使用 UML 对 Web 应用程序体系结构建模

Jim Conallen

Rational Software 白皮书

TP 157, 6/99
此资料的某个版本出现在 1999 年 10 月的 Communications of the ACM 期刊（卷 42、编号 10）中。

Rational®
the software development company

目录

摘要

Web 应用程序正变得越来越复杂和关键。要帮助管理此复杂性，需要对其进行建模。统一建模语言（UML）是用于对软件主导系统建模的标准语言。当尝试使用 UML 对 Web 应用程序建模时，以下情况是显而易见的：某些组件不能很好地符合标准 UML 建模元素。为了对整个系统（Web 组件和传统中间级组件）保持一种建模表示法，必须扩展 UML。本白皮书简介了使用正式的扩展机制对 UML 进行的扩展。设计此扩展是为了特定于 Web 的组件可与系统模型的其余部分进行集成，并为 Web 应用程序的设计人员、实施者和设计人员提供合适的抽象和详细程度。

概述

在过去的几年中，一个新的术语进入了 IT 词汇表：Web 应用程序。似乎参与业务软件系统的每个人都有建立 Web 应用程序的计划，对许多相关的非业务软件工作也很感兴趣。对于此体系结构的许多早期采纳者，术语 Web 应用程序就如同系统本身，从小型的成功 Web 站点附加软件发展成为健壮的分成 n 级的应用程序。同时服务于数以万计的用户，分发在世界各个角落，对于 Web 应用程序来说是很正常的。设计 Web 应用程序是很重要的业务。

当进入测试时，术语 Web 应用程序对于不同的人，含义稍有不同。某些人认为 Web 应用程序是使用 Java 的程序；而其他人认为 Web 应用程序是使用 Web 服务器的程序。一般的共识是介乎两者之间的。出于本白皮书的目的，我们将 Web 应用程序宽松地定义为 Web 系统（Web 服务器、网络、HTTP 和浏览器），其中用户输入（导航和数据输入）影响业务状态。此定义试图确定 Web 应用程序是具有业务状态的软件系统，并且它的“前端”有很大一部分是通过 Web 系统传递的。

Web 应用程序的一般体系结构是客户端服务器系统，具有某些明显的特征。Web 应用程序最重要的优势之一是它的部署。部署 Web 应用程序通常是在网络上设置服务器端组件。在客户机这一方面不需要特殊的软件或配置。另一个重要的区别是客户机和服务器通信的性质。Web 应用程序的主要通信协议是 HTTP，这是一种无连接的协议，设计目的是健壮性和容错，而不是最大的通信吞吐量。在 Web 应用程序中，客户机和服务器之间的通信通常围绕着浏览 Web 页面进行，而服务器端和客户端对象之间不直接进行通信。就某个抽象级别而言，Web 应用程序中的所有消息传递可以描述为 Web 页面实体的请求和接收。一般地讲，Web 应用程序的体系结构与动态 Web 站点的体系结构的区别并不大。

Web 应用程序和 Web 站点（甚至动态 Web 站点）之间的区别涉及它的用途。Web 应用程序实现业务逻辑，并且它的使用改变了业务状态（如系统所捕获的）。这是很重要的，因为它定义了建模工作的重点。Web 应用程序执行业务逻辑，因此最重要的系统模型侧重于业务逻辑和业务状态，而不是演示细节。演示是很重要（否则系统将对任何人都没用），但是您应该努力将业务问题和演示问题清晰地分开。如果演示问题很重要或甚至很复杂，那么也应对它们建模，但不必作为业务逻辑模型的不可或缺的一部分。另外，演示中使用的资源将倾向于更具艺术性，而较少关注业务规则的实施。

与 Web 系统开发关联的一个方法论 / 表示法是关系管理方法论（RMM）。RMM 是用于 intranet 和 Internet Web 系统的设计、构建和维护的方法论。它的主要目标是减少动态、数据库驱动的 Web 站点的维护成本。它提倡系统的可视演示，以促进设计讨论。它是一个迭代进程，包括分解 Web 页面中的可视元素以及这些元素与数据库实体的关联。RMM 是创建和维护动态 Web 站点的“完整详尽”的方法。

