大数据环境下信息抽取模板自动聚类与发现

计 92 丘骏鹏 2009011282

指导老师：朱小燕　郝宇

提纲

研究内容介绍

系统框架与模块实现

实验和分析

总结和展望

提纲

背景

- “大数据”时代来临，我们获得的数据越来越多，研究工作也受到了新的挑战
- 为了便于计算机处理这些数据，需要从非结构化和半结构化数据中提取出我们关心的内容，存储成结构化的数据
- 大部分网页是通过查询后台数据库，然后选择合适模板进行渲染方式生成的。即：

背景

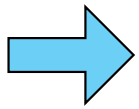
- “大数据”时代来临，我们获得的数据越来越多，研究工作也受到了新的挑战
- 为了便于计算机处理这些数据，需要从非结构化和半结构化数据中提取出我们关心的内容，存储成结构化的数据
- 大部分网页是通过查询后台数据库，然后选择合适模板进行渲染方式生成的。即：

```
<html>
  <body>
    <h1>{{ news.title }}</h1>
    <p>{{ news.content }}</p>
    <div>
      {% for comment in news.comments %}
        <li>{{ comment }}</li>
      {% endfor %}
    </div>
  </body>
</html>
```

背景

- “大数据”时代来临，我们获得的数据越来越多，研究工作也受到了新的挑战
- 为了便于计算机处理这些数据，需要从非结构化和半结构化数据中提取出我们关心的内容，存储成结构化的数据
- 大部分网页是通过查询后台数据库，然后选择合适模板进行渲染方式生成的。即：

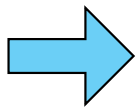
```
<html>
  <body>
    <h1>{{ news.title }}</h1>
    <p>{{ news.content }}</p>
    <div>
      {% for comment in news.comments %}
        <li>{{ comment }}</li>
      {% endfor %}
    </div>
  </body>
</html>
```



背景

- “大数据”时代来临，我们获得的数据越来越多，研究工作也受到了新的挑战
- 为了便于计算机处理这些数据，需要从非结构化和半结构化数据中提取出我们关心的内容，存储成结构化的数据
- 大部分网页是通过查询后台数据库，然后选择合适模板进行渲染方式生成的。即：

```
<html>
  <body>
    <h1>{{ news.title }}</h1>
    <p>{{ news.content }}</p>
    <div>
      {% for comment in news.comments %}
        <li>{{ comment }}</li>
      {% endfor %}
    </div>
  </body>
</html>
```



工作内容

1. 已经获得大量新闻、博客等网页数据，这些网页可能由不同的模板生成
2. 我们将网页按不同的模板分开，然后从每个集合中提取出所有可能的模板
3. 根据提取出的模板，抽取新的网页中的有用的信息，存储成 XML 格式。

工作内容

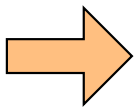
1. 已经获得大量新闻、博客等网页数据，这些网页可能由不同的模板生成
2. 我们将网页按不同的模板分开，然后从每个集合中提取出所有可能的模板
3. 根据提取出的模板，抽取新的网页中的有用的信息，存储成 XML 格式。

工作内容

1. 已经获得大量新闻、博客等网页数据，这些网页可能由不同的模板生成
2. 我们将网页按不同的模板分开，然后从每个集合中提取出所有可能的模板
3. 根据提取出的模板，抽取新的网页中的有用的信息，存储成 XML 格式。

工作内容

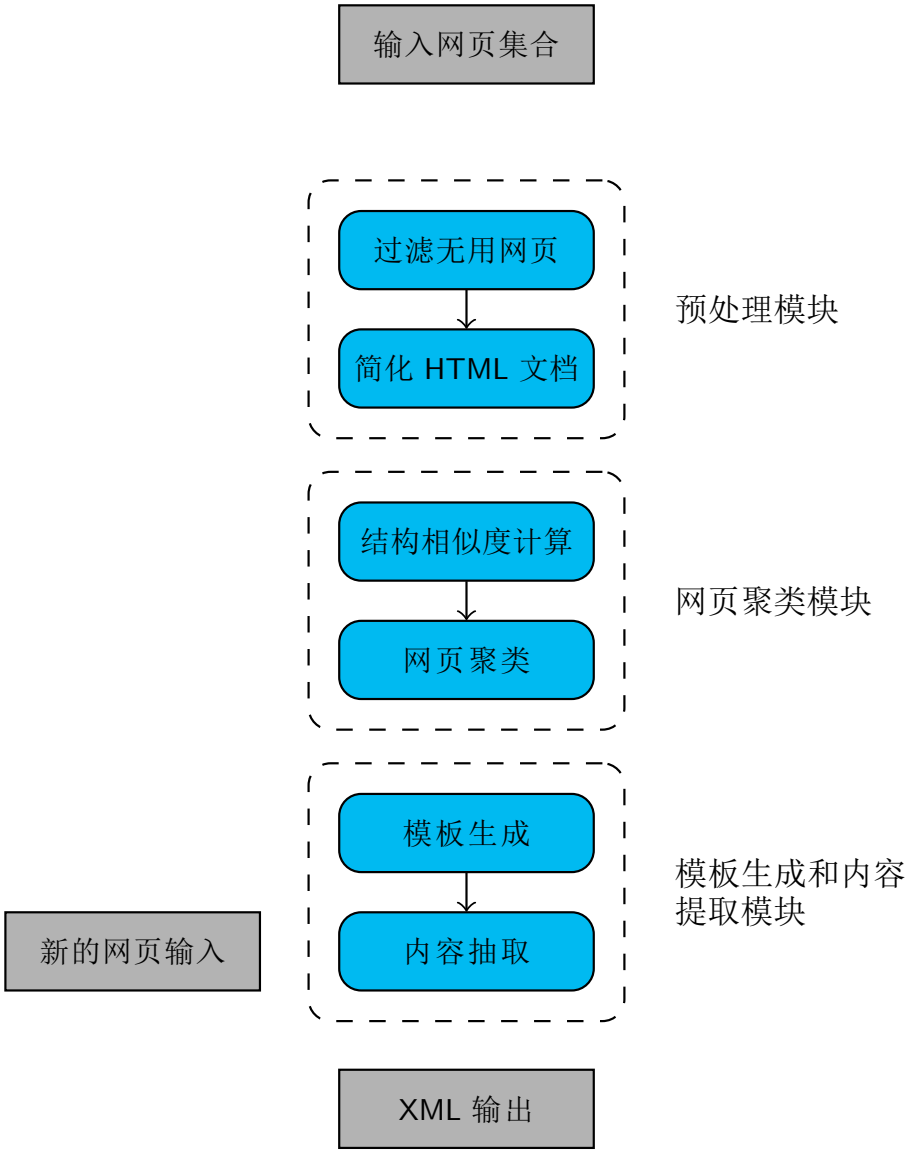
1. 已经获得大量新闻、博客等网页数据，这些网页可能由不同的模板生成
2. 我们将网页按不同的模板分开，然后从每个集合中提取出所有可能的模板
3. 根据提取出的模板，抽取新的网页中的有用的信息，存储成 XML 格式。



