# 一种动态 Web 页面缓存技术*)

A Technology to Caching Dynamic Web Pages

贺 琛[1] 陈肇雄[2] 黄河燕[2]

(南京理工大学计算机科学系 南京210094)[1] (中国科学院计算机语言信息工程研究中心 北京100083)[2]

**Abstract** As a result of the drive of information diversity and the trend of personalization, more and more dynamic pages constitute the contents in the Internet. However, it is difficult to cache the dynamic Web pages because of their dynamic characteristic. In this paper, we propose a technology to solve this problem. It isolates the static and dynamic contents of the page, and stores the original page of the Web server and the information of the background data-base locally, in order to improve the client's response time. Additionally, this technology also adopts corresponding replacement algorithm and coherency policy to track the original page and the data-base update, which tries to synchronize the changes of the dynamic pages in time.

**Keywords** Dynamic pages, Cache, Knowledge base, Differences isolating

## 1 引言

信息多样性的驱动以及互联网个性化趋势,使得越来越多的动态内容被引入 Internet。动态页面的实现主要是通过 CGI(Common Gateway Interface)或类 CGI 程序(如:ASP、JSP、PHP、Perl 等服务器端嵌入式脚本)在服务器上根据客户端 GET 或 POST 方法发来的消息作为参数,并也可与后台数据库交互,动态生成 HTML 文档。所以,这些临时自动生成的页面"总是"被浏览器认为是最新的。故此,目前大多数的缓存系统都无法对动态页面进行缓存。然而由于 CGI 程序本身或其访问的数据库并不总是变化的,因而没有必要每次都要向原始服务器发送请求。

由美国 Wisconsin 大学 WisWeb 项目组提出的主动式缓存(Active Caching)技术[1]试图解决动态页面缓存问题。主动式缓存使用缓存中小程序(Applets)来定制那些用其它方法无法缓存的对象。这些 Applets 通常由与平台无关的编程语言例如 Java 来实现。当用户第一次发出个性化请求时原始服务器会连同被请求的对象及其相关联的 Applets 一起提供给缓存服务器。此后用户提交的相同的个性化请求将由缓存上的 Applets 完成响应。同时 Applets 还能够积累信息并定期将这些信息返回给原始服务器。然而主动式缓存必须改变原始服务器的配置,增加相应的 Applets 功能,实施起来有一定的困难。

我们的思路是:动态页面是由静态和动态内容组成,而动态内容通常是客户端发来的请求信息作为参数,并且往往与后台数据库交互产生来的。所以,通过对页面的差异提取,在缓存中分别存放静态和动态内容,就好像本地模拟出一个(或多个)原始页面服务器中页面的子集和一个(或多个)后台数据库的数据子集一样,从而达到对动态页面本地缓存的目的,即:提高客户端的响应时间。此外,如果采用相应的替换算法和一致性策略,就可以同步原始服务器上原页面的更新和数据库的改动。具体原理见下节。

## 2 原理

为了便于描述,我们有如下定义:

**定义1** 对于请求 $R_1$,$R_2$中的两个 URL:$URL_1$,$URL_2$,若忽略它们的 GET/POST 方法所带参数及"?"后剩余的字符串相同,则称 $URL_1$与 $URL_2$来自同一服务器端页面,记为:$URL_1 \backsim URL_2$;若字符串不相同,则称 $URL_1$与 $URL_2$来自不同的服务器端页面,记为:$URL_1 \not\backsim URL_2$。

**定义2** 对于请求 $R_1$、$R_2$中的两个 URL:$URL_1$、$URL_2$,若 $URL_1 \backsim URL_2$,且它们的 GET/POST 方法所带参数(对于 POST 参数按照 GET 形式附加在"?"后)也相同,则称 $R_1$与 $R_2$相同,记为:$R_1 = R_2$;否则(除 $R_1 = R_2$之外的情况)称 $R_1$与 $R_2$不相同,记为:$R_1 \neq R_2$。