当建立 Web 应用程序时，RMM 就显示出不足了。Web 应用程序作为业务逻辑中心，包括用于实现业务逻辑的许多技术机制，而 RMM 表示法中没有充分涉及这些机制。诸如客户端脚本运行、applet 和 ActiveX 控件之类的技术通常为系统业务规则的执行做出重大贡献。另外，Web 应用程序可用作分发式对象系统的传递机制。Applet 和 ActiveX 控件可以包含通过 RMI 或 DCOM 独立于 Web 服务器与服务器端组件异步交互的组件。复杂的应用程序也可利用客户机上的多个浏览器实例和框架，它们建立和维护自己的通信机制。

由于所有这些机制都用于系统的业务逻辑，因此同样需要对它们建模。另外，由于这些机制只表示业务逻辑部分，因此需要与其余的系统模型集成。在许多情况下，大多数业务逻辑在服务器端某级的 Web 服务器后台批量执行。建模语言和表示法的选择通常由应用程序端的需要来决定。随着 OMG 接受 UML 成为正式的对象建模语言，越来越多的系统使用 UML 表示法表示。对于许多情况，UML 是用于对软件主导系统建模的语言选择。对 Web 应用程序建模的主要问题则变为：“如何表示在特定于 Web 的组件中与应用程序的其余部分一起执行的业务逻辑？” 答案在于通过 UML 利用特定于 Web 的元素和技术表达业务逻辑执行的能力。

本白皮书旨在简介对 Web 应用程序建模的问题和可能的解决方案。它侧重于特定于 Web 应用程序的在体系结构上重要的组件和如何使用 UML 对其建模。假设读者是熟悉 UML、面向对象的原理和 Web 应用程序开发的。在本白皮书中描述的工作基于某些相当无害的假定：

- Web 应用程序是软件主导系统，这种系统正日趋复杂并插入了更多关键的角色。
- 管理软件系统的复杂性的一种方法是对其进行抽象和建模。
- 一个软件系统通常具有多个模型，每个表示不同的视点、抽象和详细程度。
- 恰当的抽象和详细程度依赖于开发流程中的工件和活动。
- 软件主导系统的标准建模语言是统一建模语言（UML）。

本白皮书中陈述的概念和想法的更全面的论述将在即将发行的书：“Building Web Applications with UML”中阐述，该书预期在今年晚些时候由 Addison Wesley Longman 在 Object Technology Series 中出版。

建模

模型通过简化某些细节，帮助我们了解系统。建模内容的选择对于了解问题和解决方案的形式有很大影响。Web 应用程序同其它软件主导系统一样，通常用一组模型表示；用例模型、实施模型、部署模型和安全模型等。由 Web 系统专门使用的附加模型是站点图，它是贯穿系统的 Web 页面和导航路线的抽象。

目前所用的多数建模技术都十分适合 Web 应用程序的各种模型的开发，不需要更多讨论。但是，一个非常重要的模型 — 分析 / 设计模型（ADM），在尝试包含 Web 页面、相关可执行代码以及模型中的其它元素时提出了某些困难。

当确定如何建模时，确定正确的抽象和详细程度是至关重要的，以提供将有益于模型用户的内容。一般来讲，最好对系统工件建模 — 它们是被构造和操纵以产生最终产品的那些现实实体。对 Web 服务器内部细节或 Web 浏览器的详细内容建模将不会对 Web 应用程序的设计人员和设计人员有所帮助。对页面、页面彼此的链接、创建页面的所有动态内容和在客户机上出现过的页面动态内容建模是很重要的 — 非常重要。这是设计人员设计和实施者实施的那些工件。客户机和服务器上的页面、超链接和动态内容是需要建模的内容。