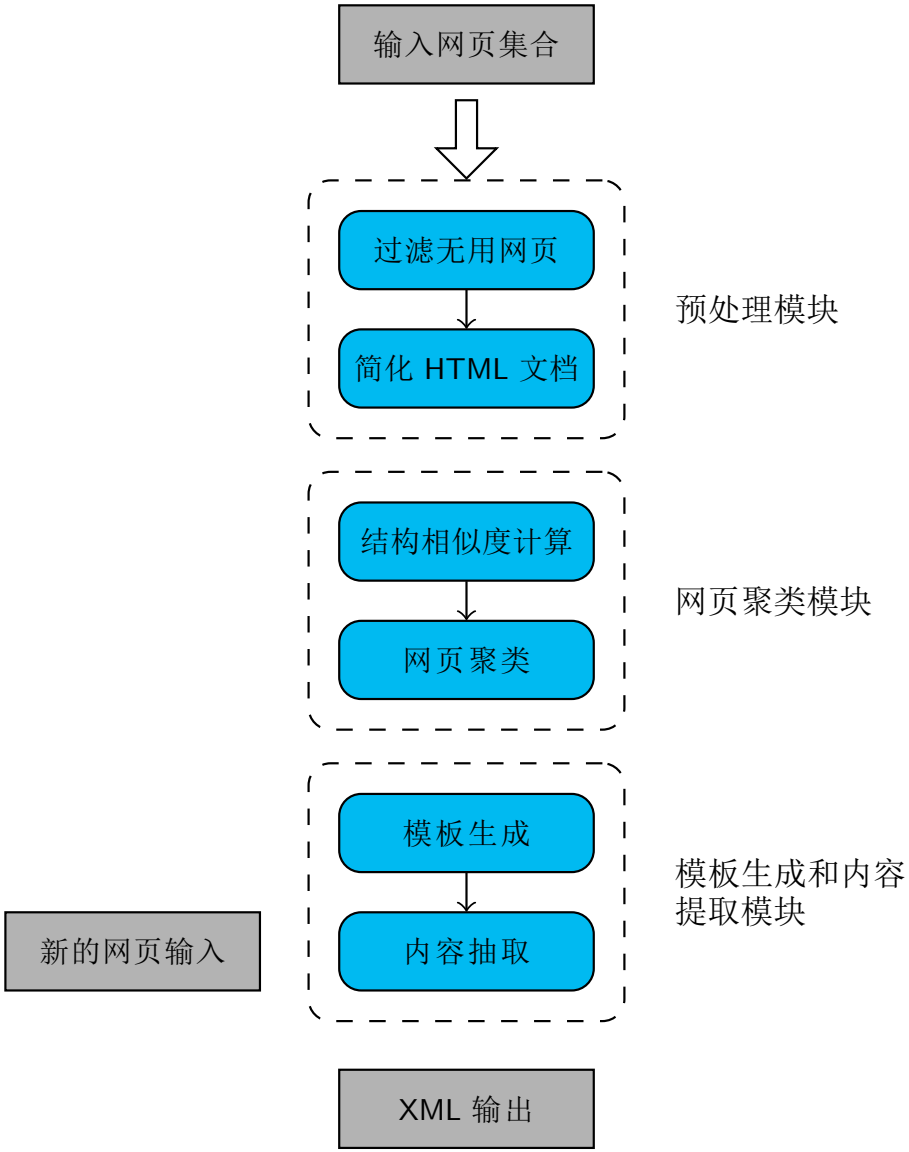
```
<documents>
  <document>
    <title>
    WWDC:2013 苹果仍未解创新和增长之困
    </title>
    <content>
    今日，备受瞩目的苹果全球开发者...
    </content>
  </document>
</documents>
```

提纲

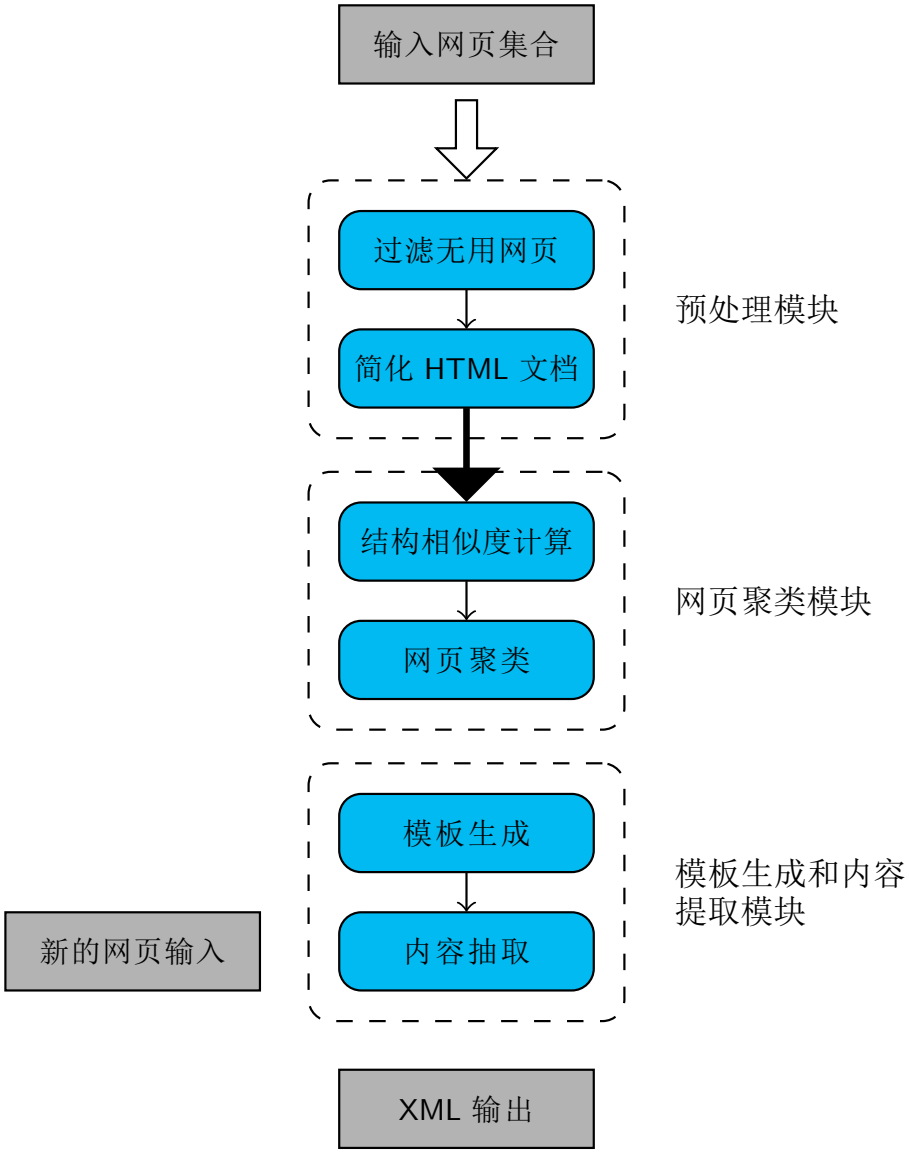
整体框架


输入网页集合
过滤无用网页
简化 HTML 文档
预处理模块
结构相似度计算
网页聚类
网页聚类模块
模板生成
内容抽取
模板生成和内容
提取模块
新的网页输入
XML 输出

