基于 B/S架构的 WEB应用中关联实体集操纵方法及其比较

邓晓飞 赵 雷 杨季文 韩月娟
(苏州大学计算机科学与技术学院 江苏 苏州 215006)

摘 要 总结 B/S架构下 WEB应用系统开发中经常遇到的操纵关联实体集的问题。定义关联实体集的概念,并结合目前流行的技术手段,例如 Ajax,给出了几种实现方案。讨论方案的实现原理,并利用开源的负载测试工具 Jmeter从多方面进行测试分析,得出各个方案的性能对比,最终给出方案的建议使用情景。

关键词 关联实体集 负载测试 Jmeter XMLHttpRequest Ajax

MANIPULATION APPROACHES FOR ASSOCIATED ENTITY SET IN WEB APPLICATION AND THEIR COMPARISON BASED ON B/S ARCHITECTURE

Deng Xiaofei Zhao Lei Yang Jiwen Han Yuejuan
(School of Computer Science and Technology, Soochow University, Suzhou 215006, Jiangsu, China)

Abstract In the paper, we summarized the issue of manipulating associated entity set frequently encountered in web application system development based on B/S architecture, defined the concept of associated entity set and put forward several popular solutions, for example, Ajax, for this issue. Then we discussed the design and implementation theories of these solutions, and employed Jmeter, which is an open source tool for web application load testing, to perform the testing analysis on multiple aspects. Finally, we gave out some suggestions on when to use these solutions in web application development after obtaining the performance comparison from the test results.

Keywords Associate entity set Loading test Jmeter XMLHttpRequest Ajax

0 引 言

现实世界的事物之间,有时存在着某种从属关系。开发 WEB应用程序时也常常遇到诸如:选择公司某部门,列出该部门下属子部门清单的问题。现实世界中的事物是可以抽象成实体的,将部门和子部门分别抽象成实体,则所有部门的实体集合同所有子部门的实体集合之间存在严格的从属关系,即:上一级实体集的某个实体可以严格确定下一级实体集的一个子集。定义这种有着严格从属关系的实体集为关联实体集,两个关联实体集也称为 2级关联实体集,3个或 3个以上的关联实体集称为多级关联实体集。

我们知道 WEB应用是基于 B/S架构的,它建立在 HTTP无状态协议的基础之上,不可能像 C/S架构的应用程序一样有着丰富的客户端表现。但是自从 WEB出现以来,人们一直努力追寻着具有丰富客户体验的 WEB应用程序。在开发 WEB应用程序时如何操纵关联实体集,如何选用恰当的方法使得客户得到更好的体验,往往成为 WEB应用程序需要重点关注的问题。实现 WEB应用中操纵关联实体集的问题具有一定的通用性,而且实现这种操作有很多方法,如下:

(1) 方案 A 不断刷新整个页面,获取下一级实体集,刷新时需保存用户当前工作;

(2) 方案 B 一次性获取所有实体集,客户端处理所有业务逻辑和控制逻辑;

(3) 方案 C 利用 Iframe中嵌入的子页面的刷新来获取下一级实体集,客户端处理部分业务逻辑和控制逻辑;

(4) 方案 D 利用 Ajax获取下一级实体集,客户端处理部分业务逻辑和控制逻辑。

为了更加清楚地比较这些方法,本文以下拉菜单为例来进行说明。

1 实现原理

1.1 问题描述

假设:某所综合性大学里有若干的院系所 (设某个院系所为 YXS_i),每个院系所又包含若干的专业 (设某个专业为 ZY_i),每个专业又包含若干的研究方向 (设某个研究方向为 $YJFX_i$)。

现用如下方式描述院系所、专业、研究方向这三种实体:

$YXS_i = < D_i, Name_i >$,其中 D_i是院系所编码,$Name_i$是院系所名称;

$ZY_i = < D_i, Name_i >$,其中 D_i是专业编码,$Name_i$是专业名称;

$YJFX_i = < D_i, Name_i >$,其中 D_i是研究方向编码,$Name_i$是研究方向名称。

以上三种实体可以分别组成三种实体集,院系所实体集、专业实体集、研究方向实体集,分别用如下方式描述:

