs+Spring+DWR+Mybatis与元数据服务器

### 附B：对HBase的封装

### 附C：XPath与DOM