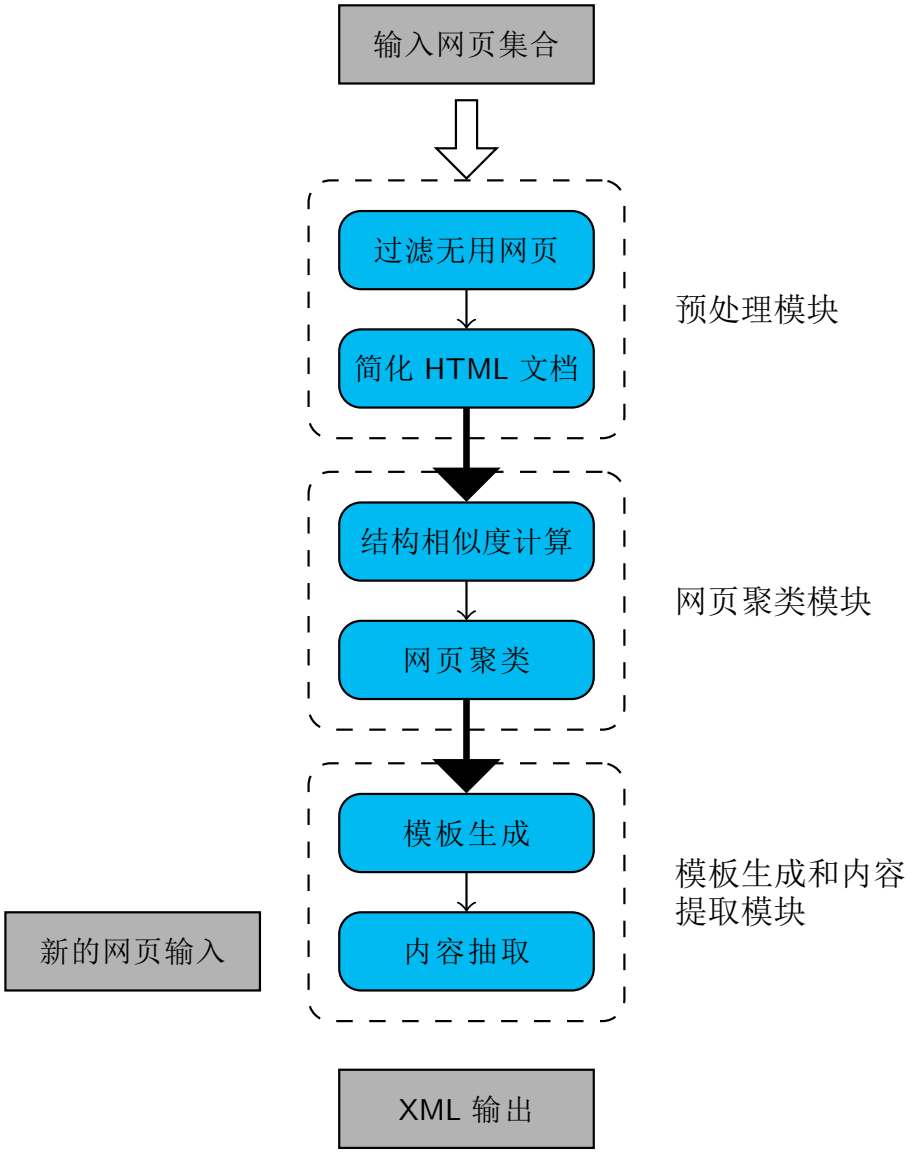
整体框架


输入网页集合
过滤无用网页
简化 HTML 文档
预处理模块
结构相似度计算
网页聚类
网页聚类模块
模板生成
内容抽取
模板生成和内容
提取模块
新的网页输入
XML 输出

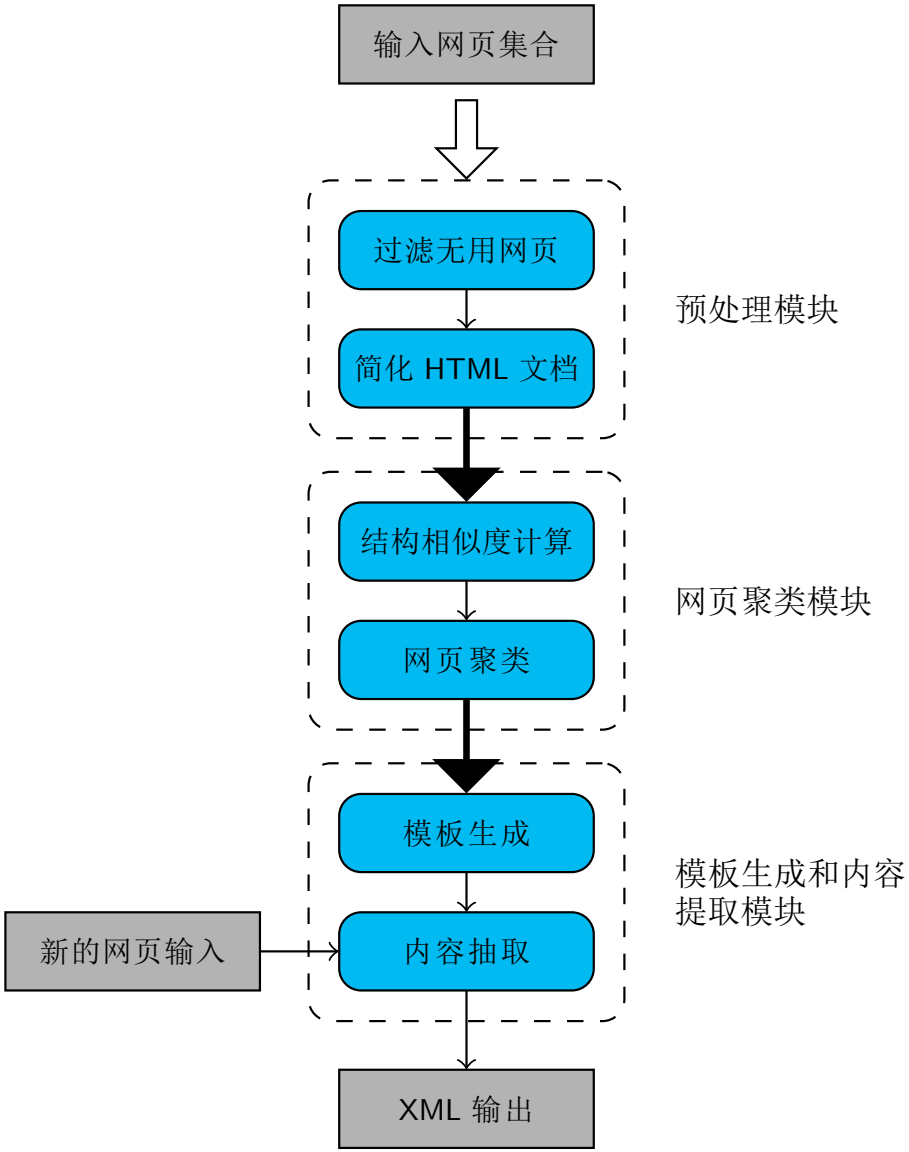
整体框架


输入网页集合
过滤无用网页
简化 HTML 文档
预处理模块
结构相似度计算
网页聚类
网页聚类模块
模板生成
内容抽取
模板生成和内容
提取模块
新的网页输入
XML 输出

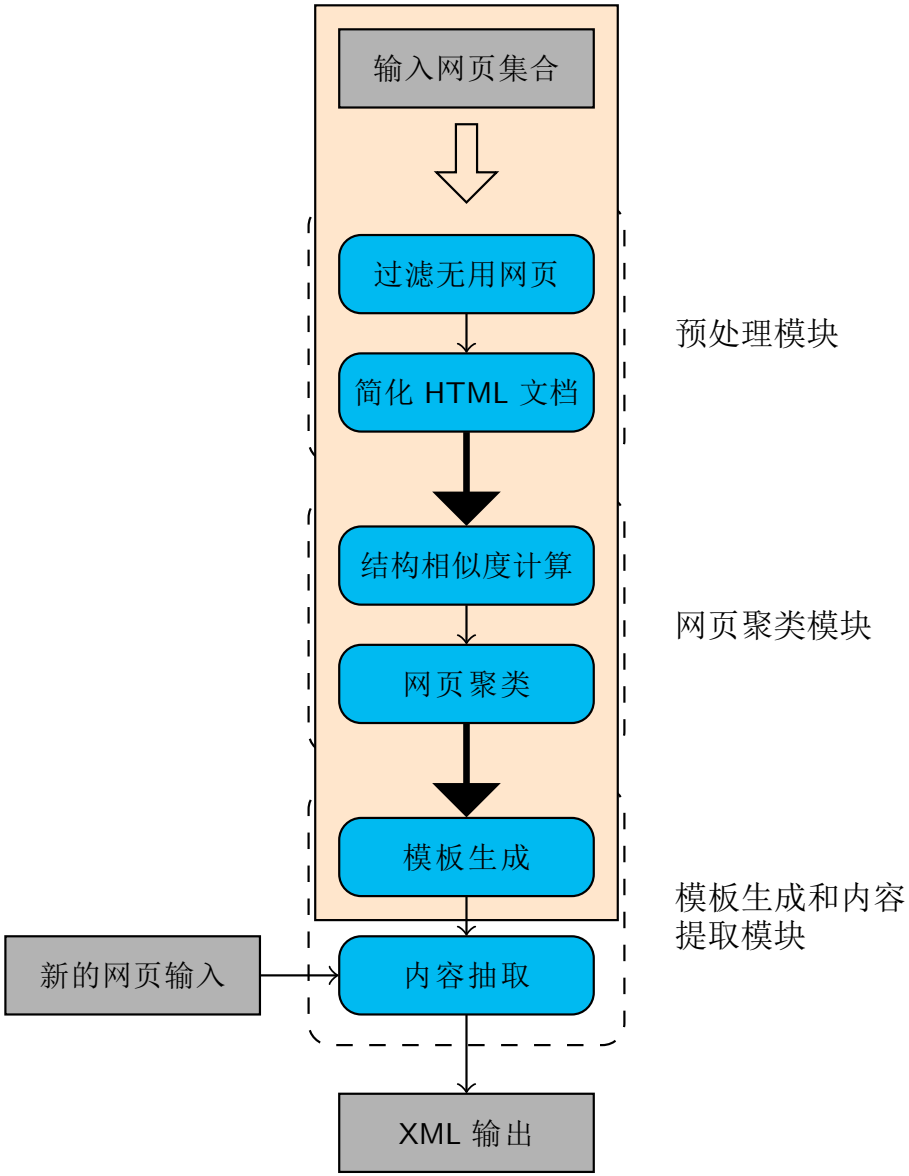
整体框架


输入网页集合
过滤无用网页
简化 HTML 文档
预处理模块
结构相似度计算
网页聚类
网页聚类模块
模板生成
内容抽取
模板生成和内容
提取模块
新的网页输入
XML 输出