下一步是将这些工件映射到建模元素。例如，超链接将通常映射到模型中的关联元素。超链接表示从一个页面到另一个页面的导航路径。扩展此想法，页面可映射到模型逻辑视图中的类。如果 Web 页面是模型中的类，则页面脚本将通常映射到类的操作。脚本中的任何页面范围的变量将映射到类属性。当您考虑 Web 页面可以包含在服务器上执行的一组脚本（准备页面的动态内容）和只在客户机上执行的完全不同的一组脚本（即 JavaScript ）时，出现了一个问题。在此场景中，当查看模型中的 Web 页面类时，在服务器上哪些操作、属性、甚至关系是激活的（当准备页面时）以及当用户与客户机上的页面交互时上述内容哪些是激活的，对此存在混淆。另外，在 Web 应用程序中提供的 Web 页面确实最好作为系统组件进行建模。仅仅将 Web 页面映射到 UML 类不会帮助我们更好了解系统。

UML 的创建者意识到总是存在以下情况：完全现成的 UML 将不足以捕获特定领域或体系结构的相关语义。为了满足此目的，定义了正式的扩展机制以使业务人员可以扩展 UML 语义。该机制允许我们定义可应用于模型元素的*构造型*、*标注值*和*约束*。

*构造型*是一种装置，允许我们为建模元素定义新的语义含义。*标注值*是可与建模元素关联的关键值对，允许我们在建模元素上“标记”任何值。*约束*是定义模型的合适形式的规则。它们可用自由格式的文本或更正式的对象约束语言（OCL）来表示。

在本白皮书中讨论的工作简介了用于 Web 应用程序的 UML 扩展。此扩展的完整内容超出了此白皮书的范围，但这里讨论了大多数概念和说明。

关于建模的最后一点 — 在业务逻辑和表示逻辑之间需要作出非常明确的区分。对于典型的业务应用程序，只有业务逻辑应是 ADM 的一部分。演示细节，像动画按钮、悬浮帮助和其它 UI 增强功能通常不属于 ADM。如果为应用程序构造单独的 UI 模型，则可使用此类细节。ADM 需要始终侧重于业务问题和解决方案空间的表示方法。在这个 Web 艺术家的时代，最好由专家（技术美工师），而不是由传统的开发人员来设计和实现 Web 页面的外观。

Web 应用程序体系结构

Web 应用程序的基本体系结构包括浏览器、网络和 Web 服务器。浏览器从服务器请求“Web 页面”。每个页面融合了内容和格式编排的指示信息，用 HTML 格式表示。某些页面包括由浏览器解释的客户端脚本。这些脚本为显示页面定义附加动态行为，并通常与包含在页面中的浏览器、页面内容和附加控件（Applet、ActiveX 控件和插件）交互。用户查看页面中的内容并与之交互。有时，用户在页面中的字段元素中输入信息，并将信息提交给服务器进行处理。该用户也可通过超链接浏览系统中的不同页面来与系统交互。在任一情况下，该用户向系统提供可以改变系统“业务状态”的输入。

从客户机的角度看，Web 页面始终是 HTML 格式的文档。但是在服务器上，“Web 页面”可以许多不同的方式显示。在最早的 Web 应用程序中，动态 Web 页面是使用公共网关接口（CGI）建立的。CGI 为脚本和经编译的模块定义一个接口，以用于访问随页面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目录，以便能够执行脚本而响应页面请求。当请求 CGI 脚本时，服务器不是只返回文件内容（对任何 HTML 格式的文件都如此），而是使用适当的解释器（通常是 PERL shell）处理或执行文件并将输出流式传送回请求客户端。此处理的最终结果是发送回请求客户端的 HTML 格式的流。当处理该文件时，在系统中执行业务逻辑。在此期间，有可能与服务器端的资源（如数据库和中间级组件）交互。