$entity\_YXS = \{ YXS_i \}$ $i = 0, 1, 2, \cdots$

收稿日期: 2007 - 11 - 26。高等学校博士学科点专项科研基金 (20060285008)。邓晓飞,硕士生,主研领域:数据库,权限管理。

entity_ZY = { < ZY_i, $BelongCode_j$ > },其中 $BelongCode_j$是该专业所属院系的编码, i, j = 0, 1, 2, …。

entity_YJFX = { < $YJFX_i$, $BelongCode_j$ > },其中 $BelongCode_j$是该研究方向所属专业的编码, i, j = 0, 1, 2, …。

根据关联实体集的定义, entity_YXS和 entity_ZY是 2级关联实体集, entity_ZY和 entity_YJFX也是 2级关联实体集, entity_YXS、entity_ZY和 entity_YJFX是多级关联实体集。通过一个院系所编码 D_k (j = k)和专业实体集 entity_ZY中每个具体专业的 $BelongCode_j$可以确定属于该院系所的专业子集合,即 entity_ZY的子集合:

$$\{ < ZY_i, BelongCode_j > | BelongCode_j = D_k \}$$

从以上描述可以看出,这种下拉菜单并不是几个下拉菜单在 WEB页面中位置上的简单组合。上级菜单选择好以后,需要重新确定下级菜单的实体集,这两个菜单所对应的实体集之间是关联的,称这种操纵关联实体集的下拉菜单为级联下拉菜单。

1.2 方案 A:刷新整个页面获取实体集

首先讨论一种方案,设为方案 A。先获取院系所实体集 entity_YXS,接着刷新页面将用户选择好的某个院系 D_K提交给服务器,服务器再返回客户端专业实体集 entity_ZY。按照这种方式,不断刷新整个页面,取回相应实体集。这种方案的缺陷比较多,难免被用户和开发人员抛弃:向服务器提交请求的过程中用户当前工作被打断;保存用户当前工作的临时数据都存储在 WEB服务器上;频繁刷新加重了 WEB应用服务器和数据库服务器的负担;对网络流量造成一定影响 (下文将不对方案 A进行测试、比较)。那么能不能一次获取所有数据呢?

1.3 方案 B:一次获取所有实体集

方案 B:将院系所、专业、研究方向实体集一次性全部发送到客户端,利用客户端脚本语言,如: Javascript,操纵 HTML文档对象中的相关元素向用户呈现不同的实体集。 entity_YXS、entity_ZY、entity_YJFX实体集将被完全加载到客户端,并以 JavaScript数组形式或者以 XML文件形式存储。加载所有的实体集信息,数据量很大;另外,页面加载完成之后将由客户端来承担处理所有的业务逻辑和控制逻辑,这些处理逻辑也必须存储在客户端。苏州大学 2007年博士研究生网上报名系统就采用了方案 B。

1.4 方案 C:利用 Iframe获取实体集

方案 C:基本思路仍然是利用传统的请求/响应模式。由于方案 A的失败之处在于频繁地刷新用户当前工作页面。方案 C进行了改进:在页面中嵌入 Iframe[5]来模拟与服务器之间的异步交互[1],在此过程中利用 Iframe中嵌入的页面请求和接收数据。即:用户当前工作页面不进行刷新,该页面的 DOM树元素基本没有改变;刷新的页面是嵌入在当前页面中的某个隐藏页面——一个独立的文档布局,该隐藏页面的 DOM树元素刷新后发生了改变。

1.5 方案 D:利用 Ajax获取实体集

方案 D:利用 Ajax[1,4,5,6]完成级联下拉菜单的功能。首先假设客户选择好一个院系所 D_K,通过院系所下拉菜单的 onchange事件触发客户端的处理逻辑,该处理逻辑会创建一个 XmlHttpRequest[1,6]对象 (简称 XHR对象)。客户端通过 XHR对象和服务器端 servlet[7,8]进行交互,即发出一个 HTTP请求。服务器处理逻辑处理好该请求后,会将响应数据通过 HTTP协议发送给客户端。在初次创建 XHR对象时, JavaScript处理逻辑会指定一个回调函数[5]来处理 XHR对象获得的响应结果,向客户呈现"化学化工学院"所属的所有专业信息。以上"请求/响应"过程并没有刷新用户当前工作页面,采用的是真正的异步交互方式[6]。目前出现了许多开源的 Ajax组件以及专门针对 Ajax的设计模式,使用 Ajax进行开发、维护效率很高。

