第34卷 第22期 计 算 机 工 程 2008年11月
Vol.34 No.22 Computer Engineering November 2008
·开发研究与设计技术· 文章编号：1000—3428(2008)22—0283—03 文献标识码：A 中图分类号：TP311

基于 AOP 的智能 Web 缓存框架

邓 磊，陈志刚，黄 键，邱 亮
(中南大学信息科学与工程学院，长沙 410083)

摘 要：通过引入面向方面编程技术，提出一种新的智能 Web 缓存框架。描述该框架的组成结构与工作原理，对缓存设计时需要解决的透明性、一致性、替换算法和预取策略等主要问题进行讨论并给出性能测试和分析。实验结果表明，该缓存框架可较大程度提高 Web 应用系统的性能。

关键词：Web 缓存；面向方面编程技术；一致性；替换算法；预取策略

Intelligent Web Cache Framework Based on AOP

DENG Lei, CHEN Zhi-gang, HUANG Jian, QIU Liang
(School of Information Science and Engineering, Central South University, Changsha 410083)

【Abstract】 Through introducing Aspect-Oriented Programming(AOP), this paper proposes a new intelligent Web cache framework ABWC. It describes in structure and principle of ABWC and discusses transparency, consistency, replacement algorithm, prefetching policy and other main problems which need to be solved in cache designing. Significance testing and analysis in it validates that this cache framework can improve the performance of Web application system significantly.

【Key words】 Web cache; Aspect-Oriented Programming(AOP); consistency; replacement algorithm; prefetching policy

1 概述

对 Web 页面进行缓存可缓解 Web 服务器和数据库服务器的压力，提高访问速度。Web 页面缓存分为静态缓存和动态缓存。静态缓存是指在 Web 页面内容更新的同时立即生成静态页面，只适用于更新频率低而并发访问量大的场合，目前已广泛应用于新闻发布等系统中；动态缓存是指并不预先生成静态页面，当用户请求动态网页时才将其网页副本纳入缓存，在下次访问时，不必执行查询操作，而直接由上次保留的副本提供。通常大部分 Web 请求是对动态网页的请求，其响应速度通常要远远慢于对静态网页的请求，因此，对动态 Web 内容进行缓存是减少网络延迟和服务器负载的有效方法。

本文在对传统缓存系统分析和研究的基础上，设计一种基于面向方面编程思想的对动态 Web 内容进行缓存的智能框架(AOP Based Web Cache, ABWC)。ABWC 通过对前端(如 Servlet 引擎)和后端(如 JDBC 接口)2 个接口进行横切拦截，为应用系统透明植入缓存及其缓存管理体系。

2 框架设计

2.1 设计思想

ABWC 的设计目标是为 Web 应用提供一个对开发者完全透明、与业务逻辑及运行平台无关的、具有自我调节及控制能力和良好可扩展性的缓存框架。它基于面向方面的编程技术(Aspect-Oriented Programming, AOP)，采用有效的缓存管理方案来保证缓存的性能。

ABWC 的系统组成结构如图 1 所示，共包括 6 个组件：切点拦截器(Point Interceptor)，查询分析引擎(Query Analysis Engine)，一致性验证引擎(Consistency Verify Engine)，缓存存取管理(Cache Access Manager)，缓存替换管理(Cache Replace Manager)和预取引擎(Prefetching Engine)。

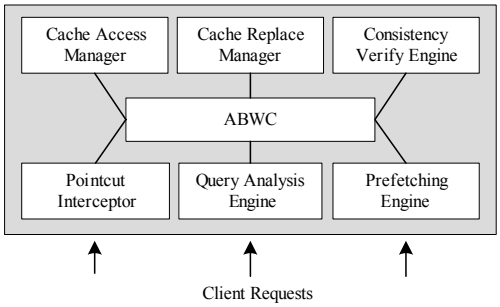

图 1 ABWC 组成结构

切点拦截器通过对前端用户请求和后端 SQL 查询请求 2 个横切关注点实现面向方面编程，将缓存模块独立起来，从而实现与业务逻辑及运行平台脱离。应用系统开发者进行简单配置即可将 ABWC 应用到 Web 系统中，而无须关心缓存的实现。开发者还可通过 ABWC 提供的接口，对 ABWC 进行扩展。

查询分析引擎和一致性验证引擎用以确保缓存的一致性，即保证动态页面的内容与其缓存中的副本内容一致。ABWC 采用了一种新的查询分析的办法来保证缓存的一致性，而缓存替换管理和预取引擎则是 ABWC 的提升缓存性能