目前的 Web 服务器已改进了此基本设计。现在它们更具安全意识并包含了诸如以下的功能：在服务器上管理客户机状态、事务处理集成、远程管理和资源池（仅仅列出了几个）。在总体上，最新一代的 Web 服务器正在解决对于关键、可扩展和健壮的应用程序的设计人员很重要的那些问题。

当查看 CGI 脚本的角色时，目前的 Web 服务器可分为三个主要类别：用脚本编写的页面、编译好的页面和两者的混合。第一个类别中，客户端浏览器可请求的每个 Web 页面在 Web 服务器的文件系统上表示为脚本文件。此文件通常是 HTML 和某些其它脚本语言的混合。当请求此页面时，Web 服务器将对此页面的处理委派给识别它的引擎，最终的结果是将 HTML 格式的流发送回请求客户端。示例有 Microsoft 的 Active Server Pages、Java Server Pages 和 Cold Fusion。

在第二个类别即编译好的页面中，Web 服务器装入和执行二进制组件。此组件与用脚本编写好的页面一样，有权访问与页面请求（格式字段值和参数）一起出现的所有信息。编译好的代码使用请求细节，并通常访问服务器端的资源，以产生返回给客户端的 HTML 流。虽然不是规则，但编译好的页面倾向于比用脚本编写的页面包含更多的功能。通过将参数传入编译好的页面请求，可以获得不同的功能。任一编译好的组件实际可包含整个目录的用脚本编写好的页面的所有功能。表示此类型体系结构的技术是 Microsoft 的 ISAPI 和 Netscape 的 NSAPI。

第三种类别表示一旦请求便进行编译的用脚本编写的页面，并且此编译好的版本以后将由所有后续请求使用。仅当原始页面的内容变更时，页面进行另一次编译。此类别兼顾了用脚本编写好的页面的灵活性和编译好的页面的高效性。

对 Web 页面建模

Web 页面（编写好的或编译好的）与 UML 中的组件一一映射。组件是系统的“实际”和可代替部分。模型的实施视图（组件视图）描述系统的组件和它们的关系。在 Web 应用程序中，此视图描述系统的所有 Web 页面和它们彼此的关系（即超链接）。在同一级别上，Web 系统的组件图像一个站点图。

由于组件只表示接口的实际封装，因此它们不适合对页面内部的协作建模。此抽象级别（对设计人员和实施者非常重要）仍需作为模型的一部分。作为开始，可以说每个 Web 页面是模型的设计视图（逻辑视图）中的 UML 类，而它与其它页面的关系（关联）表示超链接。但是，请考虑以下情况，出现此情况时此抽象被破坏：任何给定的 Web 页面可以潜在表示只存在于服务器上的一组功能和协作，而完全不同的一组功能和协作只存在于客户端上。使用动态 HTML（客户端脚本运行）作为部分输出的任何服务器编写好的 Web 页面是此类页面的一个示例。对此问题的直接反应可能是对类中的每个属性或操作进行构造，以指示它在服务器端或客户端是否有效。此时，最初旨在帮助简化事情的模型变得相当复杂。

解决问题的更好方法是考虑“分离关注点”原理。从逻辑上说，Web 页面的行为在服务器上与在客户端上是完全不同的。当在服务器上执行时，它有权访问服务器端的资源（中间级组件、数据库、文件系统等），即与这些资源具有关系。而同一个页面或该页面的流式 HTML 输出在客户机上具有完全不同的行为和一组关系。在客户端上，用脚本编写的页面通过文档对象模型（DOM）与浏览器本身具有关系，并且与该页面指定的任何 Java Applet、ActiveX 控

件或插件具有关系。对于严谨的设计人员，还可存在与客户端上的出现在另一个 HTML 框架或浏览器实例中的其它“活动”页面的附加关系。

通过分离关注点，我们可以使用一个类对 Web 页面的服务器端方面建模，使用另一个类对客户端方面建模。通过使用 UML 的扩展机制为每个 — «服务器页面» 和 «客户端页面»定义构造型和图标来区分两者。UML 中的构造型可使我们为建模元素定义新的语义。构造好的类可使用定制图标或仅仅在引号（«»）之间加上构造型名称，显示在 UML 图中。图标对于概述图很有用，其中当显示类属性和操作时，最好使用简单的标记。

