使用 UML 为 Web 应用程序构架建模

Jim Conallen

Rational Software 白皮书

TP 157, 6/99

这份材料发表在 1999 年 10 月 （第 42 卷，第 10 期）的 Communications of the ACM 上。

Rational®
the software development company

目录

摘要

Web 应用程序正变得越来越复杂，越来越重要。为了帮助管理这种复杂性，需要为 Web 应用程序建模。 UML 是软件密集型系统的标准建模语言。在尝试用 UML 为 Web 应用程序建模时，很明显它的一些组件不能与标准的 UML 建模元素一一对应。为了让整个系统（Web 组件，以及传统的中间层组件）使用同一种建模表示法，必须扩展 UML。本文介绍 UML 的一种扩展方式（使用正式的扩展机制）。进行扩展是为了让 Web 特有的组件能与系统模型的其余部分集成，向 Web 应用程序的设计员、实施员以及构架设计师展示适当的抽象和明细级别。

概述

近年来，IT 词汇表中出现了一条新的术语，它就是“Web 应用程序”。参与业务软件系统的所有人似乎都有构建 Web 应用程序的计划，而在与业务不相关的软件方面也有很多人对此感兴趣。对于很早前就采用这种构架的许多人来说， Web 应用程序这个词象系统本身一样，已经从成功的小型 Web 站点插件发展成了强壮的 n 层应用程序。 Web 应用程序可以同时为分布在世界各地的、成千上万的用户提供服务，这种情况早已司空见惯。构建 Web 应用程序是一件严肃的事情。在实际应用中，Web 应用程序这个词对不同的人而言含义略有不同。一些人认为凡是用到 Java 的都是 Web 应用程序，而另一些人则认为凡是使用 Web 服务器的都是 Web 应用程序。多数人的意见介于这两者之间。站在本文的角度，我们将 Web 应用程序大体定义为 Web 系统 （Web 服务器、网络、 HTTP、浏览器），在这个系统中，用户的输入（导航和数据输入）会影响到业务状态。该定义试图将 Web 应用程序确立为一个具有业务状态的软件系统，并且它的“前端”基本上是通过 Web 系统传递的。

Web 应用程序的总体构架是一个客户机服务器系统，但二者有几点显著的区别。 Web 应用程序最重要 的优点之一在于它的部署。部署 Web 应用程序通常指的是建立网络的服务器端组件。客户端不需要特别的软件或配置。两者的另一个重大差异在于客户机和服务器通信的本质。 Web 应用程序的基本通信协议是 HTTP，这是一个无连接协议，它不是为最大的通信吞吐量设计的，而是为强壮性和容错而设计的。在 Web 应用程序中，客户机和服务器的通信通常围绕 Web 页导航进行，而不是在服务器端和客户端对象之间直接通信。在一定的抽象程度上，Web 应用程序中所有的信息传递都可描述为 Web 页实体的请求和接收。通常所说的 Web 应用程序构架与动态 Web 站点的构架并无太大区别。

Web 应用程序与 Web 站点，甚至是与动态 Web 站点的区别都要涉及到使用。 Web 应用程序实现的是业务逻辑，它的使用改变了业务的状态（其状态为系统捕获）。这是很重要的，因为它确定了建模工作的重点。 Web 应用程序执行业务逻辑，因此大多数重要的系统模型都侧重于业务逻辑和业务状态，而不是表示细节。表示很重要（否则系统将毫无用处），不过应尽量将业务和表示所关注的问题区分开。如果表示问题是重要的，甚至是复杂的，那么也需要对它们建模，但不必将它们作为业务逻辑模型的构成部分。此外，用于表示的资源更注重外观设计，而与实施业务规则关系不大。

关系管理方法 (RMM) 是与 Web 系统开发有关的一种方法/表示法。RMM 是一种用于设计、构建和维护 Intranet 及 Internet Web 系统的方法。它的根本目标是降低动态数据库驱动的 Web 站点的维护成本。它提倡系统进行形象化表示，以便展开设计上的讨论。它是一个迭代式过程，包括 Web 页可视元素的分解，及这些元素与数据库实体的关联关系。 RMM 是一种用于动态 Web 站点创建和维护的“完整详尽”的方案。

不过，在构建 Web 应用程序方面 RMM 就显得无能为力了。 Web 应用程序以业务逻辑为中心，它包括了许多实施业务逻辑的技术机制，而这些内容在 RMM 表示法中并未充分说明。客户端脚本编写、Applet 和 ActiveX 控件等技术为促进系统业务规则的执行发挥了重大作用。另外，Web 应用程序还可用作分布式对象系统的交付机制。 Applet 和 ActiveX 控件可以包含那些独立于 Web 服务器，通过 RMI 或者 DCOM 与服务器端组件异步交互的组件。复杂应用程序还可利用多个浏览器实例和客户机上的框架，建立并维护自己的通信机制。