2 测试分析

下面将对采用 B、C、D三种方案实现的 WEB应用进行测试。为了简化测试,下文中仅讨论 2级关联实体集。以下出现的 Jmeter测试参数的解释请参考 Jmeter相关帮助[9]。

测试采用 Jmeter在测试机上发出 HTTP请求,服务器响应请求,先执行 Action[8]获取 JSP页面加载时初始化的数据 (即:上一级实体集,如,院系所实体集 entity_YXS),然后该 Action跳转到 JSP页面,客户端加载页面。页面加载完成后,用户选择院系所实体集,客户端再次发出请求,服务器响应并执行对应的 Action(另一个 Action),获取下一级实体集,如专业实体集 entity_ZY。因此对每个方案的测试分为两部分:页面加载 (获取上一级实体集)、页面加载完成后获取下一级实体集。

2.1 稳定状态下性能度量

一般选在稳定状态度量性能,设置 Jmeter的线程组中虚拟用户数为 1,持续运行时间 60秒。表 1为三种方案页面加载的性能监测数据,表 2为页面加载完成后获取下一级实体集时的性能监测数据。每次测试的测试数据分为两部分: Jmeter在客户端所监测到的数据、Websphere服务器的 Tivoli性能查看器 (TPV)[10]所监测到的数据。

表 1 加载页面的性能监测数据

Jmeter监测数据				
Method	Samples	Average	KB/sec	Avg. KB
B	133	163	263.35	118.74
C	202	2	11.46	3.39
D	197	3	25.64	7.76
TPV监测数据				
Method	Page load service Time	Jdbc UseTime	Action Service Time	
B	140.02	162.4	167.87	
C	0.25	2.31	2.69	
D	0.42	2.41	2.85	

表 2 页面加载完成后获取下一级实体集的性能监测数据

Jmeter监测数据				
Method	Samples	Average	KB/sec	Avg. KB
C	182	1	4.45	1.46
D	193	1	0.54	0.17
TPV监测数据				
Method	Page load service Time	Jdbc UseTime	Action Service Time	
C	0.13	3.18	3.44	
D	-	3.06	3.22	

以上两表格中 JDBC UseTime为 JDBC操作平均处理时间,

理论上包含了数据库服务器处理时间、数据库服务器和 WEB服务器之间的网络传输时间。加载页面之前需要先执行 Action获取初始化 JSP页面的数据,因此 Action ServiceTime在表 1中指:加载页面之前获取页面初始化数据的 Action的平均处理时间;而在表 2中指:获取对应页面下一级实体集数据的 Action的平均处理时间。表 1中"加载页面 ServiceTime"为 Action获取完页面初始化数据之后服务器处理页面的平均时间。"刷新 Iframe嵌入页面 ServiceTime"为 WEB服务器处理嵌入在 Iframe中的页面刷新动作的平均处理时间。

设 T'_{avg}为 Jmeter测得的平均响应时间,方案 i的平均响应时间为 $T'_{avg}(i)$,下文用类似方法指明是哪个方案的变量; $T_{network}$为 WEB服务器和客户端之间网络传输时间; $T'_{network}$为 WEB服务器和数据库服务器之间网络传输时间; T_{page}为服务器处理需要加载的 JSP页面的平均处理时间; T_{JDBC}为 JDBC操作平均处理时间。那么,平均响应时间可以表示为:

$$T_{avg} = T_{network} + T_{page} + T_{JDBC} + T'_{network} \quad (1)$$

执行 Action初始化 JSP页面(包括获取上一级实体集数据)或者获取下一级实体集数据时,会从数据库服务器获取数据,其中包含了 JDBC操作时间和网络传输时间,则:

