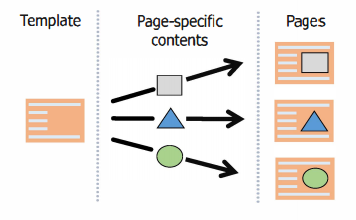
Mvc架构驱动的重构来实现客户端网页构成

摘要：本文呈现了一种将java web应用的页面构成从服务器重定位到浏览器的方式。目的是减少过多重复的操作和传输页面共享的代码或数据。减少量是动过执行重构算法产生的，从mvc架构的视角来看，有效地保证了源和目标应用的一致性，因为当前的问题是要求目标应用要保存源应用上的可观察的行为。研究案例表明，我们的重构工具能有效的支持重构的程序。

关键字：重构，mvc，单页应用，基于模板的网页应用，软件引擎，软件引擎。

1. **介绍**

在一个网页应用中，网页页面通常分享着一些内容，比如页头，页脚和菜单。在一块地方用一个模板去维护这些公共内容分隔页面上展示的特定内容。一个基于模板的web应用(TWA)使用这样一种模板去动态地生成页面，并将页面上特定内容与模板进行绑定。举个例子，在图1中，这三个页面通过使用一个模板和每个页面特定内容生成。

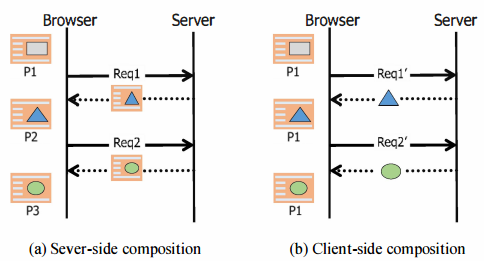


图一.基于模板的web应用

在java web应用中，<jsp:include>标准动作常用于将页面特定内容填充进模板的占位符中

当用户发出一个请求之后，一个网页浏览器下载和展现一个新的页面，TWAs 主要同过多页面应用模型与用户进行交互。如图2a展示的一样，一件最重要的记住的事情就是TWAs是对于每个请求，服务器都会结合对应的模板和页面特定内容并发送。因此对于么每个请求，模板数据被重复进行传输并用整个页面刷新进行渲染。

对于这种重复问题，其中一种解决方案是将页面构成由服务器转交给浏览器。无论用户何时点击一个超链接或是提交一个表单，网页浏览器接收到当前页面和新页面之前的不同处，这时局部地更新当前页面不同之处而不同过整个页面的刷新。如图2b所示。每个请求，请求1和请求2造成一次关于有联系的差异部分的部分页面刷新。页面1是物理上的一个单独页面但不能有不同的统一资源定位地址（URLs）,取决于他的状态（即包含的内容）。这种网页应用称它为单页应用（SPAs）。



图二.web页面组件

MVC架构被普遍用于网页应用开发中。因此，当重构或重建多页面应用成一个单页面应用时，它能很自然地考虑到三个组件和他们之间的交互作为一个基本的研究和改动的单元。然而，这些架构组件还没有在大多数之前的工程中充分地描述关于转化的问题。

这个转化是主要要求在它执行前前提条件能够满足。特别是，关于重复代码和数据的减少量有时候会导致在源和目标应用上的不一致：目标应用上的可观察行为能与他的源应用行为有所不同。然而，大多数之前的关于转换问题的调查确实没有在具体的当转换手段无法应用时的环境进行过。

因此，从MVC模型的组件架构方面来说，非常有必要去确定先前条件和转换的步骤。这篇论文提出一种方式，从MVC架构的层面来说，不通过改变源应用可观察的行为并从一个完整页面的请求能重构成一个部分页面的请求。也推出一种重构算法去描述重构的步骤以及重构的前提条件。我们展示一个通过重构算法生效的重构程序的总体图片。此外，这篇论文还描述了当实现这种重构算法时我们遇到的最重要的问题。此外，我们展示了案例研究的结果，当我们的工具被使用时，重组所需的工作量减少了，web应用程序的性能得到了改善。