既然所有这些机制都对系统的业务逻辑有促进作用，因此同样也需要为它们建模。而且，由于它们只表示部分业务逻辑，它们需要与其余的系统模型集成。在很多情况下，大部分业务逻辑在 Web 服务器后、服务器端的某一层执行。建模语言和表示法的选择通常要按照这一端的应用程序的需要来决定。随着 UML 作为一种正式的对象建模语言被 OMG 所接受，越来越多的系统开始用 UML 表示。许多人选择 UML 作为软件密集型系统的建模语言。于是 Web 应用程序建模的主要问题变成了 :“如何在应用程序的其余部分表示在特定 Web 组件中执行的业务逻辑？” 答案取决于我们用 UML 在那些特定的 Web 元素和技术中表示系统业务逻辑执行的能力。

本文旨在简要介绍 Web 应用程序建模存在的问题和可能的解决方案。其中着重讲述在构架上对 Web 应用程序有重要意义的组件，以及如何使用 UML 对它们进行建模。本文假定读者熟悉 UML、面向对象技术的原理和 Web 应用程序开发。文中描述的工作基于一些无偏向性的假设。

- Web 应用程序是日益复杂的软件密集型系统，它们在关键的任务中发挥着越来越重大的作用。
- 管理软件系统复杂性的一种方法是对它们进行抽象和建模。
- 软件系统一般有多个模型，每一个模型代表一个不同的观点、不同的抽象和详细级别。
- 哪个抽象和详细级别是合适的，这取决于开发流程中的工件和角色活动。
- 软件密集型系统的标准建模语言是统一建模语言 (UML)。

本文介绍的概念和思想在今年年底即将由 Addison Wesley Longman 出版的“Object Technology Series” 中的“Building Web Applications with UML”一书中有更为充分的论述。

建模

通过简化一些细节，模型可以帮助我们理解系统。如何选择建模对象对理解问题和提供解决方案有重大影响。 Web 应用程序与其他软件密集型系统一样，通常由用例模型、实施模型、部署模型、安全模型等一组模型来表示。 Web 系统还另有一个专用模型，即站点图。站点图是对贯穿整个系统的 Web 页和导航路线的抽象。

目前采用的大部分建模方法都适用于各种 Web 应用程序模型的开发，因而无需对它们进行进一步的讨论。不过，有一个非常重要的模型，分析/设计模型 (ADM)，在尝试将 Web 页、与其相关的可执行代码和其他元素纳入模型时，确实出现了一些困难。

在决定如何建模时，确定正确的抽象和详细级别对于是否能让模型用户享受到建模带来的好处至关重要。一般而言，最好对系统的工件建模。工件就是那些为生成最终产品而构建和操纵的真实生活中的实体。对 Web 服务器的内部组件建模，或者对 Web 浏览器的具体构成部分建模，这对于 Web 应用程序的设计员和构架设计师并没有什么帮助。页、页之间的链接、构成页的所有动态内容，以及在客户机上出现过的页的动态内容，对这些建模才是重要的，而且是非常重要的。设计员设计的、实施员实施的正是这些工件。页、超链接、客户机和服务器上的动态内容正是需要建模的对象。

下一步是将这些工件映射到建模元素。例如，超链接自然映射到模型中的关联关系元素。超链接代表了从某一页到另一页的导航路径。将这种思路进行扩展，页就可以映射到模型逻辑视图中的类。如果 Web 页是模型的一个类，那么页的脚本自然就映射为这个类的操作。脚本中的所有页范围(内的)变量映射为类属性。 Web 页可能包括一套在服务器上执行的脚本（准备页的动态内容），以及另一套完全不同的只在客户机上执行的脚本（如 JavaScript），考虑到这一点时，一个问题就出现了。在这种情况下，当我们查看模型中的 Web 页类时，我们搞不清楚在准备页的过程中哪些操作、属性甚至关系在服务器上处于活动状态，而当用户与页交互时哪些在客户机上处于活动状态。另外，Web 应用程序中传递的 Web 页最好是作为系统组件建模。简单地将 Web 页映射到 UML 类不会对我们理解系统有所帮助。

UML 的创始人意识到总会存在这样的情况：初始的 UML 不足以获取一个特定领域或构架的相关语义。为了解决这个问题，他们确定了一种正式扩展机制，允许实践者扩展 UML 的语义。该机制允许定义可应用到模型元素的*构造型*、*标注值*和*约束*。*构造型*是一种修饰，允许我们为建模元素定义新的语义。*标注值*是可以与建模元素相关联的键值对，允许我们在建模元素上“标注”任何值。约束是定义模型外形的规则。它们可表示为任何形式的文本，或者用更正式的对象约束语言(OCL)表示。