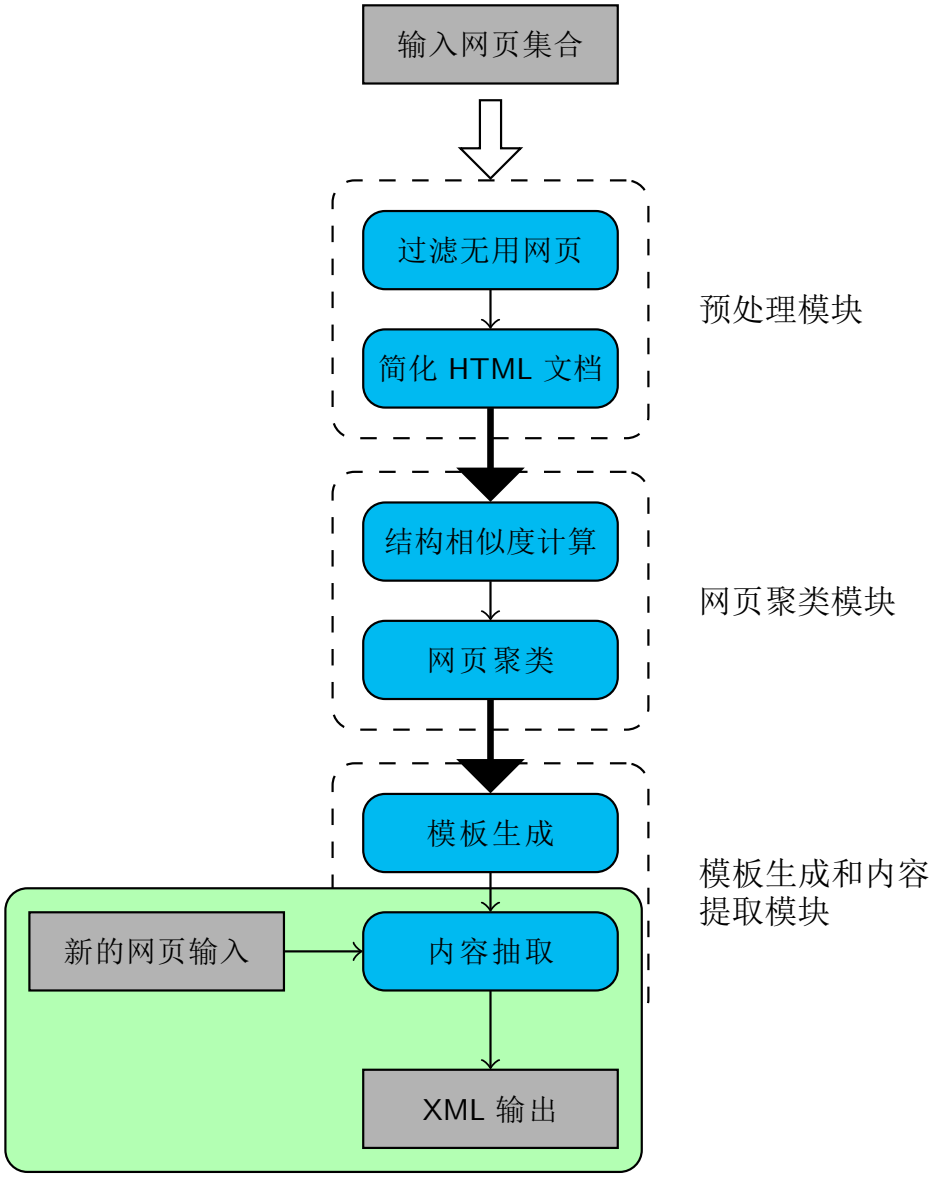
整体框架


输入网页集合
过滤无用网页
简化 HTML 文档
预处理模块
结构相似度计算
网页聚类
网页聚类模块
模板生成
新的网页输入
内容抽取
模板生成和内容
提取模块
XML 输出

整体框架


输入网页集合
过滤无用网页
简化 HTML 文档
结构相似度计算
网页聚类
模板生成
新的网页输入
内容抽取
XML 输出
预处理模块
网页聚类模块
模板生成和内容
提取模块

整体框架


输入网页集合
过滤无用网页
简化 HTML 文档
预处理模块
结构相似度计算
网页聚类
网页聚类模块
模板生成
新的网页输入
内容抽取
模板生成和内容
提取模块
XML 输出

预处理模块 -过滤无用网页

- 404 等错误页大小一般比较小，可以通过设置文件大小阈值过滤
- 目录页的 URL 有一定的模式，可以用为每类网页分别建立正则表达式的方法过滤目录页。比如对于新浪博客来说，详细页的 URL 都以.html 结尾，而目录页则没有。以下两个 URL:

  `http://blog.sina.com.cn/u/1439351555`
  `http://blog.sina.com.cn/s/blog_55cac30301016yb1.html`

  第一个 URL 对应的是某个博主的文章目录，第二个 URL 则是该博主的某篇文章。

预处理模块 -过滤无用网页

- 为了完备性考虑，可能存在某些网站, 目录页的 URL 和详细页的 URL 在模式没有太大的区别, 或者人工设置规则的办法太麻烦了
- 目录页的 HTML 文档的主要部分是由列表环境 <ul> 或 <ol> 中的列表项 <li> 及其包含的超链接标签 <a> 组成。我们提取出网页中的所有 <li> 标签中的 <a> 标签, 计算这些 <a> 标签的文本内容占网页所有文本内容的比重, 超过一定的阈值则判定为目录页。

```
<ul>
  <li><a>红会社监委今日讨论是否重启调查郭美美事件</a></li>
  <li><a>北京今日14时起堵车 中度以上拥堵将持续7小时</a></li>
  <li><a>湖北黄冈拒卖地反投8亿建免费公园 市长称不亏</a></li>
  <li><a>湖南株洲市档案局工会主席杀害局长后跳楼</a></li>
  <li><a>陕西神木集资大王被查 结婚时曾空运20辆林肯</a></li>
  <li><a>内地学生扎堆报京沪港高校 在校平均恋爱0.78次</a></li>
</ul>
```

Figure: 百度新闻部分简化的 HTML 代码

预处理模块 -简化 HTML 文档

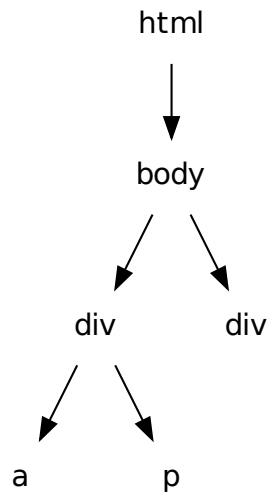
去除无用标签

1. 与网页模块无关。去除这些标签时, 可以将标签本身及标签的内容一同去掉。比如 <script>、<link>、<style> 等。
2. 通常是深层次的 HTML 文档的节点, 用于控制格式, 在模板中意义不大。去除这些标签时只取出标签本身, 保留标签所包含的内容。比如
、<p>、、 等。
3. 我们在实验中不考虑非文本标签, 因此将 <img>、<audio> 等直接去除。
4. 最后是去除一些在模板中变化很大的太复杂的标签, 典型的是表格相关的一些标签, 包括 <table>、<th>、<tr>、<td> 等。

预处理模块 -简化 HTML 文档

对解析好的 DOM Tree 进行简化

- 树形结构上做相关的操作较为复杂, 我们首先将树形结构转化为更便于处理的序列形式
- 采用先序遍历的方式, 将 DOM Tree 转化成一个标签序列, 在遍历的同时在每个节点处保存了该节点的深度信息
- 每个标签采用 标签名 + 深度 的表示方法，下图可以表示为：`<html0><body1><div2><a3><p3><div2>`