基金项目：国家自然科学基金资助项目(60573127)；高等学校博士学科点专项科研基金资助项目(20040533036)
作者简介：邓 磊(1982－)，男，硕士研究生，主研方向：分布式系统，对等计算，Web 服务；陈志刚，教授、博士生导师；黄 键、邱 亮，硕士研究生
收稿日期：2007-12-24 **E-mail：**daly221@163.com

的关键。

缓存存取管理对缓存的内容进行管理，包括缓存结构的维护、索引的重建、缓存的插入与移除等，为其他组件提供基本的操作接口，是 ABWC 框架的基础。

2.2 缓存的存取

ABWC 通过为缓存中网页 URI 建立索引来加快缓存页面的检索速度，同时为了保持缓存一致性，保存了缓存页面对应的 SQL 查询模板和动态参数集。图 2 为 ABWC 的基本存储结构。其中，图 2(a)存储了通过客户端请求的 URI(包括请求参数)索引起来的缓存网页的目录。图 2(b)维护了缓存页面的只读 SQL 依赖信息(查询模板+动态向量值=依赖信息)。当一个写查询产生时，查询分析引擎将决定受更新影响的只读查询集。这些信息将用来移除缓存中的无效缓存页面。

Index：URI	Cached Web Page
URI_1	WebPage1
URI_2	WebPage2
URI_3	WebPage3
…	…

(a)缓存网页目录

Index: SQL String	<Value vector, URI>Pair
$ReadQueryTemplate_1$	$<instance\ values_{1a}, URI_1>$ $<instance\ values_{1b}, URI_{25}>$ $<instance\ values_{1c}, URI_{37}>$
$ReadQueryTemplate_2$	$<instance\ values_{2a}, URI_8>$
$ReadQueryTemplate_3$	$<instance\ values_{3a}, URI_{12}>$
…	…

(b)SQL 依赖信息

图 2 ABWC 的存储结构

在对客户端请求的处理上，先将请求分为只读请求和更新请求(包括插入)，分别进行不同的处理。其中与缓存存取相关的核心机制如下：

(1)缓存检测。对于一个只读的客户端请求，首先检查请求的网页是否在缓存中。如果命中，就把缓存中的网页简单地返回给客户端，结束此次请求的执行。

(2)缓存插入。如果一个只读请求的网页没有在缓存中，就需要在应用服务器上执行查询(SQL 查询要存取数据库)来动态产生 Web 文档，然后返回给客户端；该文档的一个副本将会被保存到缓存中，如果 Web 缓存池已满，将会通过一定的策略替换掉池中的旧文件。

(3)收集一致性信息。对于一个只读请求，为其添加用来产生该请求响应的潜在的数据集(依赖信息)的映射。同样地，对于一个写请求，为与其关联的缓存文档添加失效信息。

(4)缓存失效。对于一个客户端写请求，必须将与此请求关联的缓存目录置为无效，否则会影响缓存的一致性和准确性。此时要使用前面收集到的一致性信息。

2.3 面向 AOP 的缓存

AOP为开发者提供了一种描述横切关注点的机制[1]，并能自动地将这些横切关注点织入到面向对象的软件系统中，从而实现横切关注点的模块化。ABWC采用Spring的动态AOP机制来实现，即通过动态Proxy模式，在目标对象的方法调用前后插入相应的处理代码。在ABWC中实现AOP主要包括以下步骤：

(1)方面分解。业务逻辑处理和方法跟踪，前者为一般关注点，后者为横切关注点。如图 3 所示，通过在应用的前端和后端建立 2 个横切关注点来建立 Web 缓存及其管理体系，其中，Pointcut A 为客户端请求横切关注点，Pointcut B 为 SQL 请求横切关注点。

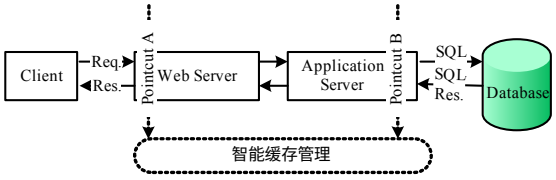

图 3 基于 AOP 的动态缓存管理

(2)关注点实现。以前端的客户请求为例，通过捕获 Servlet 的 main 方法来注入缓存检测、插入和失效等操作。Java 的 Servlet 是通过标准的 API 实现的，主要方法包括实现 HTTP GET 和 POST 的 doGet 和 doPost 方法。

(3)方面重新织入。在 Spring 的配置文件中进行配置，使用 Spring 的 IOC 机制来装载和管理 ABWC 实例，通过读取配置文件得到 ABWC 的实例，从而减少了应用程序与缓存管理之间的耦合。