本文讨论的工作引入了为 Web 应用程序所作的一种 UML 扩展。这种扩展从整体上看超出了本文的范围，然而这里还是讨论了其中大部分的概念和解释。关于建模最后还要注意一点，一定要明确区分业务逻辑和表示逻辑。对于典型的业务应用程序而言，只有业务逻辑才应成为 ADM 的一部分。表示细节，如动画按钮、浮动帮助和其他 UI 增强方式通常不属于 ADM。如果为应用程序单独构建一个 UI 模型，则可在其中纳入表示细节。 ADM 需要始终将重点放在业务问题和解决方案的表达上。在今天这个 Web 设计师的时代，对 Web 页外观的设计和实现最好由专业人员（技术绘图师）取代传统的开发人员来完成。

Web 应用程序构架

Web 应用程序的基本构架包括浏览器、一个网络和一个 Web 服务器。浏览器向服务器请求“Web 页”。每一页都是内容和以 HTML 表达的格式指令的组合。一些页包括客户端脚本，它们由浏览器解释。这些脚本为显示的页定义了其他动态行为，而且它们经常与浏览器、页内容和页中包含的其他控件（Applet、ActiveX 控件和插件）交互。用户查看页中的内容，并与其交互。有时，用户在页的字段元素中输入信息，并提交给服务器处理。用户还可以通过超链接导航到系统的其他页，与系统进行交互。无论是哪种情况，用户都在向系统提供输入，这样就可能改变系统的“业务状态”。

从客户端来看，Web 页总是一个采用 HTML 格式的文档。然而在服务器端，“Web 页”可表现为多种形式。在最早的 Web 应用程序中，动态 Web 页用公共网关接口 (CGI) 构建。CGI 定义了一个供脚本和已编译模块使用的接口，它们通过该接口访问与页请求一起传递的信息。在基于 CGI 的系统中，通常在 Web 服务器上配置一个特殊的目录，以便针对页请求来执行脚本。在请求 CGI 脚本时，服务器会用解译器（通常是一个 PERL shell）处理或执行相应的文件，以流的形式将输出返回给发出请求的客户机，而不仅仅是返回文件内容（就像处理 HTML 格式的文件时一样）。处理的最终结果是 HTML 格式的流，它会被返回给发出请求的客户机。业务逻辑在处理文件的同时在系统中执行。在这段时间内，它能够与服务器端资源（如数据库和中间层组件）交互。

目前的 Web 服务器已经在这个基本设计上有所改进。它们现在已非常注重安全性，而且包含了一些特性：如在服务器端的客户机状态管理、事务处理集成、远程管理、资源共享等，这里只列举了其中的几种。总的来说，最新一代 Web 服务器处理的都是那些对构架设计师来说非常重要的问题，它们关系到任务至上、可缩放和强壮的应用程序。

根据 CGI 脚本的作用，可以将现今的 Web 服务器分为三大类：脚本页、编译页，以及两者的混合体。在第一类中，客户机浏览器能请求的每一个 Web 页在 Web 服务器的文件系统中都用一个脚本文件来表示。这个文件一般是 HTML 和其他一些脚本语言的混合。对页发出请求后，Web 服务器委派一个可识别该页的引擎对其进行处理，最终结果以格式为 HTML 的流的形式返回给发出请求的客户机。Microsoft 的 Active Server Pages、 Java Server Pages 和 Cold Fusion 都属于这一类。

在第二类中，Web 服务器加载并执行一个二进制组件。这个组件和脚本页一样，有权访问与页请求一起发送的所有信息（表单字段值和参数）。经过编译的代码利用请求的详细信息，并通常要访问服务器端的资源，以生成 HTML 流并返回给客户机。编译页包含的功能往往比脚本页大，尽管并未明确这一规律。通过向编译页请求传递不同的参数，可获得不同的功能。任何一个编译组件实际都可以包含整个目录脚本页的所有功能。Microsoft 的 ISAPI 和 Netscape 的 NSAPI 就是表示这种构架类型的技术。

第三类指的是那些一旦发出请求即进行编译的脚本页，以后的所有请求都使用编译过的页。只有最初页的内容改变了，该页才会进行另一次编译。这类页介于灵活的脚本页和高效的编译页之间。