1. **问题和背景**
   1. **问题**

本文讨论的重构问题可以被认为是低级的重构，因为用户界面(UI)结构，页面导航模型（即两个页面之间的链接），数据模型和源应用程序的功能都没有改变。目标应用程序具有与源应用程序相同的外观和感觉，但它提高了响应时间的性能和在网络上传输的数据量，并在服务器和浏览器之间提供了异步通信功能。此外，模板和页面特定内容之间的耦合通过重构减少，因此维护（例如，一个额外的新的特定页面内容的请求）变得更加容易。选择低层次结构调整的根本原因是，将一个输入应用程序的多页面架构无缝地重构到单一的页面架构中，而不会让用户感到困惑。

我们假定输入的TWAs是使用采用MVC模式的Java Server Pages (JSP) model 2架构开发的。这个假设是合理的，因为JSP模型2架构在JSP的早期规范中被提及并且在业界得到了认可。这里有许多关于解决将一个多页面应用重构或重建成单页面应用问题的方式。但是，从MVC这个web应用程序开发的流行架构模式的角度来看，很难找到解决这个问题的研究。本文的早期版本明确地考虑过修改模型、视图、控制器及他们之间的交互。但是，这个早期研究有一些限制。重构的先决条件没有描述。算法不能自动进行重构，所以不太容易实现这个方法。在重构过程中不能显示实验结果。

在本文中，我们改进了之前的研究，并提出了一种重构算法，该算法考虑了从MVC模式的角度对输入和输出应用程序的一致性检查。此外，我们还从过程质量的角度提出了案例研究的结果。

* 1. **背景**

本段简要地描述了我们之前的工作，帮助人们更清楚地理解本论文的概念。Java web应用程序主要由JSP、servlet和JavaBean对象组成，它们分别从MVC架构的角度对应到视图、控制器和模型。web模板已被用于为用户提供一致的视图，并减少web应用程序中的代码。模板创建并管理共享组件，比如菜单。当用户请求一个页面时，请求被发送到一个servlet，该servlet执行其业务逻辑。执行的结果这时包括被发送到浏览器上的模板生成的页面。在请求处理过程中，通过servlet和视图来访问JavaBeans对象来操作业务数据。图3a展示了动态的架构处理用户请求。

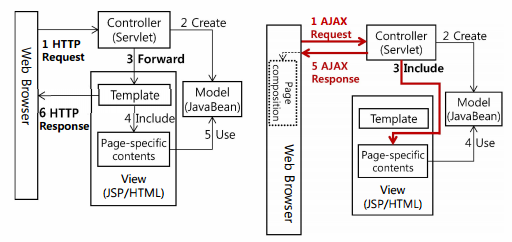
该架构存在一个问题，即模板在每个用户请求上都会加载并渲染整个页面。因此，我们之前的工作提出了一种将TWA重组为客户端web页面组合来解决问题的方法。主要的想法是在第一次访问应用程序时下载模板，然后，尽可能地绕过模板下载进行后续访问。此外，用户请求从正常的超文本传输协议(HTTP)请求转换为异步JavaScript和XML (AJAX)请求，以便在浏览器中启用部分更新。在JavaScript代码的帮助下，完整的页面完全由浏览器组成，它更新当前文档对象模型(DOM)树，同时用下载的部分替换旧的页面特定部分。如图3b展示的动态架构处理用户请求。因此，输出应用程序是一个SPA。

我们之前的工作还支持向后导航和可书签化，来支持与经典web应用程序相同的方式访问重组的应用程序。，默认情况下，当用户在SPA中单击浏览器的后退按钮时，浏览器不会回到之前的SPA状态，而是回到SPA之前访问的页面(见图4a)。为了解决这个问题，我们之前的工作应用了超文本标记语言5 (HTML5)历史应用程序编程接口，它在用户请求的浏览器历史堆栈中存储当前状态，并在使用back按钮时检索状态(见图4b)。默认的，一个SPA有不同的状态，但是只有一个URL。