对于 Web 页面，构造型表示类是客户机或服务器上的 Web 页面的逻辑行为的抽象。两个抽象彼此相关，两者之间具有有方向的关系。此关联构造为 «建立»，因为可以说服务器页面建立了客户端页面（图 1）。每个动态 Web 页面（即其内容是在运行时确定的页面）用一个服务器页面构造。每个客户端页面至多由一个服务器页面建立；但是，一个服务器页面建立多个客户端页面是可能的。

Web 页面之间的常见关系是超链接。Web 应用程序中的超链接表示通过系统的导航路径。此关系在模型中用 «链接» 构造的关联表示。此关联始终由客户端页面产生并指向客户端页面或服务器页面。

在系统中，超链接作为 Web 页面的请求实施，并且 Web 页面作为实施视图中的组件进行建模。与客户端页面的链接关联在很大程度上等价于与建立该客户端页面的服务器页面的链接关联。这是因为链接实际上是对页面的请求，而不是类抽象之一。由于 Web 页面组件实现了两个页面抽象，因此到由页面组件实现的任何类的链接都是等价的。

标注值用于定义与链接请求一起传递的参数。«链接»关联标注值“参数”是处理请求的服务器页面期望和使用的一列参数名称（和可选值）。在图 2 中，“搜索结果”页面包含到“获得产品”服务器页面的数量不定的超链接（0..*），其中每个链接对于“产品标识”参数具有一个不同的值。“获得产品”页面建立由“产品标识”参数指定的产品的“产品细节”页面。