Web 页建模

Web 页，不管是脚本页还是编译页，都一对一地映射到 UML 中的组件。组件是系统的“物理”可更换部件。模型的实施视图（组件视图）描述了系统的组件及它们之间的关系。在 Web 应用程序中，这个视图描述了系统的所有 Web 页及它们彼此之间的关系（如超链接）。在一定程度上，Web 系统的组件图就相当于站点图。由于组件仅仅代表了接口的物理打包方式，它们并不适用于对页内部的协作关系建模。这一抽象级别仍需要成为模型的组成部分，而且对设计员和实施员而言极其重要。为引入正题，我们可以说每个 Web 页在模型的设计视图（逻辑视图）中都是一个 UML 类，它与其他页的关系（关联关系）即代表了超链接。但如果考虑到任何 Web 页都可能既代表一组只存在于服务器上的功能和协作，同时又代表只存在于客户机上的一组完全不同的功能和协作，那么这种抽象就不成立了。举例来说，使用动态HTML（客户端脚本）作为部分输出的所有服务器 Web 脚本页都是这样的页。对这个问题的直接反应可能就是为类中的每个属性或操作设计原型，指明它是在服务器端还是在客户端有效。至此，运用模型倒让问题变得复杂了，而我们的初衷是要简化问题。

一个更好的解决方案是“分别考虑”。从逻辑上讲，服务器端的 Web 页行为与客户端是完全不同的。在服务器上执行时，页有权访问服务器端资源（中间层组件、数据库、文件系统等），即与这些资源具有某些关系。同一页（或者是该页的 HTML 流输出）在客户端有完全不同的行为和关系集。在客户端，脚本页与浏览器本身（通过文档对象模型，即 DOM），以及该页指定的所有 Java Applet、 ActiveX 控件或插件相关。对于认真的设计员而言，页还可能与客户机上在另一个 HTML 框架或浏览器实例出现的其他“活动”页相关。

通过分别考虑，我们就可以用一个类为 Web 页的服务器端建模，用另一个类为客户端建模。我们采用 UML 扩展机制为两者分别定义构造型和图标：«server page» 和 «client page»，以此来区分两者。 UML 中的构造型允许我们为建模元素定义新的语义。已指定构造型的类在 UML 图中可用定制图标表示，或者仅仅用 («») 之间的构造型名称说明。图标对概述图很有用，在概述图中，最好使用简单的标记对显示出的类属性和操作进行标注。

对于 Web 页，构造型指出了类是客户机或服务器上 Web 页逻辑行为的抽象。两种抽象通过两者之间的定向关系相互关联关系。这种关联关系的构造型为：«build»，因为可以说服务器页构建了客户机页（图 1）。每个动态 Web 页（即页内容在运行时才能决定的页）都用一个服务器页构建。每个客户机页至多只能用一个服务器页构建，而一个服务器页可以构建多个客户机页。Web 页之间通过超链接建立公共关系。 Web 应用程序中的超链接代表系统的一条导航路径。这种关系在模型中用一个构造型为 «link» 的关联关系表示。关联关系总是从客户机页出发，指向另一个客户机页或服务器页。

超链接作为 Web 页请求在系统中实施， Web 页作为实施视图中的组件来建模。指向客户机页的链接关联关系大体等同于指向构建该客户机页的服务器页的链接关联关系。这是因为链接实际是一个页请求，不是上述两种类抽象。由于 Web 页组件同时实现两种页抽象，因此指向由该页组件实现的任何类的链接都是等同的。

标注值用于定义随链接请求一起传递的参数。«link» 关联关系标注值“Parameters” 是一个参数名（和可选值）的列表，处理请求的服务器页要用到它。在图 2 中，SearchResults 页包含数目可变的指向 GetProduct 服务器页的超链接 (0..*), 每一个链接都有一个不同的 productId 参数值。 GetProduct 页用于构建 productId 参数 所指定产品的 ProductDetail 页。