如图1,假设 T 时刻缓存中无动态页面 $O_T$,此时缓存得到请求 $R_1$,通过请求分析器判断请求页面是否为动态页面。如果是则把远端 Web 服务器得到的响应对象存入缓存作为动态页面的基对象 $O_T$,同时将 $O_T$ 发回给客户端浏览器;否则按静态页面处理。

当 $T+\Delta t$ 时刻,缓存收到请求 $R_2$,若 $R_1 = R_2$,缓存直接将基对象 $O_T$ 返回;若 $R_1 \neq R_2$,但 $URL_1 \backsim URL_2$,请求分析器会判断 $URL_2$是否被缓存过,如果没有缓存过,则差分引擎将会调入基对象 $O_T$ 与新返回的对象比较,取其差异部分作为差分对象 $O_{T+\Delta t}$存入缓存,并将返回的新对象发回给客户端浏览器;如果缓存过,则直接调入基对象 $O_T$ 与差分对象 $O_{T+\Delta t}$进行拼装返回客户端浏览器。同时假若动态部分占全部内容的比例大大超过了以往改动的平均水平,就认为生成该页面的程序或后台数据库发生了改变,需要刷新基对象 $O_T$,而不缓存差分对象 $O_{T+\Delta t}$。

以上的过程可以看作把原始服务器上的页面子集(用户访问过的页面集合)以及后台数据库上的数据子集(用户能后访问过的数据集合)存储在本地,当下次发生相同的请求时,根据情况由本地返回数据,从而提高响应时间。下面将具体介

*)本课题得到国家杰出青年科学基金资助(项目编号69925102)。贺 琛 博士生,主要从事人工智能、网络信息处理技术、嵌入式 Internet 的研究开发。陈肇雄 研究员,博导,主要从事人工智能、机器翻译、网络信息处理技术的研究开发。黄河燕 研究员,博导,主要从事人工智能、机器翻译、网络信息处理技术的研究开发。

绍请求分析器、差异提取和替换算法及一致性策略。

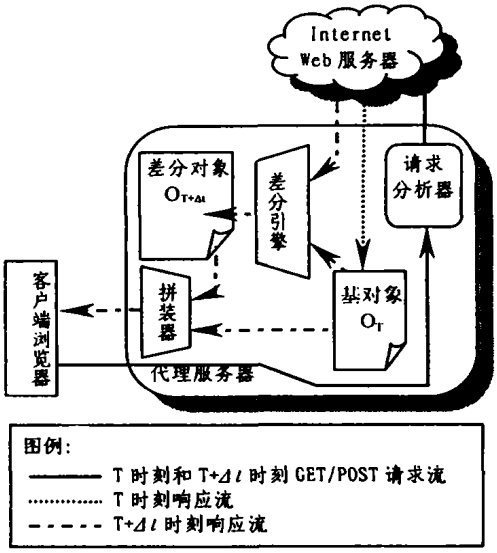


图1 T+Δt 时刻的 CGI 请求/响应过程

### 2.1 请求分析器

请求分析器实际上是一个微型知识库系统[2]，它主要完成对 URL 请求的判别，分析请求的页面是否属于动态页面。该分析器主要由学习机制、后缀库、特例库、人机交互等四个部分构成(如图2)。后缀库负责纪录动态页面的后缀表示形式(如：.asp、.jsp、.php、.pl 等)；特例库负责特殊的动态页面，如无 GET/POST 参数信息的页面以及无后缀的页面等；学习机制负责对输入的请求进行学习提取新的动态后缀或特例页面；人机交互模块是人获取了 Internet 最新技术后以人工方式对后缀库、特例库和规则库进行添加或修改。