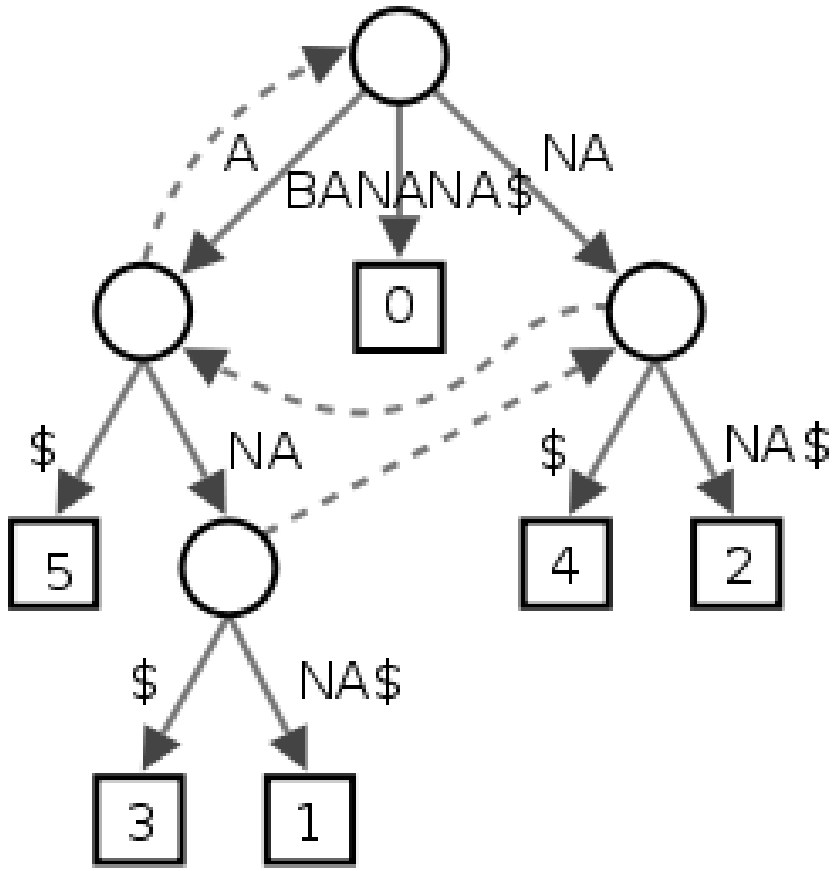
预处理模块 -简化 HTML 文档

检测重复记录 -后缀树简介

- 网页模板中有不定个数的重复模式，通常由模板语言的 for 语句生成，我们需要将这些记录合并
- 后缀树是一种高效的数据结构，可以快速完成重复字串的查找。
- 后缀树定义：由序列所有的后缀组成的 Trie 树。后缀树的每一条边都代表着一个序列, 从根节点到后缀树叶子节点的每条路径都对应着原序列的一个后缀。
- 后缀树普通的构造算法复杂度很高，系统实现中采用了 Ukkonen 在 1995 年提出了一个 $O(n)$ 时间复杂度的在线构造算法。

后缀树示例

BANANA 对应的的后缀树



后缀树查找重复序列算法

原始算法

- 任意一条从根节点到内部节点的路径组成的序列都是原序列中重复出现的字串，且重复的次数是以该内部节点为根的子树的叶子节点个数。
- 不能处理：不同的重复子串之间可包含或者相交的关系；同一个重复子串在原序列上有交集

新的算法要求

- 重复序列不能横跨两个子树
- 重复序列必须有公共的父亲
- 重复序列必须尽可能地长

后缀树查找重复序列算法

算法 1 从根节点出发，找出所有的重复子序列

输入: 已经构建好的后缀树，根为 *root*

输出: 该后缀树中所有的重复子序列

```
1: // 从根节点出发，寻找所有的重复子序列
2: for edge ← root.edges if edge.endNode.isNotLeaf do
3:     // 取后缀树根节点的每条边的第一个元素作为每个子树的根节点
4:     subTreeRoot := edge.firstElement
5:     // 查找以该节点为根的所有重复子序列
6:     findAllRepetitions(root, nil, subTreeRoot)
7: end for
```

后缀树查找重复序列算法

算法 2 简化的 findAllRepetitions 实现

```
输入: 一个内部节点 node，当前已经找到的重复序列 prefix，要找的子树的根节点 subTreeRoot
输出: 所有经过该内部节点的符合要求的重复序列
    function findAllRepetitions(node, prefix, subTreeRoot)
        // 定义一个空集合
        results := Collection.empty
        // 对于该内部节点的每一条不连接叶子节点的边
        for edge ← node.edges if edge.endNode.isNotLeaf do
            // 依次取出该条边上属于该根节点子树上的点
            seq := edge.takeWhile(element inSubTreeOf subTreeRoot)
            if seq.length == edge.length then
                // 遍历完了该条边上所有元素，
                // 则取该条边连接的下一个内部节点进行递归查找
                findAllRepetitions(edge.endNode, prefix + seq, subTreeRoot)
            else
                // 否则，将当前得到的序列加入到结果集合中
                addToResults(prefix + seq, results)
            end if
        end for
        return results
    end function
```

合并重复记录

- 所有序列按实际的父节点分组，然后对每个分组进行合并，保证重复串有公共的父亲。
- 但是仍然存在序列相交的情况，如

  `<div3><div4><div3><div4><a5><img5><div4><a5><img5>`

  `<div3><div4>` 和 `<div4><a5><img5>` 都是符合算法要求的重复串，但互相之间有交集 `<div4>` 。我们采取的策略是只取更深的重复序列。这里只取 `<div4><a5><img5>`

网页聚类模块 -计算结构相似度

- 序列 S_1, S_2 ，x_i 和 y_j 分别表示 S_1 的第 i 个元素和 S_2 的第 j 个元素，$f(depth)$ 是根据深度加权的函数，我们认为深度越深的节点，成为模板的概率就越小。

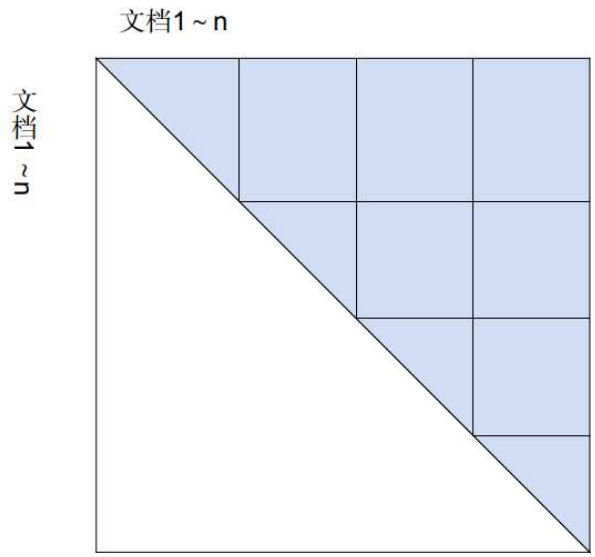
$$t(i)(j) = \begin{cases} 0 & i = 0,\ j = 0 \\ t(i-1)(j-1) + f(x_i.depth) & i,\ j > 0, x_i = y_j \\ \max(t(i)(j-1), t(i-1)(j)) & i, j > 0,\ x_i \neq y_j \end{cases} \quad (1)$$

- 结构相似度计算公式

$$Sim(D_1, D_2) = \frac{|elcs(S_1, S_2)|}{\max(\sum_{n \in S_1} f(n.depth), \sum_{n \in S_2} f(n.depth))}$$