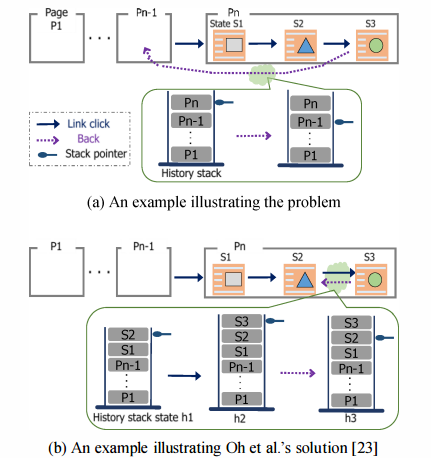
1. **将基于模板的应用程序重组为单页应用程序。**
   1. **重构程序**

在本小节中，我们展示了修改TWAs到SPAs的重组过程的总体图(见图5)。首先，一位工程师从用户的角度使用并理解一个源应用程序。然后，她收集可以进行重构的正常HTTP请求的链接和表单。对于每个请求，工程师都可以提取一个动态架构，它可以被表示为一个与图3a相似的协作图，来显示web组件之间的协作。该架构可以在重组应用程序的验证步骤中有效地使用。可以使用Java web开发技术中提供的过滤器和包装器组件来实现架构的提取。由于空间有限，不详细描述提取步骤。现在，工程师理解了源应用程序的功能和体系结构，于是，执行重构算法将输入的应用程序重构成为一个SPA。在重组完成后，对输出应用程序进行验证。当发现包括遗漏和不正确的转换的问题时，我们进行手动转换。

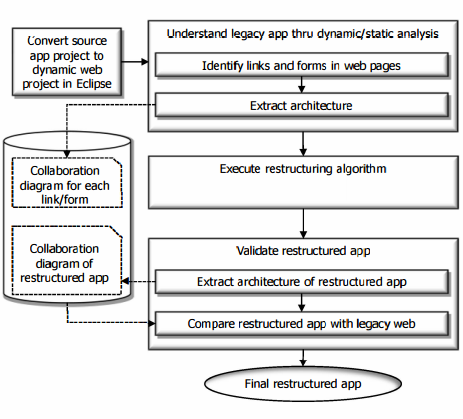
请注意，本文主要讨论架构转换及其支持工具。因此，我们开发了一个Eclipse插件来支持“执行重组算法”步骤，如图5所示。因此，我们开发了一个Eclipse插件来支持“执行重组算法”的步骤如图5所示，在后面的章节中将详细描述这个步骤。