3 智能缓存管理

3.1 维持缓存一致性

为保证缓存的一致性和有效性，必须及时将缓存中的无效页面清除出缓存。确定一个更新请求要使哪些缓存中页面失效，可将其产生的 SQL 查询与缓存页面的 SQL 查询进行关联性验证。本文实现办法包括使用查询分析引擎和一致性验证引擎，用来决定 SQL 查询之间的依赖性，并将与更新请求关联的缓存页面置为无效。查询分析主要由 2 个部分组成：

(1)确定查询之间可能的依赖

SQL 查询作为模板提供，如果一个只读查询模板跟一个更新查询模版有共同的表和字段，则存在依赖性。

(2)采用交集测试暴露真实的依赖

如果一个更新查询修改了一个或多个将要被读取的数据行的某些字段，则该更新查询就与只读查询存在依赖。

关联分析需要系统开销，因此，本文的查询分析引擎必须在关联判断的精度和关联评估的开销之间寻找平衡。ABWC 目前支持以下 3 种缓存失效策略，另外还可通过自定义查询分析策略提高分析的精度。

(1)简单字段检测方法。在只读查询中使用更新查询中更新的字段，则更新查询和只读查询之间存在交集。但这种只根据字段检查的方法不一定准确。如“ SELECT *a* FROM *T*… ”与“ UPDATE *T* SET *a*=new_val… ”存在交集；而“ SELECT *a* FROM *T*… ”与“ UPDATE *T* SET *b*=new_val… ”不存在交集。

(2)为使交集测试更加精确，用只读查询的 Where 字句的选择字段与更新查询的值进行匹配来确定更新的行数是否相同。如：“ SELECT *c* FROM T WHERE *a*=*X*… ”与“ UPDATE *T* SET *c*=new_val WHERE *a*=*Y*… ”，$X \neq Y$ 则不存在交集。

(3)通过执行额外的找回遗漏数据的查询可使失效检测更加精确，但增加了额外的开销。如：“ SELECT *c* FROM *T* WHERE *a*=*X*… ”和“ UPDATE *T* SET *c*=new_val WHERE *d*=*W*… ”，通过字段检测并没有关联的值。因此，为字段 *a* 生成一个查询来测试：“ SELECT *a* FROM *T* WHERE *d*=*W* ”，如果返回值等于 *X*，则只读请求和更新请求有交集。

3.2 缓存的替换策略

缓存替换管理是缓存管理的核心，是提升缓存性能的关键。由于Web缓存空间有限，当存放了一定数量的网页副本后，为保存新的客户端请求响应的副本，只能将已经保存的

一些副本替换出缓存。常用的Cache替换算法有FIFO算法、SIZE算法、LRU算法以及LFU算法等[2-3]。ABWC默认提供了LRU-K缓存替换算法。LRU-K算法是LRU算法的改进，在K=2时，性能提高比较明显，且实现代价小，整体性能最好[4-5]，所以ABWC采用LRU-2 算法，同时提供替换策略自定义接口。

3.3 自动预取策略

预取策略需要控制 2 个方面才能得到良好的效果。一方面需要控制对哪些页面进行预取，另一方面需要控制预取的量和进行预取的时刻，不能对其他应用产生较大的影响。为在系统资源使用和用户访问延迟之间做出权衡，预取策略先预测将要被访问的页面，然后只选择其中一部分下载，这项工作由预取引擎完成，自动实时地确定 Web 服务器的预取门限来决定应预取的文件。

预取引擎的工作原理如图 4 所示。页面预测模块根据页面访问的历史统计信息，计算出用户请求当前页面后将要访问页面的概率大小反馈给页面预取模块。观测器模块实时监视包括系统负载和 Web 服务器负载在内的各种系统信息，计算出预取门限 H 反馈给页面预取模块。页面预取模块根据访问概率和预取门限 H，自动预取所有访问概率超过门限 H 的文件。

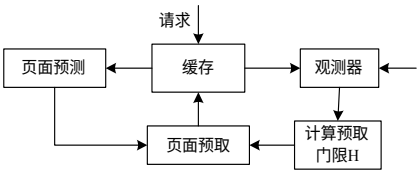

图 4　预取引擎工作原理

页面预测模块为了确定对哪些页面进行预取，需要采用预测算法。用户对Web的访问有一定规律，而且具有历史性和相对集中的爱好。在ABWC的预取模块中使用 2 类计数器：页面访问计数器和链接计数器。每一个页面A有一个相应的页面计数器C_A，$C(A, B)$表示页面B与A有直接的超链接数。每次页面A被访问时C_A加 1；当用户通过A访问B页面时，$C(A, B)$加 1。条件概率$p(B|A)$表示用户在访问页面A后将要访问页面B的概率。当页面A正被用户阅读时，对于链接到缓存上的页面B_i的访问概率为

$$P(B_i|A)=C(B_i, A)/C_A$$

预取的另一方面是对预取的控制，即选择适当的预取时间和预取的量。预取需要付出代价，在系统负载较重时，过量预取会导致用户的正常请求不能得到及时响应，性能反而降低；而当系统负载很小时，预取并不能节约更多的时间，性能并不会直线升高。ABWC 采用了门限算法，通过计算门限 H 来确定需要预取的量，具体算法参考文献[6]。