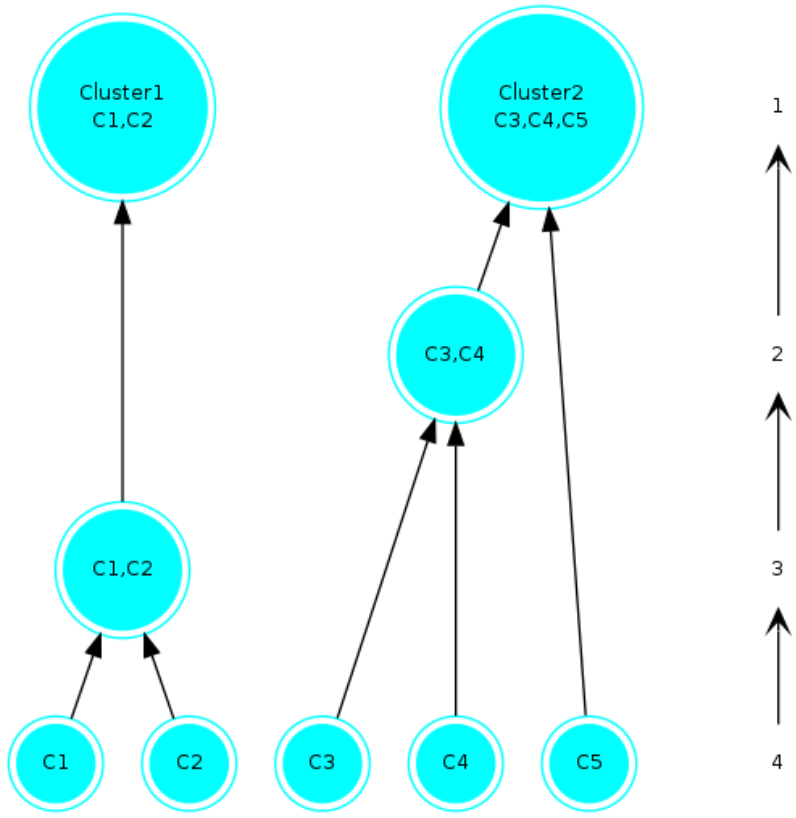
计算时间优化

- 文档数量多，计算量很大，需要一定的优化。
- 采用 Actor 库实现多线程计算，加快计算速度。将任务分割后交给每个 Actor 进行计算，Actor 的调度的算法采用 RoundRobin。
- 任务分割示意图



聚类算法

1. 初始时，让每个文档实例都单独为一类
2. 迭代时，每次选择距离最近的两个类合并
3. 直到任意两个类的距离都大于阈值时程序退出
4. 算法的结束条件由阈值决定，无需实现设定类的个数。



模板形式化定义

基本节点

- 两种组成方式
  1. 单个不重复的 HTML 标签，即 $< tag >$
  2. 由一个或多个 HTML 标签组成的序列，这些序列可以出现一次或多次，即 $(\sum_{i=1}^{N} < tag_i >)+$ ，其中 $N \geq 1$
- 第二种形式通过合并重复记录得到，对应的模板语言为：

```
{% for comment in news.comments %}
  <li>{{ comment }}</li>
{% endfor %}
```

- 还有一种模板生成形式：

```
{% if news.has_subtitle %}
  <h1>news.title</h1>
  <h2>news.sub_title</h2>
{% else %}
  <h1>news.title</h1>
{% endif %}
```

模板形式化定义

必选和可选节点

- 必选节点 EN 对应着由基本节点组成的一个序列，若模板节点用 tn_i 表示，则必选节点可以表示为

$$EN = tn_1 tn_2 ... tn_n$$

- 可选节点 ON 同时对应多个序列，每个序列由不同的基本节点组成，同时每个序列还对应着一个出现概率 p ，即：

$$ON = tn_{11} tn_{12} ... tn_{1n_1}, p_1 | tn_{21} tn_{22} ... tn_{2n_2}, p_2 | ... | tn_{k1} tn_{k2} ... tn_{kn_k}, p_k$$

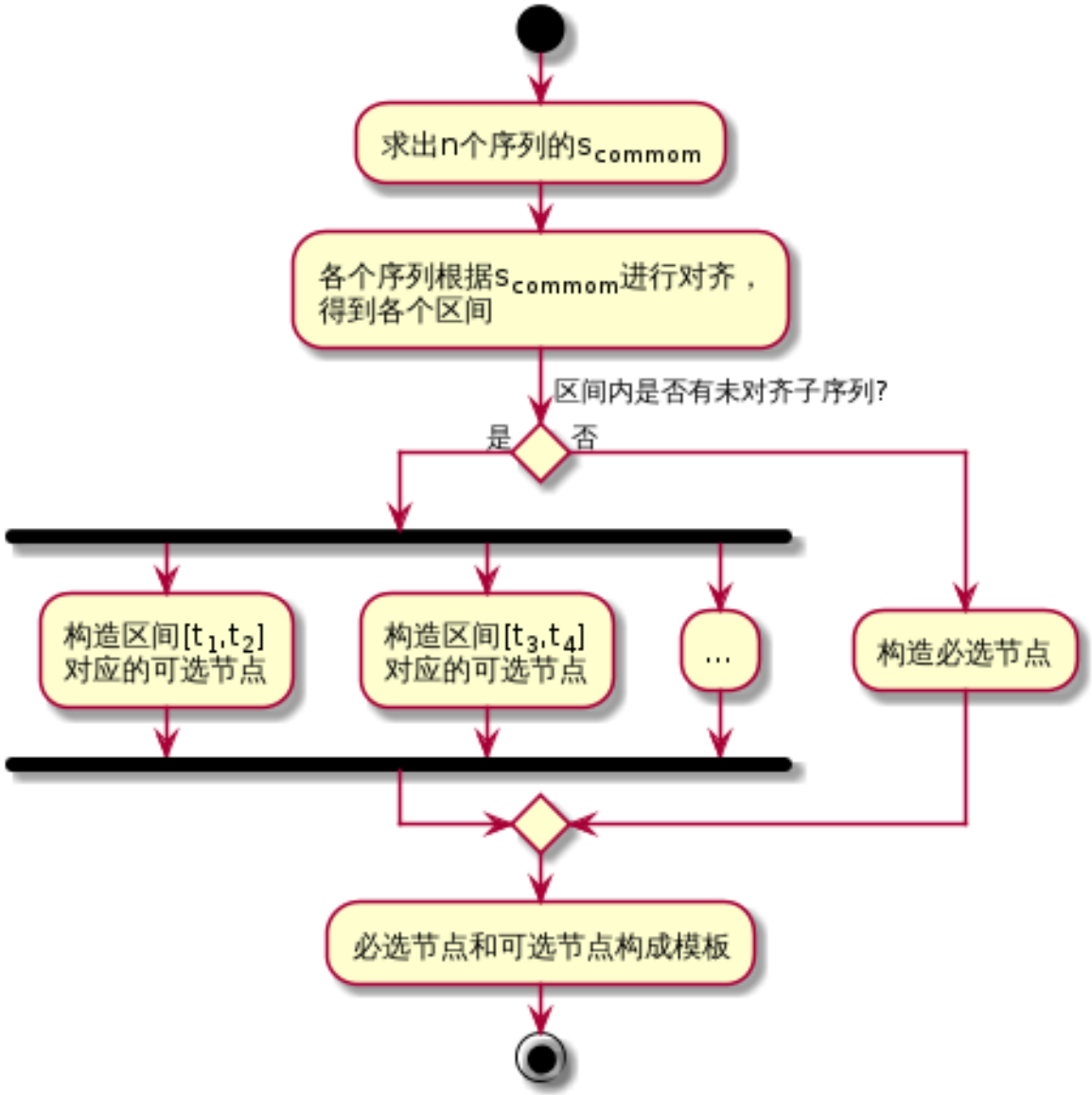
- 定义模板 Tp 必须由这两种节点交替组成：

$$Tp = [ON_0] EN_1 ON_1 EN_2 ON_2 ..... EN_n [ON_n]$$

模板生成算法

- 找出所有组成必选节点的基本节点：对于一个聚类的全部序列，从聚类中心点开始，依次计算一次最长公共子序列，得到 n 个序列的公共子序列 S_{common}
- 每个序列和 S_{common} 对齐，得到一些未对齐的区间，每个未对齐的区间计算一个可选节点
- 已对齐的基本节点组成必选节点
- 必选节点和可选节点交替出现，组成最终的模板

模板生成流程图


求出n个序列的s_{commom}
各个序列根据s_{commom}进行对齐，得到各个区间
区间内是否有未对齐子序列?
是
否
构造区间$[t_1,t_2]$对应的可选节点
构造区间$[t_3,t_4]$对应的可选节点
...
构造必选节点
必选节点和可选节点构成模板

简单的例子

- 设有 3 个序列，分别为：

$$
\begin{aligned}
s_1 &= aorzbcdlxe \\
s_2 &= athubeatcdlxe \\
s_3 &= athubeatcdpkue
\end{aligned}
$$