$$T_{Action} = T_{JDBC} + T'_{network} \quad (2)$$

$$T_{avg} = T_{network} + T_{page} + T_{Action} \quad (3)$$

方案 B的页面初始化时需要多次访问数据库获取所有的实体集, $T_{JDBC}(B)$为 162.4毫秒;方案 C、D的页面初始化时只需要获取院系所实体集, T_{JDBC}小于 2.5毫秒。方案 B在页面加载完成后由客户端处理所有的业务逻辑,页面包含所有数据和大量的客户端脚本,WEB服务器处理该 JSP页面需要相对较长的时间: $T_{page}(B)$为 140.02毫秒,因此:

$$T_{avg}(B) = T_{network}(B) + 308.07\text{毫秒}$$

而最终 Jmeter测得 $T'_{avg}(B)$仅为 161毫秒,这是因为 Websphere提供了一系列的优化机制,如: JVM优化器、高速缓存机制[10](对象高速缓存、servlet高速缓存),使得 Jmeter测得的响应时间大大缩短。总之,方案 B的 JDBC操作时间和 JSP处理时间相对较长,导致响应时间 $T_{avg}(B)$最大,性能表现最差。

方案 C、D加载页面时, JDBC操作时间相差不大。但是,方案 D为了使客户端能够处理部分业务逻辑,页面含有大量的客户端脚本,影响了 WEB服务器处理该 JSP页面的速度, $T_{page}(C)$为 0.42毫秒,是方案 C的 1.5倍以上。在网络带宽、网络服务器处理速度等不是限制因素时,网络传输时间还受传输文件的大小影响。此外:

$$Avg\ KB(D) = 7.76kb \approx 2Avg\ KB(C)$$

即平均每次请求客户端所接收到的数据量较大,因此:

$$T_{network}(C) < T_{network}(D)$$

则:

$$T_{avg}(C) = T_{network}(C) + 2.94 < T_{avg}(D) = T_{network}(D) + 3.27$$

Jmeter所测 $T'_{avg}(C)$也小于 $T'_{avg}(D)$。以 KB/sec所表示的吞吐量来衡量,页面加载时对服务器的负载和网络的影响由大到小依次为:方案 B、D、C。

从页面加载后的性能表现来看,自然是方案 B最佳, $T_{avg}(B)$为 0。由于 Websphere的优化机制以 $T'_{avg}(C)$和 $T'_{avg}(D)$比较不出方案 C、D的性能。另外: $T_{JDBC}(C) \approx T_{JDBC}(D)$。方案 C采用刷新嵌入在 iframe中的子页面的方法获取数据,除了需要获取初始化数据(纯数据),还要获取大量的关于这个页面的文档布局信息(内容),则 Avg KB(C)远大于 Avg KB(D),从而: $T_{network}(C) > T_{network}(D)$。而方案 D采用 Ajax,服务器响应数据是纯粹的数据,因此 WEB服务器不需要再花额外的时间处理任何 JSP页面,即 $T_{page}(D) = 0$。那么, $T_{avg}(D) = T_{network}(D) + 3.22 < T_{avg}(C) = T_{network}(C) + 3.57$。即,页面加载后,方案 D性能表现略高于方案 C。以 KB/sec所表示的吞吐量来衡量,页面加载时对服务器的负载和网络的影响由大到小依次为:方案 C、D、B。

2.2 并发用户下性能度量

负载测试(Loading Testing[2,11])方法的主要目的是找到系统处理能力的极限[11]。本文并不关心服务器软、硬件资源的瓶颈;关注的重点在于资源达到饱和前不同实现方案下应用程序的性能。设置 Jmeter线程组中虚拟用户数从 5、10、15逐渐增加到 60,每秒钟启动 5个线程。

当虚拟用户数超过 35时,图 2中吞吐量曲线 B的走势越来越平缓,图 1所示响应时间曲线 B的抖动也趋于稳定。这说明虚拟用户超过 35个时,采用获取所有数据的方案 B使得服务器处理资源达到饱和。图 2中曲线 C、D基本吻合,即方案 C、D的吞吐量(Throughput)接近。方案 C、D对服务器负载较小,其响应时间、吞吐量(Kb/sec)对比可以参考图 3、图 4。

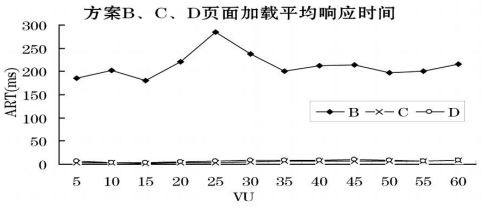

图 1 方案 B、C、D页面加载平均响应时间(ART)