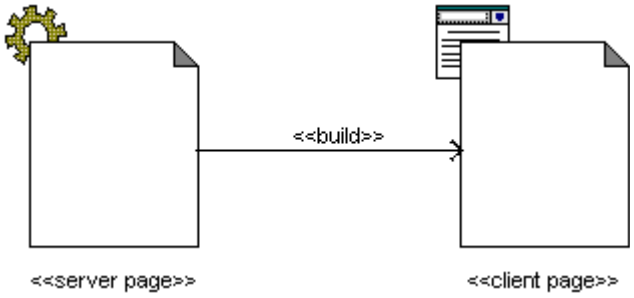

图 1. 服务器页构建客户机页。

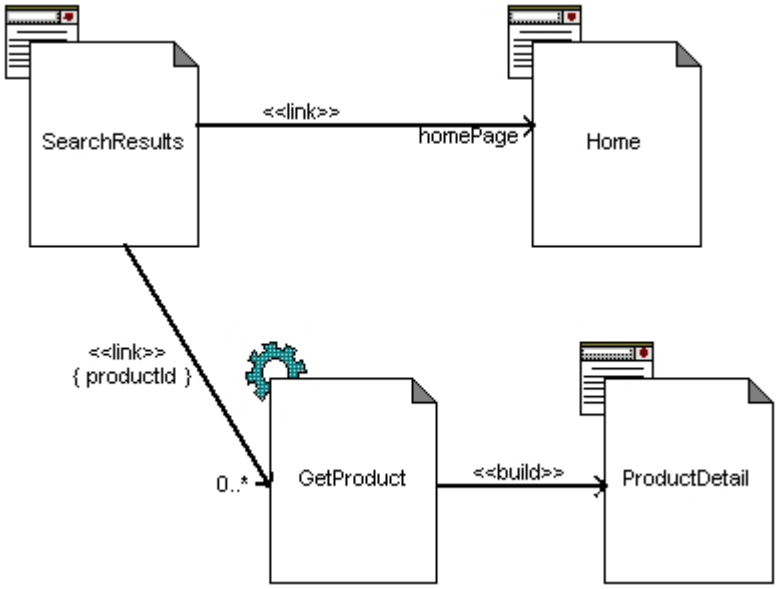

图 2. 使用超链接参数。

使用这些构造型简化了对页脚本和关系的建模。«server page» 类的操作变为页服务器端脚本的函数，它的属性变为页范围变量（页函数可对其进行全局访问）。«client page» 类的操作和属性也同样变为在客户机上可见的函数和变量。将服务器端页和客户端页作为不同的类考虑，其最大好处在于明确页与系统的其他类之间的关系。客户机页根据它们与客户端资源的关系进行建模。客户端资源有：DOM、 Java Applet、ActiveX 控件和插件（图 3）。服务器页根据它们与服务器端资源的关系进行建模。服务器端资源有：中间层组件、数据库存取组件、服务器操作系统
等(图 4)。

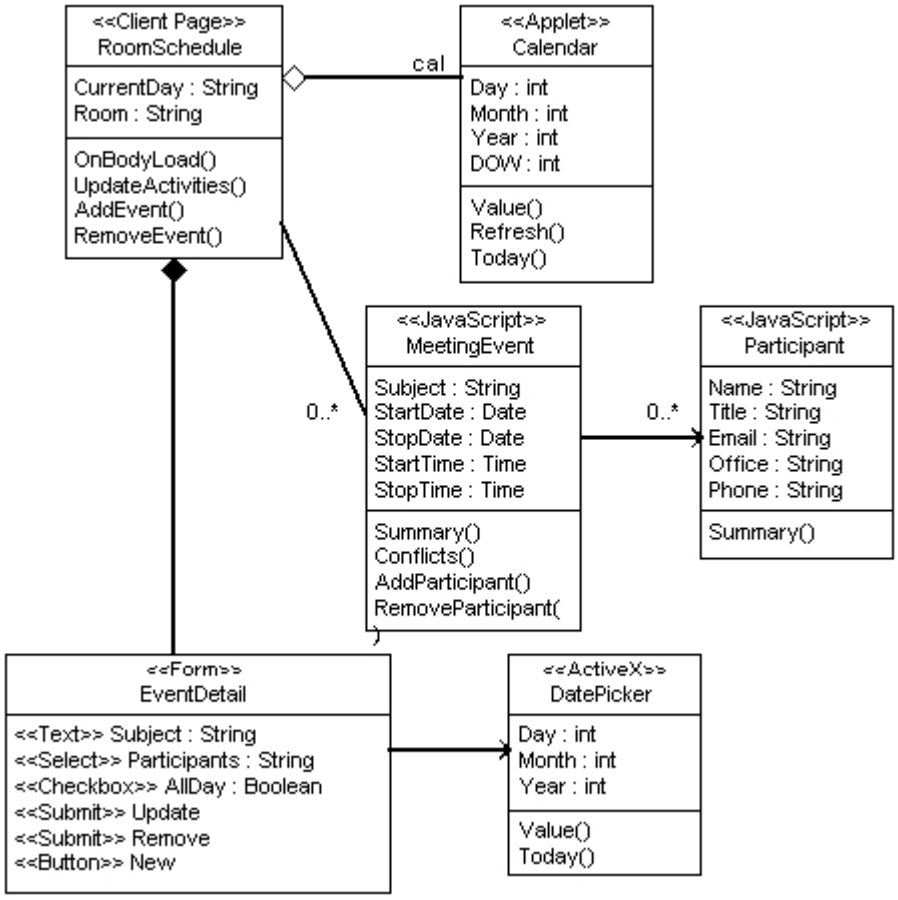
<<Client Page>>
RoomSchedule
CurrentDay : String
Room : String
OnBodyLoad()
UpdateActivities()
AddEvent()
RemoveEvent()
cal
<<Applet>>
Calendar
Day : int
Month : int
Year : int
DOW : int
Value()
Refresh()
Today()
0..*
<<JavaScript>>
MeetingEvent
Subject : String
StartDate : Date
StopDate : Date
StartTime : Time
StopTime : Time
Summary()
Conflicts()
AddParticipant()
RemoveParticipant(
0..*
<<JavaScript>>
Participant
Name : String
Title : String
Email : String
Office : String
Phone : String
Summary()
<<Form>>
EventDetail
<<Text>> Subject : String
<<Select>> Participants : String
<<Checkbox>> AllDay : Boolean
<<Submit>> Update
<<Submit>> Remove
<<Button>> New
<<ActiveX>>
DatePicker
Day : int
Month : int
Year : int
Value()
Today()