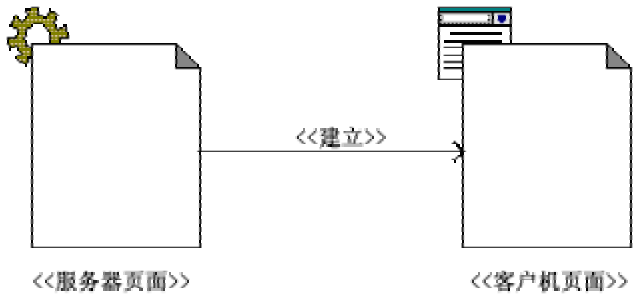

图 1. 服务器页面建立客户端页面

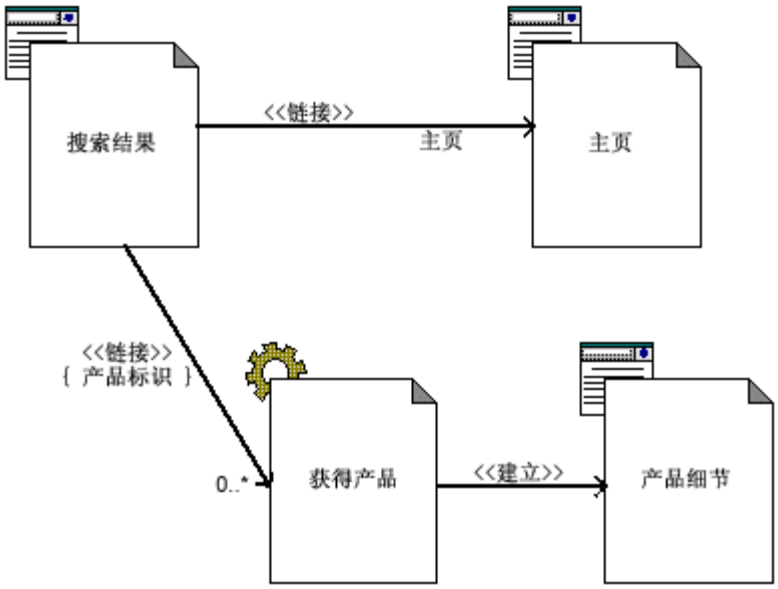

图 2. 使用超链接参数

使用这些构造型，能够更简便得对页面的脚本和关系建模。«服务器页面»类的操作变成页面服务器端脚本的函数，而它的属性变为页面范围的变量（通过页面函数可全局访问）。«客户端页面»类的操作和属性同样变成在客户端上可见的函数和变量。将页面的服务器端和客户端方面分成不同的类的关键好处在于页面和系统其它类之间的关系。客户端页面是用与客户端端资源（DOM、Java Applet、ActiveX 控件和插件，见图 3）的关系建模的。服务器页面是用与服务器端资源（中间级组件、数据库访问组件、服务器操作系统等，在图 4 中已说明）的关系建模的。

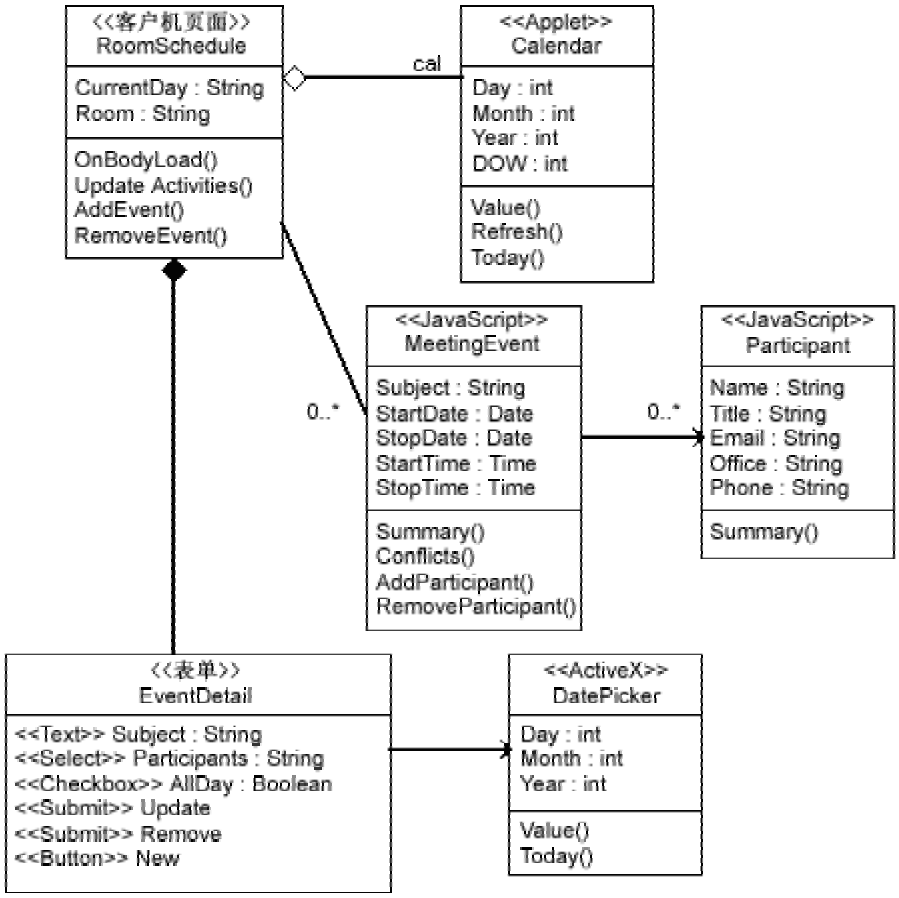
〈〈客户机页面〉〉
RoomSchedule
CurrentDay : String
Room : String
OnBodyLoad()
Update Activities()
AddEvent()
RemoveEvent()
cal
<<Applet>>
Calendar
Day : int
Month : int
Year : int
DOW : int
Value()
Refresh()
Today()
0..*
<<JavaScript>>
MeetingEvent
Subject : String
StartDate : Date
StopDate : Date
StartTime : Time
StopTime : Time
Summary()
Conflicts()
AddParticipant()
RemoveParticipant()
0..*
<<JavaScript>>
Participant
Name : String
Title : String
Email : String
Office : String
Phone : String
Summary()
〈〈表单〉〉
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. 客户端协作