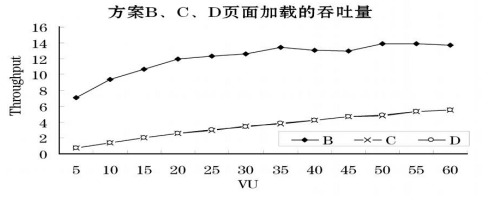

图 2 方案 B、C、D页面加载吞吐量(Throughput)

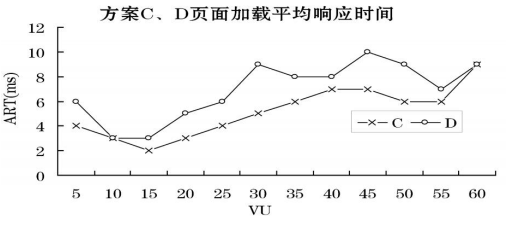

图 3 方案 C、D页面加载平均响应时间(ART)

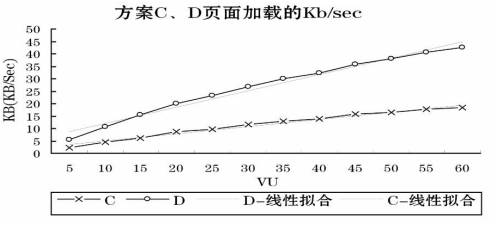

图 4 方案 C、D页面加载吞吐量(Kb/sec)

从图 3中曲线 C、D可以看到,页面加载的平均响应时间,方案 C比方案 D略快,与 2.1节中单用户持续运行所得结论基

本一致。对图 4中方案 C、D的点进行线性拟合,分别得到直线公式: Kb/sec(C) =3. 3123VU +5. 24和 Kb/sec(D) =1. 4452VU +2. 1153,其中 R^2分别为 0. 9835和 0. 9823非常接近于 1,说明拟合度非常好。上述两个公式表明:随着虚拟用户数的逐渐增加,方案 C、D的页面加载时吞吐量 (Kb/sec)是线性增长的,方案 D由于采用 Ajax需要在加载页面中加入大量客户端处理逻辑,增长较快,对服务器和网络影响也较大。但是,以上所讨论的和下面将要讨论的结论都是建立在服务器资源没有达到饱和的基础之上。页面加载完成后的响应性能请参考图 5、图 6。

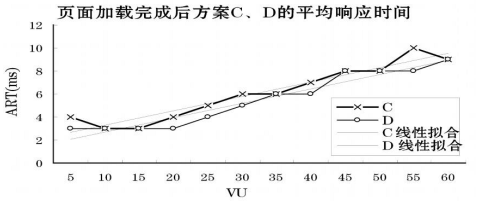

图 5　页面加载完成后方案 C、D的平均响应时间

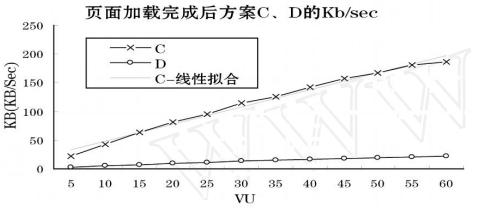

图 6　页面加载完成后方案 C、D的吞吐量 (Kb/sec)

图 5中曲线 C、D分别表示:方案 C、D在页面加载完成后,获取下一级实体集时,所耗费的平均响应时间 (ART)。对图 5中方案 C、D的点进行线性拟合,分别得直线公式:

$$ART(C) = 0.6259VU + 2.0152$$

$$ART(D) = 0.6224VU + 1.4545$$

R^2分别为 0. 9195和 0. 9388接近于 1,拟合度较好。这说明了,此时 ART是线性增长的;直线 ART(C)和 ART(D)的斜率比较接近表明:方案 C、D的 ART增长趋势接近;但是 ART(D)的截距较小,因此方案 D的响应时间较小,性能略高,这点也体现了 Ajax响应数据是纯数据的优势。