图 3. 客户机协作

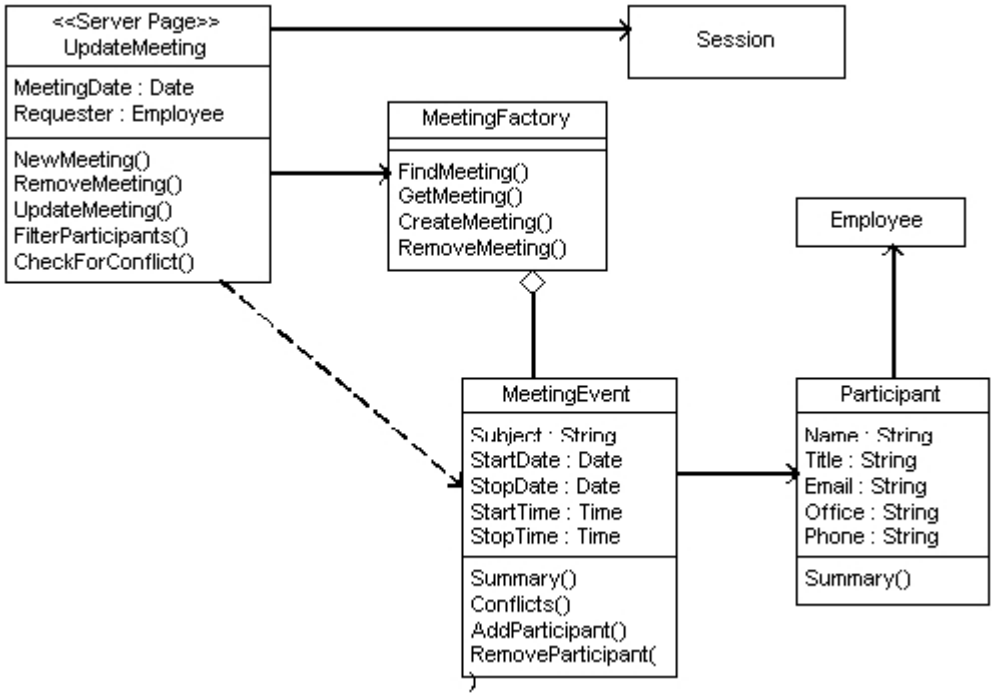

图 4. 服务器协作

用类构造型对 Web 页的逻辑行为建模的最大优势在于：页与服务器端组件的协作可以基本上用其他任意服务器端协作所使用的方式来表示。«server page» 仅仅是参与系统业务逻辑的另一个类。上升到概念的层次来讲，服务器页一般扮演控制者的角色，协调必要的业务对象活动，以满足浏览器页请求所提出的业务目标。

客户端的协作更复杂一些。部分原因可归咎于可用技术的多样性。即使是最简单的客户机页，至少也是一个既包含内容又包含表示信息的 HTML 文档。浏览器利用页的格式指令提供 HTML 页，有时还带有单独的样式表。在逻辑模型中，这一关系可用客户机页对构造型为 «Style Sheet» 的类的依赖关系来表示。样式表基本上是一个表示问题，通常不包含在 ADM 内。

表单

Web 页的基本数据输入机制是表单。表单在 HTML 文档中用 <form> 标记来定义。每个表单都会指明它自身要提交到哪一页。表单包括许多输入元素，它们全用 HTML 标记表示。最常用的标记是 <input>、<select> 和 <textarea>。输入标记多种多样，它可以是一个文本字段、复选框、单选按钮、按钮、图像、隐藏字段，还有其他一些不太常见的类型。对表单建模要使用另一个类构造型：«Form»。«Form» 没有操作，这是因为可能在 <form> 标记中定义的所有操作实际上都为客户机页所有。表单的输入元素都是 «Form» 类的建有构造型的属性。«Form» 可以与作为输入控件的 Applet 或者 ActiveX 控件有关系。表单还与服务器页有关系，服务器页即是处理表单提交内容的页。这种关系的构造型为 «submit»。由于表单完全包含在 HTML 文档内，所以它们在 UML 图中用一种强聚合关系形式表示。图 5 是一个简单的购物车页，它定义了一个表单，显示了与要处理表单的服务器页的提交关系。