图三.对客户端页面构成的架构转换



图四.后退导航问题和解决方案



图五.重构过程总览

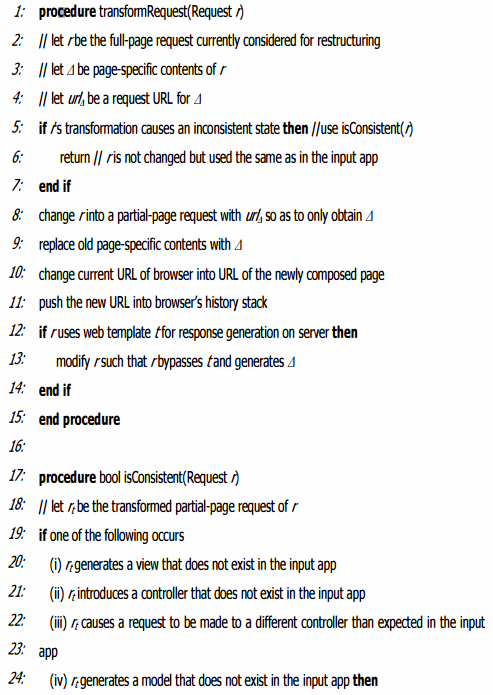
* 1. **重构算法**

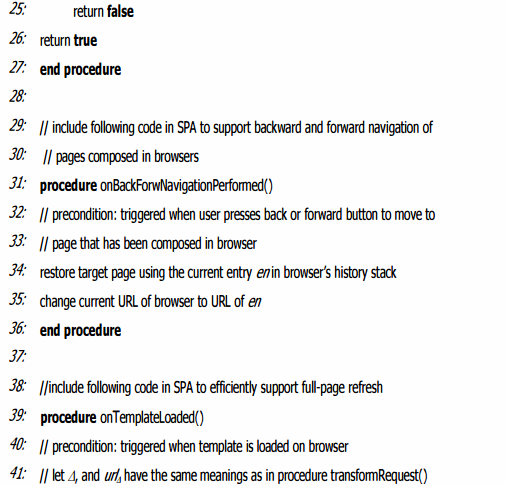
图6显示了我们如何从抽象的角度将TWA转换为SPA。首先，我们对TWA进行解析来手机整个页面的请求，可能的话我们对这个请求用transformRequest()（参见1-15行）进行分析和修改成部分页面的请求。如果转换后的请求将生成的SPA带到不一致的状态，则原始请求不会被转换(参见第5-7行)。从MVC架构的角度来看，这种不一致的状态可以在三种情况下发生。（参见17-27行）。

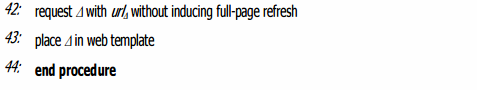
第一个例子是一个转换后的请求生成一个在TWA中不存在的视图(见第20行)。例如，假设我们有一个完整页面的请求r，它会改变当前的模板以及页面的特定内容。如果将r转换为一个部分页面请求，可以下载新的特定页面的内容，那么转换后的请求会产生一个不一致的视图。

模板不是静态页面，而是动态页面(见图7)。根据系统/用户状态，从模板类实例化不同的模板对象。例如，在图中，模板对象在用户登录后被替换，因为它们（登陆前后）的菜单是不同的。当用户请求需要一个新的模板实例时，一个SPA也会下载新的实例。一种可能的设计是下载新旧模板对象之间的相差内容。但是，这些差异可以分散到模板对象中，因此实现起来并不容易。因此，我们决定不转换原始的完整页面请求。

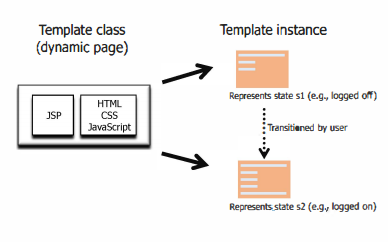
第二个例子是对原始请求转换导致了错误的情况。1）将一个无效的控制器引入到SPA中。2）在TWA中，向不同的控制器发出请求(参见第21-23行)。







图六.重构算法



图七.模板及其实例化

首先，web浏览器以HTML、级联样式表(CSS)和JavaScript执行客户机代码。其中，JavaScript代码在客户端扮演控制器的角色。关于客户端代码的一个重要问题是，当页面特定内容被替换时，旧的特定页面内容的javascript代码不像HTML和css，它所执行的效果仍然存在（而html和css很可能被完全替换）。另一方面，完整页面请求将卸载旧页面的所有客户端代码。因此，需要进行分析，以确定旧的JavaScript代码是否让SPA与TWA运行起来不同。

改变TWA可观察行为的具体情形如下: 例如，假设为了见少页面加载的时间，在TWA中某个页面的特定页面部分有JavaScript代码，可以在用户滚动到页面底部时下载图像。(例如，通过在window.onScroll()事件处理器中执行AJAX请求)。如果用户移动到一个新的物理页面，这样的事件处理代码因为页面卸载将自动删除。但是，在重组的SPA中，代码删除不会自动发生，因为进入一个新状态不会导致整个页面刷新。这意味着在SPA(第21行)中存在一个无效的控制器。