对图 6中方案 C、D的点进行线性拟合,分别得出如下两个直线公式: Kb/sec(c) =15. 01VU +17. 493和 Kb/sec(D) =1. 7222VU +2. 0592,其中 R^2分别为 0. 9886和 0. 9868,非常接近于 1,拟合度非常好。直线 Kb/sec(D)的斜率和截距比直线 Kb/sec(C)的斜率和截距小得多,因此,随着虚拟用户数的逐渐递增,方案 D比方案 C的吞吐量 (Kb/sec)小且增长更缓慢,对服务器负载和网络的影响也小。

2.3　测试分析结论

综上分析,方案 B的优点在于页面加载成功后就不存在任何与服务器性能有关的问题了。然而,这种做法使得页面加载响应性能相对较差,对服务器负载较重,同时对网络的影响也很大。随着并发用户数的逐渐增加,服务器会很快达到饱和状态。但是,在总数据量较小的情况下特别适合采用该方案。

方案 C、D应该从加载页面和页面加载后两方面比较。加载页面时主要是获取上一级实体集的数据,方案 D利用了 Ajax需要在页面中添加大量的客户端代码,而方案 C的客户端代码则简单得多。所以,页面加载时的性能是方案 C略高一点,随着并发用户数的增加,方案 D对服务器负载和网络的影响较大。

页面加载完成之后,获取下一级实体集时,方案 D利用 Ajax交互,服务器响应数据是纯数据,比方案 C响应整个文档的信息要小得多。因此方案 D响应性能要比 C高,随着并发用户逐渐增长,对服务器和网络的影响比较小。关于这三种方案的总体比较参考表 3所示。

表 3　B、C、D三种方案总体比较

	页面加载			获取下级实体集			实际开发	
方案	响应时间	服务器负载	网络影响	响应时间	服务器负载	网络影响	项目开发	项目维护
B	最长	最重	最大	无	无	无	复杂	较难
C	最短	轻	最小	较好	较大	较大	容易	最难
D	短	稍重	小	最好	较小	较小	容易	容易

依据实际项目中的使用经验和本文测试比较部分所作的分析,本文建议在数据总量较小,开发维护要求较低的情形下应用方案 B;在数据量大,后期维护要求较低,页面加载性能要求较高的情况下应用方案 C;在数据量大,维护要求高,页面加载性能要求不高,加载后响应性能要求较高的情况下应用方案 D。

3　结束语

结合实际应用分析了 B/S架构下 WEB应用中常用到的操纵关联实体集问题,并结合当前常用技术手段,总结出了几种实现方案。本文阐述了实现操纵关联实体集的原理,并从多个方面进行测试分析,最终得出各个方案的性能对比,最后给出方案的建议使用情景,对相似问题的研究具有一定的参考价值。

参　考　文　献

[1] Jeese James, Garrett Ajax A New Approach to Web Applications[EB/OL](2005 - 08). http://www.adaptivepath.com/publications/essays/archives/000385.php.

[2] Daniel A Menasce. Loading Testing, Benchmarking, And Application Performance Management For The Web[C]. CMG Conference, Reno, 2002: 271-282.

[3] 梁晟,李明树,梁金能,等. Web应用程序运行响应时间的实验研究[J]. 计算机研究与发展, 2003, 40(7).

[4] 王星,潘郁. 基于 Ajax技术的 Web模型在网站开发中的应用研究[J]. 微计算机信息, 2006, 09X: 206-207, 241.

[5] Ryan Asleson, Nathaniel T. Schutta. Ajax基础教程[M]. 北京:人民邮电出版社, 2006.

[6] Dave Crane, Eric Pascarello with Darren James. Ajax in Action[M]. Manning Publications Co, 2006.

[7] 飞思科技产品研发中心. JSP应用开发详解[M]. 北京:电子工业出版社, 2005.

[8] 孙卫琴. 精通 Struts:基于 MVC的 JavaWeb设计与开发[M]. 北京:电子工业出版社, 2005.

[9] The Jakarta Apache Project[EB/OL]. http://jakarta.apache.org/jmeter/index.html

[10] IBM公司, IBM WebSphere应用服务器 5.0系统管理与配置[M]. 北京:清华大学出版社, 2004.

[11] 段念. 软件性能测试过程详解与案例剖析[M]. 北京:清华大学出版社, 2006.