- 根据最长公共子串算法，得到 s_{common} 为 $abcde$
- 每个序列同 s_{common} 进行对齐，得到

$$
\begin{array}{rcccccccc}
s_1 & : & \mathbf{a} & orz & \mathbf{b} & & \mathbf{cd} & lx & \mathbf{e} \\
s_2 & : & \mathbf{a} & thu & \mathbf{b} & eat & \mathbf{cd} & lx & \mathbf{e} \\
s_3 & : & \mathbf{a} & thu & \mathbf{b} & eat & \mathbf{cd} & pku & \mathbf{e} \\
s_{common} & : & \mathbf{a} & & \mathbf{b} & & \mathbf{cd} & & \mathbf{e}
\end{array}
$$

简单的例子

- 根据上述对齐结果，生成以下可选节点：

$$
\begin{aligned}
ON_{a,b} &= thu, 2/3 \mid orz, 1/3 \\
ON_{b,c} &= eat, 2/3 \\
ON_{d,e} &= lx, 2/3 \mid pku, 1/3
\end{aligned}
$$

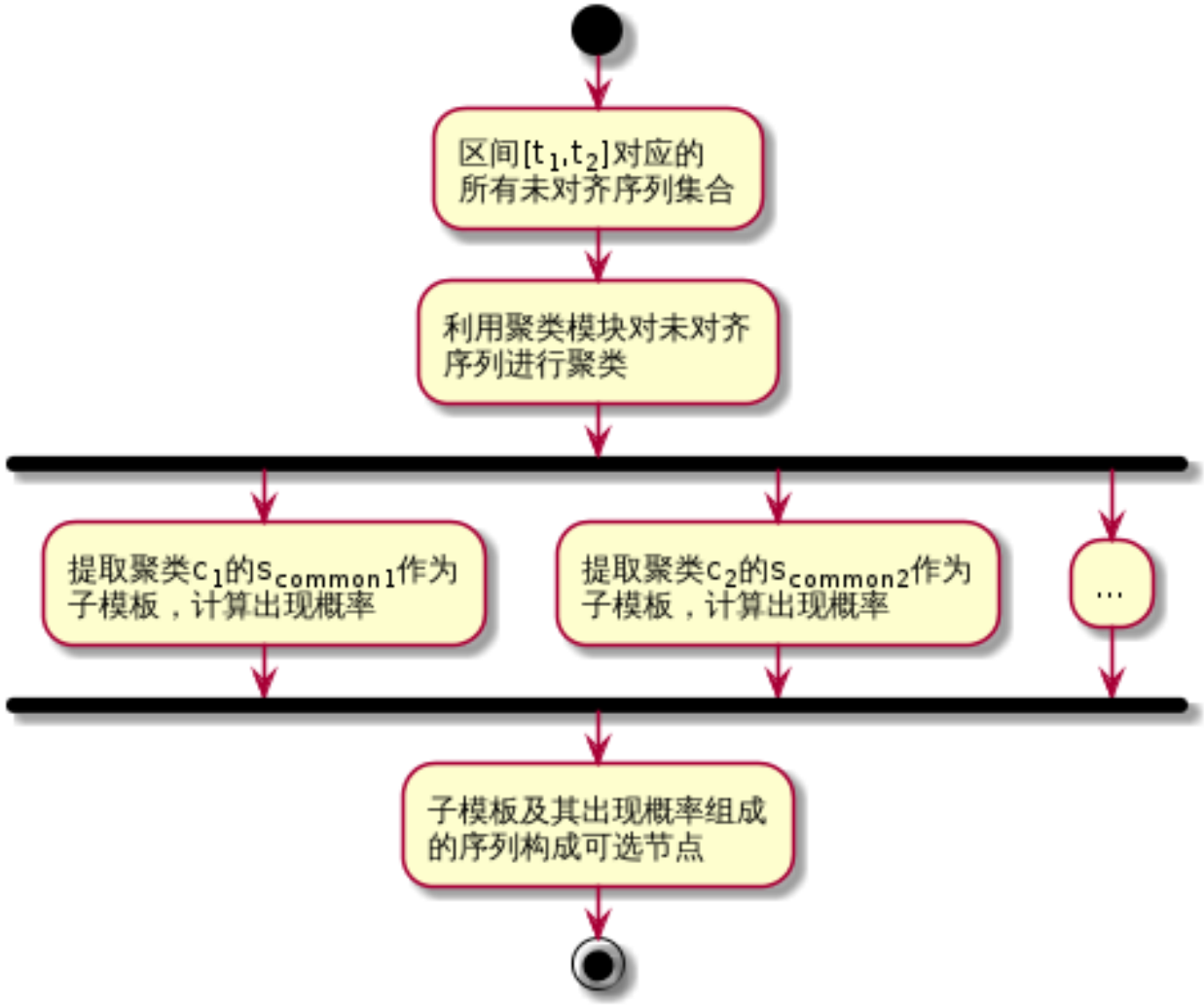
- 对齐的部分则对应生成必选节点，注意将连续的基本节点合并，分别为：a, b, cd, e

可选节点生成的实际实现

- 实际情况要比上面的例子复杂。区间 $[t1, t2]$ 中未对齐的那些标签子序列，有些可能差异很大，有些则可能非常相近，但不完全一样。
- 我们发现，解决构造可选节点的这个子问题和模板提取的问题的流程都可以简单描述为：

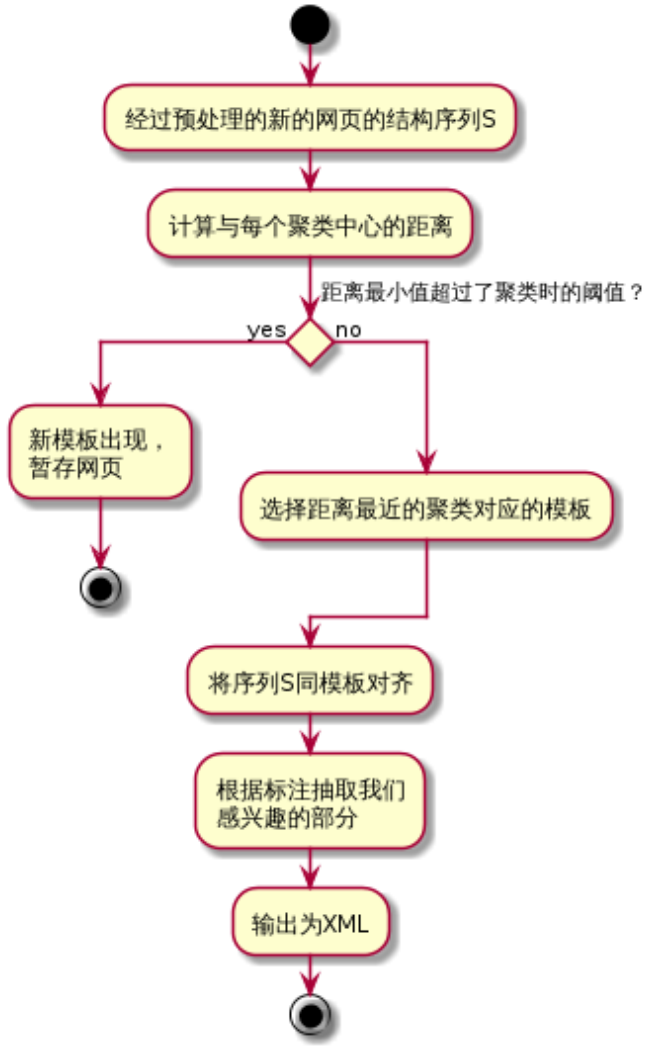
  存在一个序列集合，元素由几种不同的模式生成，先将这些元素分成几种类别，然后针对每个类别去提取“模板”
- 因此，我们可以使用一个“自相似”的框架解决可选节点生成问题。下面，我们把成为构造可选节点时需要提取的公共的模式称为“子模板”。

构造可选节点流程


区间[t_1,t_2]对应的所有未对齐序列集合
利用聚类模块对未对齐序列进行聚类
提取聚类c_1的$s_{common1}$作为子模板，计算出现概率
提取聚类c_2的$s_{common2}$作为子模板，计算出现概率
...
子模板及其出现概率组成的序列构成可选节点