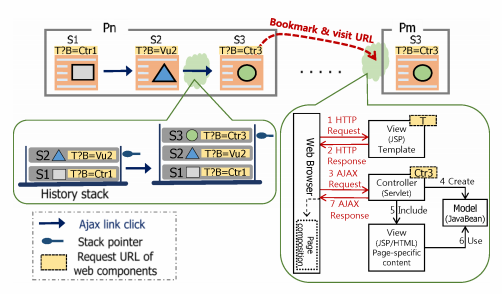
其次，SPA应该保留其输入TWA的页面导航模型(包括向后/向前导航和刷新)。例如，假设TWA的页面Pt被重构为SPA页面Ps，那么Ps和 Pt是相同的。如果这两个页面被刷新，那么新的页面也应该的是相同。为了实现这一点，每个SPA状态都有一个独特的URL，就像TWA的每一页一样。这样一个状态的URL需要包含用于标识状态的信息(例如，请求获取特定页面内容的请求URL)。因此，当刷新页面时，可以与模板一起获得特定页面的内容(见图8)。当使用AJAX将整个页面请求重组为一个部分页面请求时，需要注意一点: 当初始请求被重定向(通过从服务器到web浏览器的HTTP重定向请求)到另一个组件c，重定向URL(即:c的请求url)默认情况下不能通过SPA获得。另一方面，重定向的URL在带有TWA的web浏览器的地址栏中显示。因此，当我们尝试刷新一个包含HTTP重定向请求的响应的SPA页面时（这个请求为特定页面内容），SPA的运行方式与它的输入TWA不同。这意味着SPA将请求发送给不适当的控制器(第22-23行)。

再次，转换后的请求可以生成在TWA中不存在的模型(见第24行)。举个例子，相同地更新服务器状态请求的转换能允许生成生成的SPA有无效的数据。一个特定的场景如下：假设，在TWA中，我们有一个页面，它的支付表单不能同时提交相同的数据。但是，如果其转换的表单被允许提交两次相同的数据，则转换会产生双倍的支付，这是二次提交问题的实例[1,8,19]。下一节将展示一个更具体的示例。

另一方面，如果原始请求的转换造成了一致的状态，那么将执行转换(参见第8-15行)，这样输出应用程序的工作方式如下: 通过转换后的请求，浏览器将接收到一个新的特定于页面的部分(第8行)，并将旧的部分替换为新的部分(第9行)，而不需要刷新整个页面。为了将这样的页面差异从服务器传输到浏览器，请求URL(第4和第8行中的URL’)标识了其特定页面部分。另外，一个唯一的URL被分配到整个页面，这样用户就可以添加书签并重新访问页面(参见第10和11行，以及图8)。修改控制器使其跳过与模板的交互，并连接到生成特定页面的内容的视图(第12-14行)。

onBackForwNavigationPerformed()方法被新增到SPA中，以便用户能够前后浏览网页(参见第29-36行)。如果用户按下前进或后退按钮使目标页面在浏览器中被重组，我们则会获得之前的状态（通过使用历史状态栈）并插入当前页面去重建目标页面。然后，浏览器的URL更改为目标页面的URL。另一方面，当一个页面在浏览器中被重新组成时，新页面的信息会被压入历史状态栈中(参见行11)。

为了有效地支持王征页面刷新，onTemplateLoaded()方法也被加入到SPA中（参见行38-44）。当用户输入SPAs或刷新当前页面时，页面将被完全加载。这样的完整页面刷新需要下载模板和特定页面部分。我们使用混合了 (如图8所示)服务器端和客户端之间的web组件，以便为页面启用并行下载。因此首先先请求模板，接着立刻请求特定页面部分，而不需要完整页面的刷新，这种更新类似于转换请求部分页面的更新步骤。



图八.可书签化和页面刷新的解决方案

假设页面Pn在在状态S3中，一个关于特定页面部分的请求URL为Ctr3。此外，让

一个模板的请求URL作为T。我们能指定”T?B=Ctr3”为一个包括状态3完整页面。

1. **实现。**

这个部分描述了我们所开发的重构TWAs个SPAs的Eclipse插件。实现这个具备重构算法的工具过程中所遇到的问题也将会在以下进行详细的描述。

* 1. **重构工具**

一个重组工具在Eclipse的动态web项目中以两个步骤运行。步骤一，根据我们重组的前提，确定要转换的代码段。该工具使用Jericho HTML解析器来搜索JSP和HTML文件，以获取这些代码段。选择解析器是因为JSP和HTML标记能够被其识别和修改。在步骤2中，通过使用重组算法来改变被识别的重组代码片段。