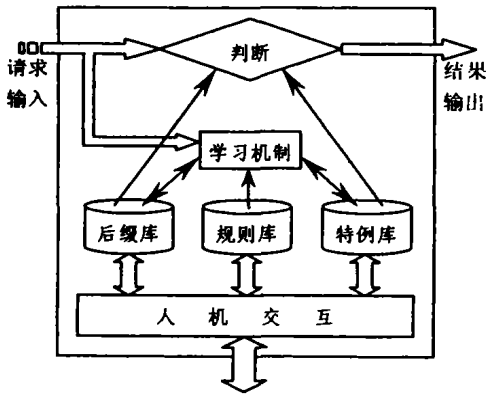


图2 请求分析器

学习机制采用产生式知识表示作为系统的主要知识表示方法，即：如果〈前提〉，则〈结论〉。如下例就是学习机制中一个产生式规则的结构：

“如果页面 URL 包含后缀库中没有的后缀，并且具有 GET/POST 属性，则把该后缀加入后缀库中”。输出信息包括是否为动态页面，是否需要刷新(从 Web 服务器重新取回页面)等。

### 2.2 差异提取

差异提取，就是比较文件后把不同的部分提取出来，这也是差分引擎的主要任务。通常比较文件都是以行为单位[3]，然而有一些页面为了减小文件的大小，去掉了所有的回车/换行(在浏览器中仍然能被正确处理)。为此我们按照 XML/HTML 特征标记“<”和“>”作为分割符号，括在它们中间和在它们外面的字符串作为比较单位，称为字符块(block)。算法基于以下两条文件比较的经验：

**经验1** 在两个页面中，出现且只出现一次的相同块，我们认为是绝对相同的块(只是该块在新页面中被移动了位置)。

**经验2** 如果与绝对相同块紧邻的上下块在两个页面中也相同，那么这些块也被认为是相同的。

基页面 BP、新页面 NP。建立三个数据结构：块表 T、基页面数组 BPA、新页面数组 NPA。用块的字符串存放在 T 中，具体的数据结构如表1，其中 BPC，NPC 分别表示基页面、新页面中某块的个数；BPBNo 表示基页面中对应的块号。

表1 块表

| 字串 | BPC | NPC | BPBNo |
|---|---|---|---|

1. 分别初始化，BPA 和 NPA 以及块表 T

2. 利用经验1，查找 NPC＝BPC＝1的块。即：若 NPA[i] 和 BPA[BPBNo]中块内容相同，则把 NPA[i]设为 i

3. 利用经验2，如果 NPA[i－1]和 NPA[i＋1]分别与 BPA[j－1]和 BPA[j＋1]中块内容相同，则有如下赋值语句：

NPA[i－1]＝ j－1; BPA[j－1]＝ i－1;

NPA[i＋1]＝ j＋1; BPA[j＋1]＝ i＋1;

比较 NPA 和 BPA，在 BPA 中排除指向 NPA 的块，剩下的就是变成 NP 所要删去的块；在 NPA 中排除指向 BPA 的块，剩下的就是变成 NP 所要插入的块。

差异文件采用如下 XML 格式：

```
〈XML〉
〈BPLNo〉Del_Num$_i$〈/BPLNo〉 //要删除的块号
〈Delete/〉
〈BPLNo〉Ins_Num$_i$〈/BPLNo〉 //要增加的块号
〈Insert〉Ins_Content〈/Insert〉 //要增加的内容
〈/XML〉
```

### 2.3 替换算法及一致性策略

我们采用了 Greedy Dual-Size -Frequency (GDS-Frequency) 策略作为缓存替换算法，其计算公式如下：

$$K_i=(C_i/S_i)\times F_i+L$$

$$F_i=\frac{1}{\left(\frac{T_{it}-T_{if}}{nAcess(T_{it},T_{if})}+T_t-T_{it}\right)} \quad (1)$$

其中 $C_i$ 是取回第 i 个对象所花费的代价；$S_i$ 是第 i 个对象的大小；$F_i$ 是第 i 个对象的访问频率；$L$ 是年龄老化因子，它的初始值是0，此后每次替换对象时取 $K$ 的最小值。其具体的算法和解释参看文[4]。