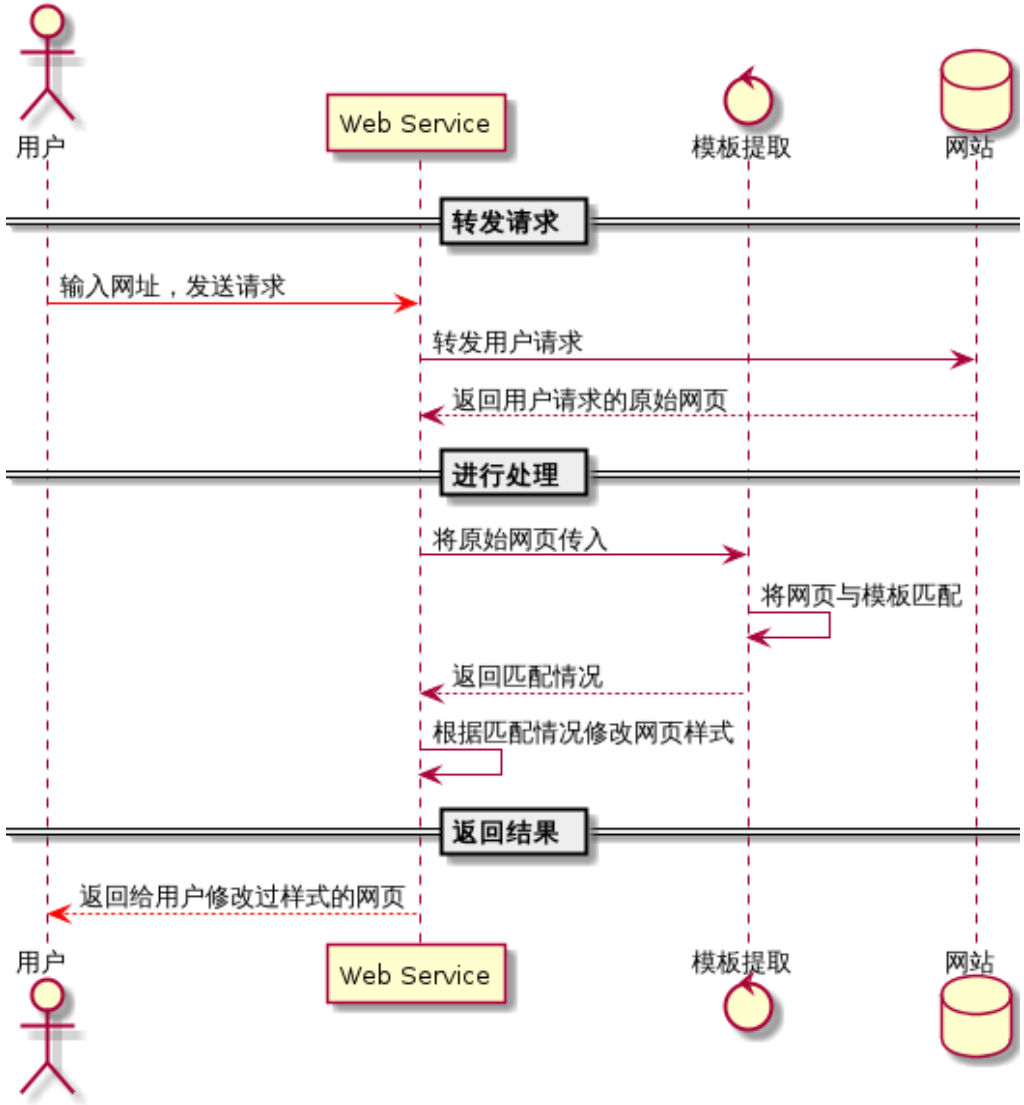
内容提取模块

1. 针对每个模板做一些简单的人工标注，如新闻网页模板的标题，正文，评论等
2. 判断新输入的网页属于哪个聚类，选择对应的模板
3. 模板与网页的序列对齐，根据标注抽取我们感兴趣的部分



模板匹配演示系统

- 基于 Play! Framework(Scala) 实现了一个 Web Service



提纲

实验环境和数据

- 机器的配置情况是：16 个逻辑核的 CPU，24G 内存，操作系统为 64 位的 CentOS。
- 数据统计

	blog	news	other
文件个数	59998	81561	183635
总大小	5.4G	7.9G	18G
来源	blog.sina.com.cn	news.xxx.com	

预处理模块

- 过滤目录页的规则

	blog	news	others
目录页 URL 规则	.*(?<!\\.html?)$	.*(?<!\\.shtml)$	
错误页最大长度	6000	6000	
文件个数	59998	81561	183635
过滤后详细页个数			

- 训练集和测试集分割

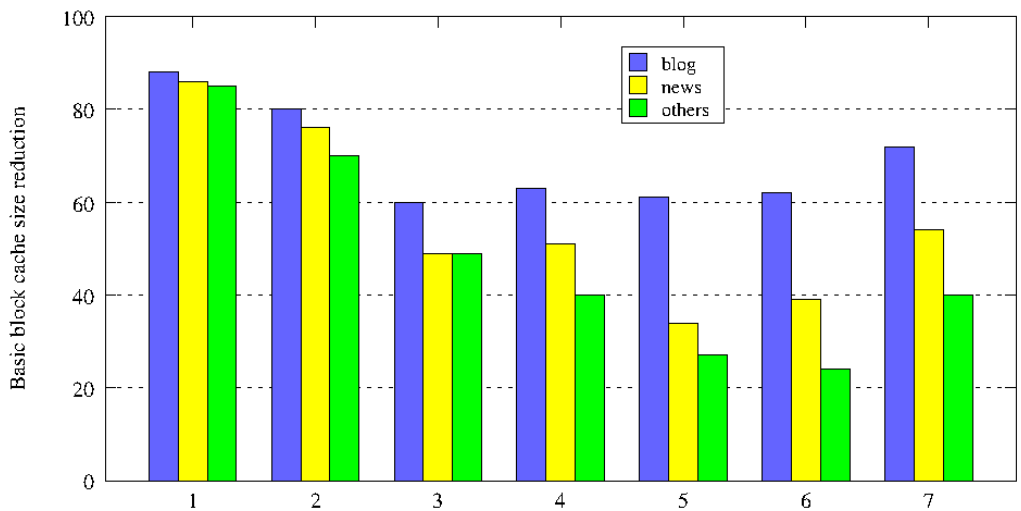
	训练集	测试集	详细页总数
blog			
news			
others			

预处理模块

无用标签去除规则

正则表达式模式	对应的标签名
(?is)<tag.*?>.*?</tag>	style,script
(?is)<[/]?tag.*?>	link,input,br,img,meta,wbr
(?is)<[/]?tag.*?>	strong,em,font,b,p,table

检测出的重复记录统计



聚类和模板提取

- 选择 news 数据集做实验，调整聚类时的阈值，观察聚类结果

阈值	聚类个数	模板长度	必选节点包含的点数	可选节点平均长度
0.3				
0.4				
0.5				
0.6				

- TODO: 结果分析

提取效果

- 我们在测试集上运行我们的程序，抽取出我们需要的信息，并将其保存为 XML 格式输出。

```
<documents>
  <document id=1>
    <title>What's Your Name</title>
    <content>I don't know...</content>
    <comment>Interesting!</comment>
  </document>
</documents>
```

- TODO: 结果统计和分析

演示系统

- 模板匹配演示系统的运行效果如下：

提纲

总结

1. 设计并实现了一个完整的系统，做了很多优化；实现了较高程度的自动化处理，减小人工参与的工作量。
2. 高效地实现了后缀树这个数据结构，在此基础上设计了一套适用于在树的先序遍历序列中查找重复子树的算法。
3. 实现并改进了最长公共子序列算法，将其用于计算文档的结构相似度；实现了简单的凝聚层次聚类算法，并将其用于文档的聚类
4. 实现了一个无监督的模板生成算法，通过人工指定模板中某些部分对应的语义，可以使用模板提取新的由该模板生成的网页中对应的信息。
5. 实现了一个 Web Service，可以直观地看到网页和模板的匹配情况。

未来的工作

1. 预处理模块：改进 Ukkonen 原有的在线构造算法，在构造树的时候就考虑一些原本的结构信息。查找重复时利用后缀树中其他的一些信息，比如后缀链。
2. 网页结构相似度计算和网页聚类模块：可以进一步改进计算相似度的算法，加入一些其他的除了深度之外的更多结构信息；可以考虑使用更复杂但是鲁棒性更好的一些聚类算法。
3. 模板生成和内容提取：对模板生成算法做一些修改，设计一些更高级的机器学习算法。在内容提取的时候，可以采用树的匹配算法而不是序列的匹配算法。