* 1. **实现重构算法**

本小节描述在实现重组算法时遇到的重要问题。其中一个问题是确定转换后的请求是否将生成的SPA带到给一个有效的状态。目前，重组工具考虑了以下因素。

首先，该工具考虑了HTTP请求方法，其中GET和POST是在确定的过程中被指定分析的，因为这两种方法是web应用程序最常用的方法。HTTP GET请求通常在我们的工具中进行重构，因为GET方法被设计用于从服务器检索资源，而HTTP GET请求的转换不会产生不良影响。另一方面，POST请求可以进行转换，也可以不进行转换，这取决于应用程序该如何处理请求。一般来说，HTTP POST请求的目的是更新服务器状态，如果请求未被仔细处理，则可能导致二次提交问题。因此，在文献中提出了两种解决方法：post-redirect- get (PRG)模式[1,8,20]和同步令牌机制[1,20]。当TWA的原始请求使用带有同步令牌机制的POST方法时，请求的转换不会导致双重提交问题。但是，使用PRG模式支持的POST请求的转换会导致问题，因为无法获得重定向地址。因此，转换POST请求的决定不像GET请求，需要更多的静态和动态分析。这个决定可能还需要工程师的介入。

其次，工具考虑发出请求的上下文。假设我们有一个POST请求，如果请求被转换并在浏览器的主窗口中发出，则会导致双提交问题。但是，当请求在窗口w(浏览器的主窗口之外)发出时，并且w在处理请求后消失，请求将不会由相同的数据生成两次。因此，请求不会造成这个问题。可以看到，根据请求的上下文，请求可以被转换或不转换。

再次，转换和测试请求可以被认为是决定请求转换的标准。这种方法是有效的，但不是很划算。重构/重组的基本前提是稳定测试[24]。因此，在适当地准备测试用例时，这种方法是实用和有效的。我们遇到的另一个实现问题是应用程序的检测和预处理，它破坏了HTTP请求方法的原则。例如，单击<a>元素通常是作为HTTP GET请求发出,从服务器检索信息。但是，我们可以找到一些应用程序，其中包含一些包含<a>元素的web页面，其中单击会导致服务器状态更改。因此，在执行重组算法之前，先对超链接进行静态和动态分析，并适当地改变方法类型。

另一个实现问题是标识或构建特定页面内容的url。将特定页面的部分的请求URL发送到模板作为请求参数时，可以容易地使用该参数的值来构建这样的URL。另一方面，应用程序可以使用逻辑标识符而不是请求url来指定特定页面的部分。然后可以将标识符映射到服务器端上的请求url。在这种情况下，应该在转换算法执行之前/期间收集这样的映射表。

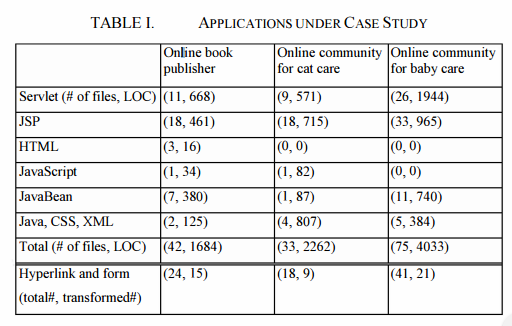
1. **案例研究。**

在本节中，案例研究的结果被描述为过程质量和产品质量。

* 1. **过程质量**

在本小节中，我们将从过程质量的角度展示案例研究的结果。我们做了三个TWAs的实验：一个在线图书出版商，一个关于分享养猫的信息的网络社区，以及另一个关于提供抚养婴儿帮助信息网络社区。对于每一个TWA，进行了有和没有工具支持的重组。表1中显示了应用程序的大小，分别从文件的数量、代码行数(LOC)、包括表单(total#)的超链接的数量，以及要转换的超链接的数量(转换#)的方面展示。