4　性能测试与分析

本文为一个基于 J2EE 的在线考试系统透明植入缓存 ABWC，使用 JMeter 进行性能测试。在测试中定义了 10 种不同的动态交互请求，其中约有 85%的请求为只读请求。测试的服务器采用 IBM xSeries 260，Web 服务器为 Apache2，应用服务器为 Jboss4.0.4GA，数据库为 MySQL V5.0。

首先测试 ABWC 在减少响应时间上的作用。在线考试系统在使用 ABWC 前后的平均响应时间测试结果如图 5 所示。在并发客户请求数较少时，ABWC 的作用并不明显，甚至比不使用 ABWC 效果还差，但随着并发访问数不断增加，不使用 ABWC 的系统响应时间急剧攀升，而使用 ABWC 的系统响应时间走势要平稳很多，在并发数达到 1 000 时，使用 ABWC 的系统响应时间减少了 53%左右。可见，在高并发的 Web 应用中，使用 ABWC 可有效减少响应时间。

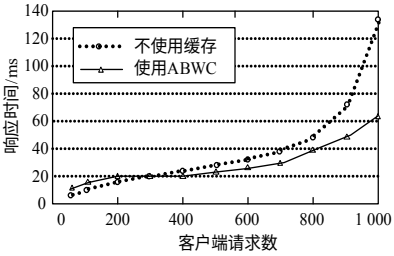

图 5　响应性能测试结果

在随后的缓存命中率测试中，在 1 000 个并发访问数的情况下，对系统预热 15 min 后进行计数，平均缓存命中率为 43.8%，达到了较高的水平。

表 1 比较了考试系统、ABWC 库文件和 ABWC 配置文件的代码规模，缓存框架包括 6 大组件在内的大部分代码都封装在 ABWC 库文件中，库文件还实现了供不同应用重用的接口，配置文件的代码规模非常小，可轻松移植到不同的应用中。可见 AOP 技术对 ABWC 的提升作用。

表 1　系统各部分代码规模

Web Exam App.		ABWC Library		AOP-based deploy
类文件个数	代码行数	类文件数	代码行数	配置代码行数
125	3.2×10^4	15	5.3×10^3	30

5　结束语

本文提出一种基于 AOP 的 Web 缓存框架 ABWC，它使用 AOP 技术提高了框架的可移植性，可透明地为不同的 Web 应用植入动态页面缓存，简化了框架的使用和维护。同时通过使用 LRU-K 缓存置换算法和自动预取策略，ABWC 具备智能而高效的缓存管理机制，提升了缓存的性能，其维护缓存一致性的方法也较为创新。

下一步的研究工作是对缓存框架进行优化，对查询分析、置换算法和预取策略进行改进，同时加强各个组件间的协作，进一步提高框架的性能。

参考文献

[1] Kiczales G. Aspect-oriented Programming[C]//Proc. of ECOOP'97. [S. l.]: Springer-Verlag, 1997: 220-242.

[2] Lorenzetti P, Rizzo L. Replacement Policies for a Proxy Cache[R]. University di Pisa, Tech. Rep.: LR2960731, 1996.

[3] Williams S, Abrams M. Removal Policies in Network Caches for World Wide Web Documents[C]//Proc. of ACM SIGCOMM'96. Stanford, CA: [s. n.], 1996: 293-305.

[4] O'neil E J, O'neil P E, Weikum G. The LRU-K Page Replacement Algorithm for Database Disk Buffering[C]//Proc. of International Conference on Management of Data. Washington D. C., USA: [s. n.], 1993: 297-306.

[5] O'neil E J, O'neil P E, Weikum G. An Optimality Proof of the LRU-K Page Replacement Algorithm[J]. Journal of the ACM, 1999, 46(1): 92-112.

[6] Jiang Zhimei, Kleinrock L. An Adaptive Network Prefetch Scheme[C]//Proc. of IEEE International Conference on Communications. Montreal, CA: [s. n.], 1997.