为了保证客户端得到的数据一致性，在维护缓存信息一致性方面，我们提出了一种基于预取机制和用户行为监控相结合的检验方法：

$$\forall\, i \text{ 且 } a_i\in A, \text{当 } t \text{ 时刻}\begin{cases}\text{若 } t-t_i\geqslant TTL_{ai} & \text{则 } l_{ai}\in L_i(L_i\in Q_p)\\ \text{其它} & l_{ai}\notin L_i\end{cases} \quad (2)$$

式(2)说明对于缓存对象集合 $A$ 中任意第一个对象 $a_i$；如果当前时刻 $t$ 与该对象的存入时间 $t_i$ 的差值大于等于 $a_i$ 的

生命周期 $TTL_{a_i}$，则把 $a_i$ 的地址信息 $l_{a_i}$ 加入 $t$ 时刻的更新列表 $L_t$；否则不加入。这里，$TTL_{a_i}$ 就是访问频率 $F_{a_i}$ 的倒数（即：$TTL_{a_i}=1/F_{a_i}$）。

如图3，$Q_p$ 是要发送给原始服务器的预取列表队列。预取队列的优先级低于请求队列（$Q_r$），缓存首先满足来自客户端浏览器的请求，然后预取队列可以在无请求队列的时候发送或网络不拥挤的时候穿插在请求队列中发送。

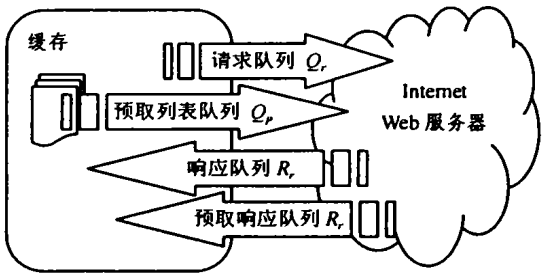


图3 请求队列、预取队列

把响应队列返回的页面与缓存中的同一页面进行差异比较，如果差异部分的比例大大超过了以往改动的平均水平，我们就认为生成该页面的程序或后台数据库发生了改变，需要刷新基对象 $O_T$，而不缓存差分对象 $O_{T+\Delta t}$。同时，记下改动的时间，并根据基对象改动的时间，计算该对象的辅助生命周期 $TTL'_{a_i}$（或者叫后台生命周期）。很明显，$TTL_{a_i}$ 是客户端用户对页面 $a_i$ 的期望周期，而 $TTL'_{a_i}$ 则是一个测量值，同样也不能反映真实的改动周期，所以我们取两者的最小值近似地作为页面的改动周期以同步原页面信息的变化。

## 3 实验与分析

我们在实验室环境下实现了该动态 Web 页面缓存技术的原型系统。实验中我们发现，由于缓存中不仅存放了原始 Web 服务器的页面，同时还存放了后台数据库的数据，因此存储量比一般的静态缓存系统要大很多。为此，我们考虑对于那些动态部分占全部内容的比例70%～90%以上的动态内容（比如搜索引擎返回的页面），我们可以对其压缩以进一步减少存储量。

在某些情况下由于没有足够的学习信息，对于第一次新发现的后缀，系统可能会把不认识的动态页面理解成静态页面，但是在一致性策略范围内是没关系的。

由于某些页面（如 BBS 等）刷新频率很高，因此我们考虑当 $TTL$ 小于某个阈值时就不作缓存，以提高处理速度。

另外，出于安全考虑，在实际系统中对于安全套接字协议层（Secure Sockets Layer，SSL）的对象将不进行缓存。

## 参 考 文 献


1 Cao P，Zhang J，Beach P B. Active Cache：Caching Dynamic Contents on the Web. In：Proc. Middleware '98 Conf. 1988

2 王克宏，汤志忠，胡蓬，等. 知识工程与知识处理系统. 北京：清华大学出版社，1994

3 Heckel P. A Technique for Isolating Differences Between Files. Commun. ACM，1978，21：264～268

4 Arlitte M，et al. Evaluating Content Management Techniques for Web Proxy Caches. In：Proc. of the 2nd Workshop on Internet Server Performance (WISP '99)，Atlanta GA，May，1999