表1.研究案例下的应用



本文使用四个著名的度量标准(精确度、召回率、准确性和工作量)来评价实验的结果。精确度的度量衡量了由我们的工具(或者实验者)转换了多少超链接，这些超链接实际上有多少被转换。召回率显示我们的工具正确地转换了需要转换的超链接的数量。准确性衡量我们的工具如何精确地改变超链接的转换。工作量衡量完成重组所需的时间。在三个案例研究中，具有相近能力和web应用程序开发经验的实验者，在实验前已经了解了我们的重组方法。手动转换应用程序的实验人员使用了我们工具中提供的相同的JavaScript库。

图9展示了结果。在所有的案例研究中，当我们的工具被使用时，重组的工作量(工时)减少了。所有的实验者都准确地改变了这三个应用程序，所以精确性、召回率和准确性的值都是100%。此外，对在线图书出版商和社区的猫护理的工具支持的重组比社区的婴儿护理更有效。其中一个原因是:婴儿护理社区在web组件之间有着复杂的交互，其中一些组件扮演了控制器和视图的角色。因此，这种转换需要对社区对婴儿护理的这个应用有深刻的理解。另一个原因是，使用我们工具的实验者试图重构社区的动态架构，以消除不必要的交互，而不像没有工具支持的实验者。

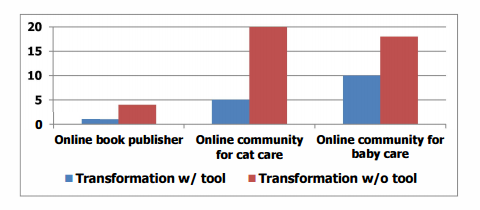


图9.工作量方面的案例研究结果

* 1. **产品质量**

为了从产品质量的角度验证我们的方法，我们测量了性能。主要是在移动设备和台式电脑上的网络应用程序的响应时间和传输数据量。

我们选择作为案例研究的是在线图书出版商[10] 的TWA。这个应用程序可以被看作是某些在线商店应用程序的简化版本[3,11]，通常被用作web应用程序的标准程序。选择TWA的另一个原因是，应用程序具有不同的web页面，包括HTTP请求的接收者、特定页面内容的类型等，如表2所示。例如，Find页面允许用户通过关键字搜索图书。当用户在表单中的文本字段中输入关键字并提交表单时，GET请求被发送到控制器，该控制器将查询发送到数据库以提取相关信息。

重组后的TWA被命名为Tapp，Tapp是通过我们的方法将Sapp转换而来的。我们的实验是在两台台式电脑上进行的。一个服务器和一个客户端，使用3.3 GHz Intel i5-2500处理器，4GB DDR3内存，以及Windows 7 32位操作系统。数据存储在Cubrid 2008 R4.1数据库中。Tomcat 7.0和Chrome 35.0.l916分别用作web容器和web浏览器。

模板下载和web页面组合的性能需要被测量，以显示本文提出的客户端页面组合的效率。浏览器中的数据缓存影响web应用程序的性能。因此，我们在两种情况下进行了实验:一个是当web页面第一次被访问时，缓存中没有相关数据。另一种是访问已经访问过的web页面，缓存里有相关数据。

图10显示了第一次访问web页面时的性能结果。通过访问主页面，我们可以看到模板下载的性能，这是用户访问的首页，只显示模板。Sapp比Tapp加载速度慢，因为Sapp不仅可以下载和处理模板，还可以处理JavaScript代码(参见图10b中的页面Main)来启用AJAX请求。但是，在将模板加载到浏览器后，当访问允许客户端页面组合的页面时（即Intra,BookList RevuFm 和 Find），Sapp加载比Tapp快。

RevuWr页面通过发出一个表单POST请求来写购买评论。Sapp刷新整个页面，因为如果页面的请求被转换，那么转换后的请求会导致双重提交问题。Sapp的完整页面生成有两步，如图8所示：:模板下载和客户端页面组成。当与主页比较时，Sapp和Tapp的性能差异较小。这是因为将web资源的下载(例如，具有总大小为240KB图像、JavaScript代码和CSS代码)的下载与服务器事务的执行同时发生。