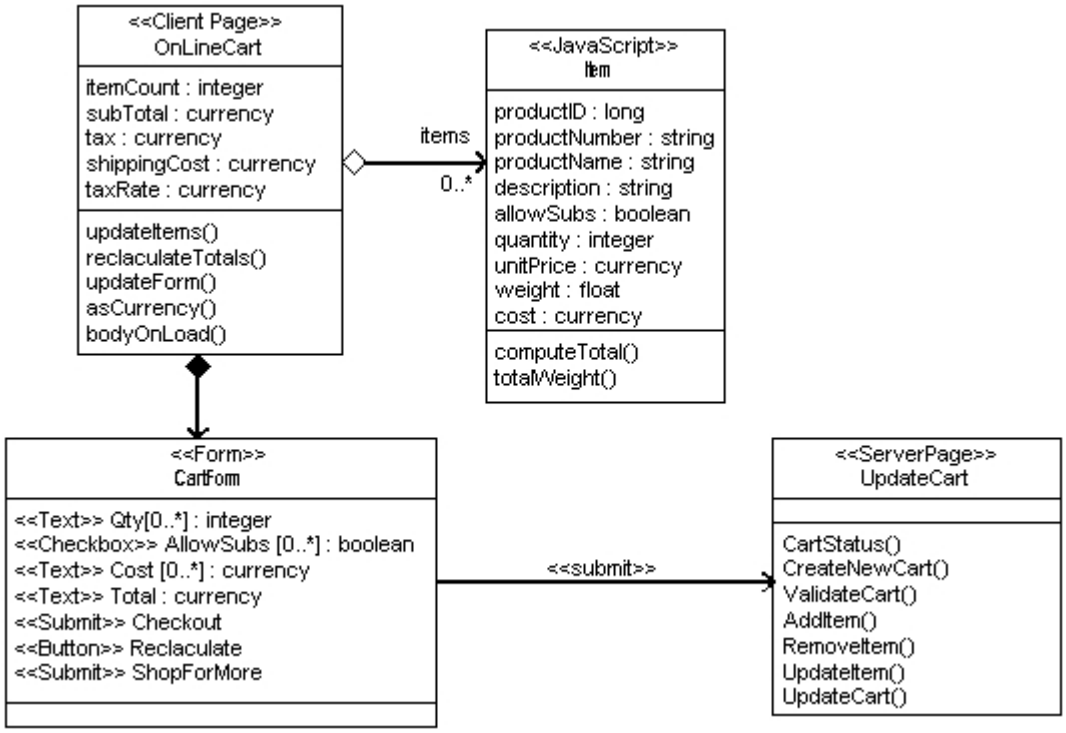

图 5. 表单提交给服务器页。

在图 5 中，以 «JavaScript» 为构造型的类是一个代表购物车中的商品项的对象。对于可有多种输入可能的字段，它们的表单属性说明中使用了数组语法。在这个例子中，这意味着购物车可以有零到多个项，每个项都有一个 Qty、AllowSubs、Cost 和 Total <input> 元素。

由于客户机页内的所有活动都用 JavaScript 执行，而 JavaScript 是一种无类型的语言，所以为这些属性指定的数据类型只是为了便于实施员辨认。在 JavaScript 中执行或作为 HTML 输入标记执行时，该类型将被忽略。这对函数参数也成立，函数参数是模型的一部分，尽管本图并未明确显示。

框架

HTML 框架从刚被引入 Web 站点和 Web 应用程序时就引发了极大的争论，且意见呈两极分化。框架允许在某一时刻有多个页激活，并对用户可见。目前最常用的浏览器还具备一组最新的特性，允许用户机器上同时打开多个浏览器实例。使用动态 HTML 脚本，这些页中的组件可以彼此交互。客户端复杂交互的潜力非常大，使得对此建模的需要变得更大。

是否在应用程序中采用框架或多个浏览器实例，这要由构架设计师决定。如果决定采用，客户端行为的模型就需要在 ADM 中表示出来，原因如上所述。

要对框架的使用进行建模，我们又定义了两种类构造型：«frameset» 和关联关系构造型 «targeted link»。框架集类代表一个容器对象，直接映射到 HTML<frameset> 标记。它包含客户机页和目标。目标类是一个被其他客户机页引用的指定框架或浏览器实例。