（上接第122页）


10 Lespérance Y，et al. Foundations of a Logical Approach to Agent Programming. In：M. Wooldridge，J. P. Müller，and M. Tambe，eds. Intelligent Agents II-ATAL95，LNAI 1037. 331～346

11 Ng H-K. Topics in High-Level Robot Control：Integrating Planning and Reactivity，and Multiple-Robot Control：[M. Sc. Thesis]. Dept. of Computer Science，York University，2001

12 Hindriks K V，Lesperance Y，Levesque H. An Operational Semantics for the Single Agent Core of AGENT0：[Technical Report UU-CS-1999-30]. Department of Computer Science，Universitry Utrecht，1999

13 Giacomo G D，Lespérance Y，Levesque H J. ConGolog，a concurrent programming language based on the situation calculus. Artificial Intelligence，2000，121：109～169

14 Grosskreutz H，cc-Golog G L. Towards More Realistic Logic-Based Robot Controllers. Proc. AAAI/IAAI 2000. 476～482

15 Reiter R. Sequential，Temporal GOLOG. In：Proc KR 1998. 547～556

16 Hindriks K V，et al. Agent Programming in 3APL. In：Jennings N，Sycara K，Georgeff M，eds. Autonomous Agents and Multi-Agent Systems，2(4)：357～401

17 Hindriks K V，et al. A Programming Logic for Part of the Agent Language 3APL. In：Rash J L，et al，eds. Procs Formal Approaches to Agent-Based Systems，FAABS 2000，LNAI 1871. 78～89

18 Hindriks K V，et al. Semantics of Communicating Agents Based on Deduction and Abduction. In：Dignum F，Chaib-draa B，eds. Proc. of the Workshop on Agent Communication Languages. 105～118

19 Wooldridge M J. A Knowledge-Theoretic Semantics for Concurrent METATEM. In：J. P. Muller，M. J. Wooldridge，N. R. Jennings，eds. Intelligent Agents III，LNAI 1193. 357～374

20 Dix J，Kraus S，Subrahmanian V S. Temporal Agent Programs. Artificial Intelligence，2001，127(1)：87～135

21 McCABE F G，CLARK K L. April-Agent Process Interaction Language. In：Wooldridge，M J，Jennings N R，eds. Intelligent Agents，LNAI 890，324～340

22 Haugeneder H，Steiner D，McCabe F G. IMAGINE：A Framework for building Multi-agent Systems. In：Deen S M，ed. Proc. of the 1994 Intl. Working Conf. on Cooperating Knowledge Based Systems (CKBS-94)，1994. 31～64

23 Poggi A. DAISY：An object-oriented system for distributed articial intelligence. In：Wooldridge M J，Jennings N R，eds. Intelligent Agents，LNAI 890，341～354

24 Hindriks K V，et al. A Formal Embedding of AgentSpeak(L) in 3APL. In：Antoniou G，Slaney J，eds. Advanced Topics in Artificial Intelligence，LNAI 1502，155～166

25 Hindriks K V，Lesperance Y，Levesque H. An Embedding of ConGolog in 3APL：[Technical Report UU-CS-2000-13]. Department of Computer Science，Universitry Utrecht，2000

26 de Vries W，et al. A Truly Concurrent Model for Interacting Agents. In：Intelligent Agents：Specification，Modeling，and Applications，4th Pacific Rim Intl. Workshop on Multi-Agents (PRIMA 2001)，LNAI 2132，16～30