图11显示访问已访问的web页面时的结果。需要注意的是构成页面的静态资源已经被缓存到浏览器中了，因此，与第一次实验相比，所传输的数据量减少了，如图10b和图11b所示。如此一来，响应时间同时也减少了，我们能在图10a和图11a中看到。在访问Intra页面时，由于特定页面的部分是静态HTML代码(见表2)，而在浏览器中已经缓存了，因此，Sapp的速度要更快。

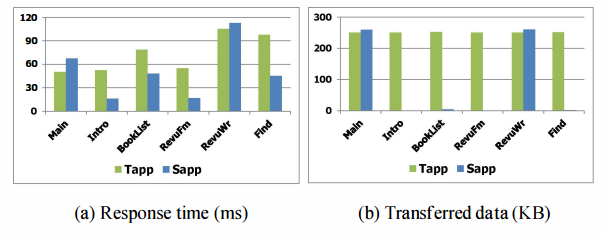
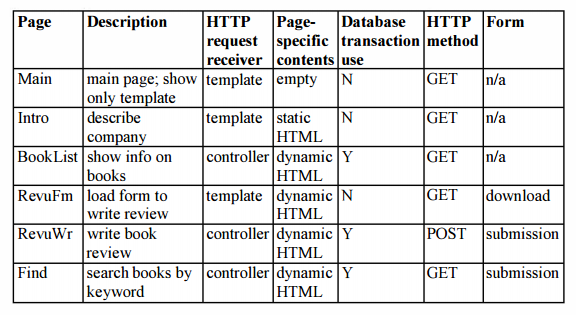


图10.Tapp和Sapp在第一次访问每个页面时的web页面组成的性能

通过这两个实验，我们可以看到，在访问导致部分页面刷新的页面时，在不考虑浏览器中的数据缓存的情况下，源应用程序可以提高目标应用程序的性能。

从智能手机这样的移动设备访问网站变得越来越普遍。用户希望在移动设备和桌面电脑上有效地浏览网页。因此，我们还通过1.2GHz处理器、1gb RAM和Android 4.1.2果冻豆操作系统测试了Sapp和Tapp在智能手机上的性能，以确定在移动设备上的性能改进。图12显示了在使用3G网络时访问两个应用程序的响应时间。可以看出，利用我们的重组方法可以取得很大的改善。当中一个原因是全页面刷新需要更多的CPU时间来显示页面。传输数据量与桌面环境中的实验数据相同。因此，在移动设备上使用Sapp的用户可以根据传输数据的数量来降低花费。

表2. 在线图书出版商的网页差异



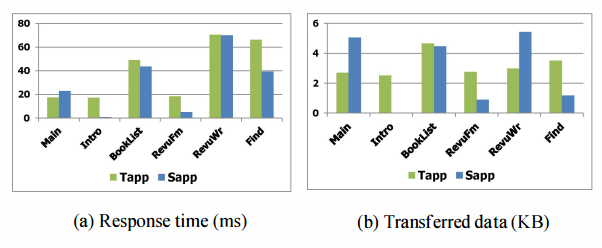


图11.Tapp和Sapp在访问已访问过每个页面时的web页面组成的性能

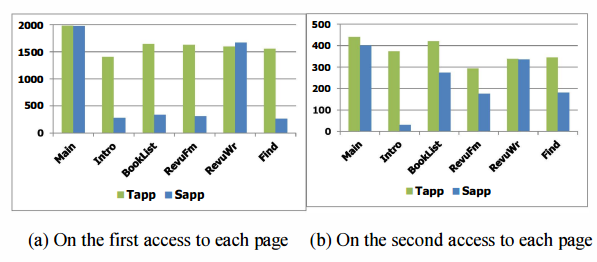


图12.Tapp和Sapp在手机端访问的响应时间(ms)