目标链接关联关系是指向另一个页的超链接，但它要在特定目标中才能提供。在图 6 的示例中，浏览器显示了一个使用两个框架的常用大纲视图。一个框架根据目标命名为 Content，另一个框架只包含一个客户机页。这一客户机页框架包含书的目录 (TOC)。该页的超链接是有目标的，因此超链接指向的内容在 Content 框架中显示。得到的效果就是页左侧的一个静态目录，以及页右侧的书中每一章的内容。

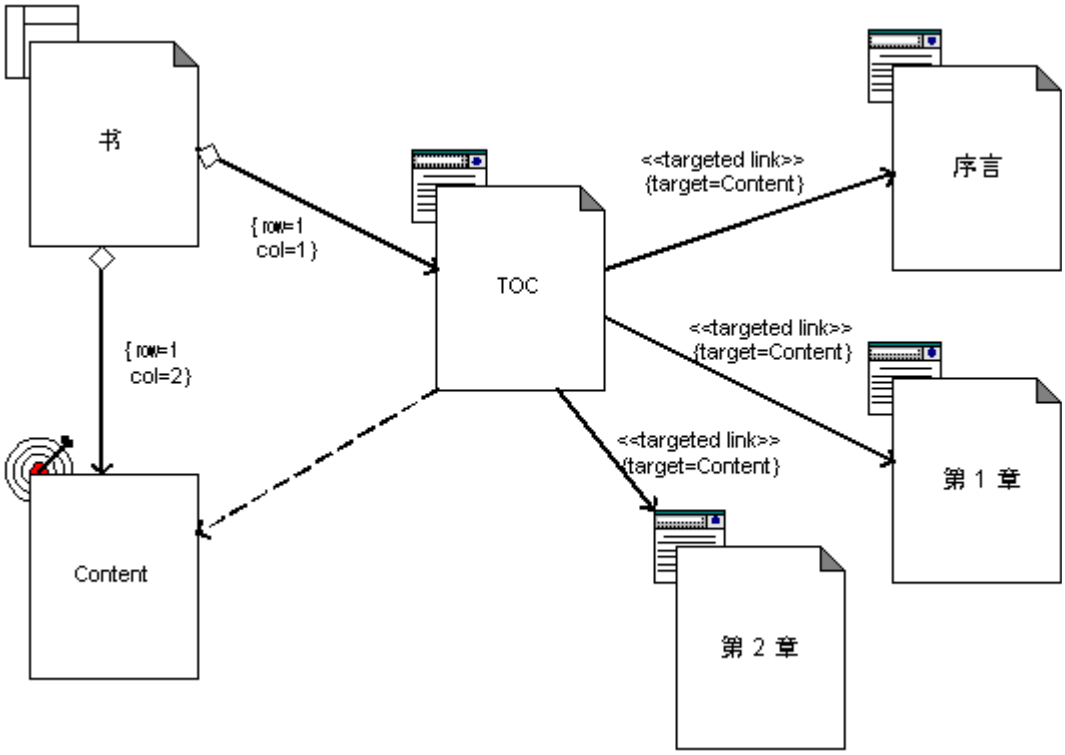

图 6. 框架示例

实际表示的很多细节在框架集和关联关系中用标注值注明。框架集与目标或客户机页之间的聚合关系上有两个标注值，它们指定目标或页所属的框架集行列。目标链接关联关系上的标注值“Target”用于确定显示页的 «target»。

若目标不与框架集聚合关系，则意味着用一个独立的浏览器实例来显示页。务必要记住这种表示方法用于表示客户机的单个实例。假定多个独立的目标全部在同一台机器上运行，这里的图用来表示一个客户机实例的客户端行为。其他所有部署配置都需要在模型中详细记录，以便于理解。

结论

对于利用 UML 对 Web 应用程序特定元素进行建模所存在的问题和已提出的解决方案，本文讨论的观念和概念只进行了初步介绍。本文的目的在于提出一种一致且全面的方法，将 Web 特定元素建模与应用程序的其余部分集成起来，以便提供对 Web 应用程序设计员、实施员和构架设计师合适的详细和抽象级别。为 Web 应用程序开发的第一个 UML 正式扩展版本已接近尾声。该扩展版本将为构架设计师和设计员提供一种常用方法，借助 UML 表示 Web 应用程序的整体设计。

关于该扩展版本的最新信息可通过 Internet 在 Rational Software Web 站点的 Rose 和 UML 相关部分中找到。

Rational
the software development company

两家总部：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
电话：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
电话：(781) 676-2400

免费电话：(800) 728-1212
电子邮件：info@rational.com
Web: www.rational.com
全球网址：www.rational.com/worldwide