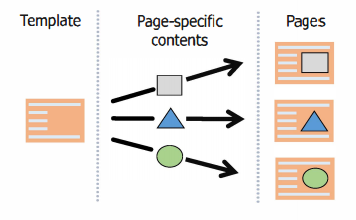
Mvc架构驱动的重构来实现客户端网页构成

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。

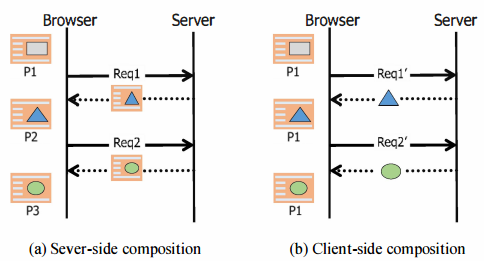


图一.基于模板的web应用

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。

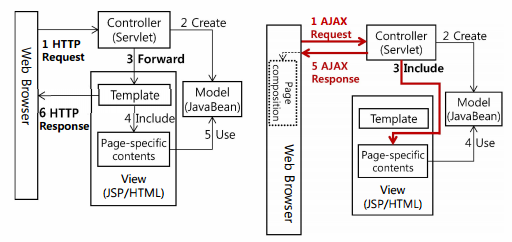
该架构存在一个问题，即模板在每个用户请求上都会加载并渲染整个页面。因此，我们之前的工作提出了一种将TWA重组为客户端web页面组合来解决问题的方法。主要的想法是在第一次访问应用程序时下载模板，然后，尽可能地绕过模板下载进行后续访问。此外，用户请求从正常的超文本传输协议(HTTP)请求转换为异步JavaScript和XML (AJAX)请求，以便在浏览器中启用部分更新。在JavaScript代码的帮助下，完整的页面完全由浏览器组成，它更新当前文档对象模型(DOM)树，同时用下载的部分替换旧的页面特定部分。如图3b展示的动态架构处理用户请求。因此，输出应用程序是一个SPA。

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。

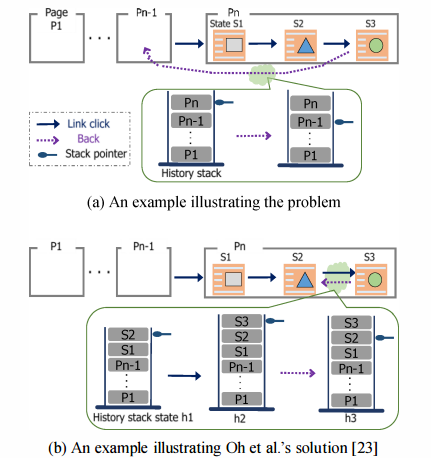
1. **将基于模板的应用程序重组为单页应用程序。**
   1. **重构程序**

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。

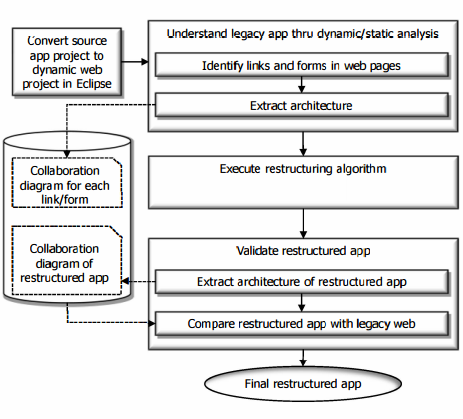
请注意，本文主要讨论架构转换及其支持工具。因此，我们开发了一个Eclipse插件来支持“执行重组算法”步骤，如图5所示。因此，我们开发了一个Eclipse插件来支持“执行重组算法”的步骤如图5所示，在后面的章节中将详细描述这个步骤。



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

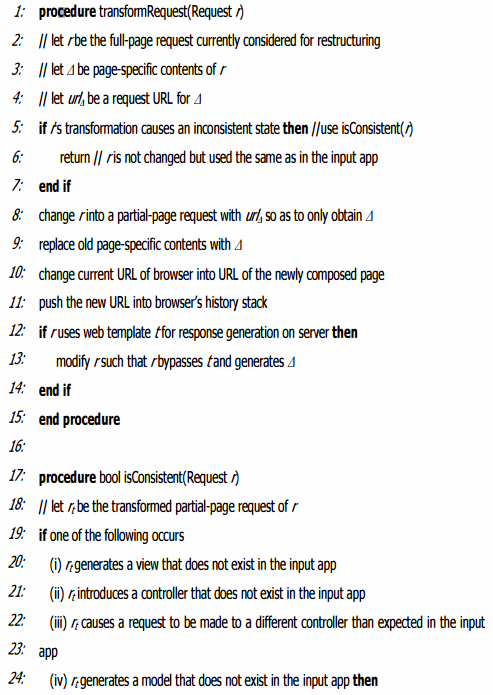
* 1. **重构算法**

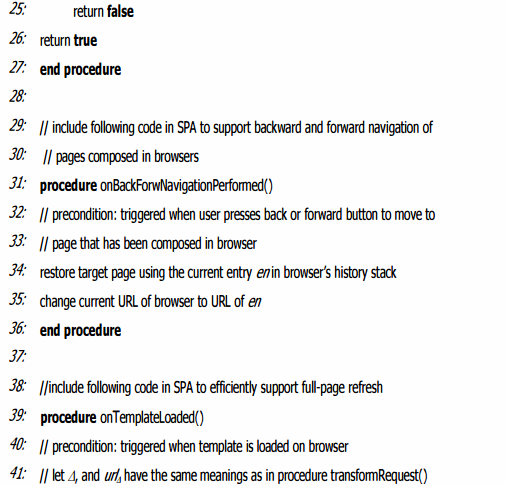
图6显示了我们如何从抽象的角度将TWA转换为SPA。首先，我们对TWA进行解析来手机整个页面的请求，可能的话我们对这个请求用transformRequest()（参见1-15行）进行分析和修改成部分页面的请求。如果转换后的请求将生成的SPA带到不一致的状态，则原始请求不会被转换(参见第5-7行)。从MVC架构的角度来看，这种不一致的状态可以在三种情况下发生。（参见17-27行）。

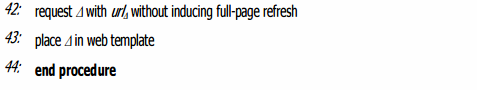
第一个例子是一个转换后的请求生成一个在TWA中不存在的视图(见第20行)。例如，假设我们有一个完整页面的请求r，它会改变当前的模板以及页面的特定内容。如果将r转换为一个部分页面请求，可以下载新的特定页面的内容，那么转换后的请求会产生一个不一致的视图。

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

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，这是双提交问题的实例[1,8,19]。下一节将展示一个更具体的示例。