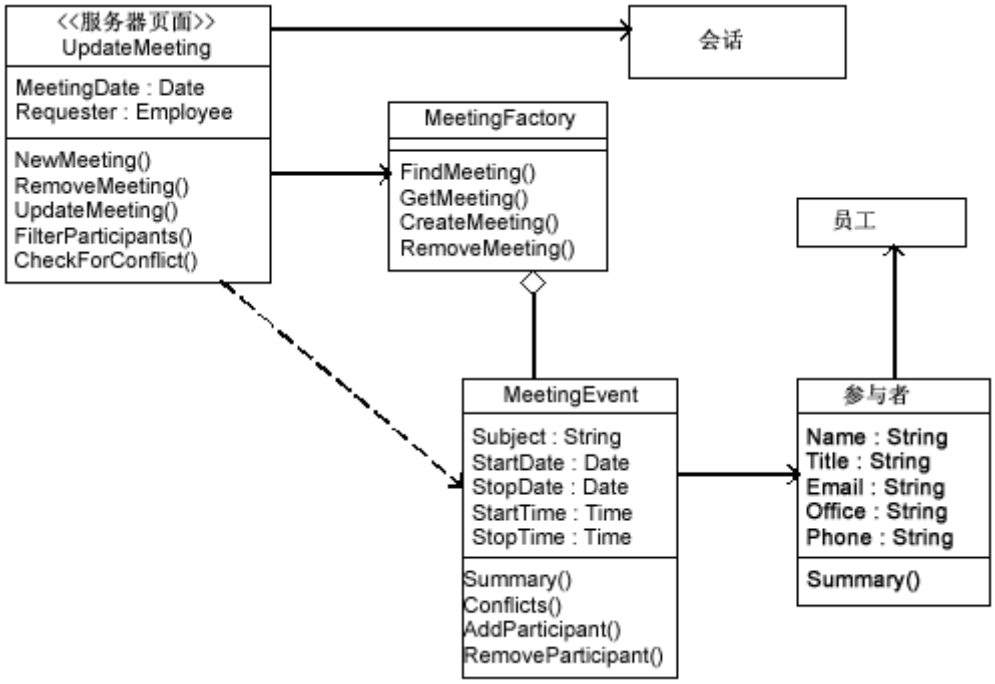

图 4. 服务器协作

使用类构造型对 Web 页面的逻辑行为建模的最大好处之一在于它们与服务器端组件的协作可用与任何其它服务器端协作类似的方式表示。«服务器页面»只是参与系统的业务逻辑的另一个类。在更概念化的层面上，服务器页面通常担当控制者的角色，协调必要的业务对象活动，以实现由浏览器的页面请求发起的业务目标。

在客户端，协作会变得复杂一点。这部分地是由于可使用的技术的多样性。最简单的客户端页面是包含内容和演示信息的 HTML 文档。浏览器使用页面中的格式化指示信息，有时使用单独的样式表，来显示 HTML 页面。在逻辑模型中，这种关系可用客户端页面到 «样式表»构造的类的依赖关系来表示。但是，样式表主要是表示问题并通常排除在 ADM 之外

表单

Web 页面的主要数据输入机制是表单。在 HTML 文档中用 <form> 标记定义表单。每个表单指定要向其提交表单的页面。表单包括许多输入元素，这些元素都表示为 HTML 标记。最常用的标记是 <input>、<select> 和 <textarea> 标记。输入标记有些负担过重，因为它可以是文本字段、复选框、单选按钮、按钮、图像、隐藏字段以及其它一些不太常用的类型。建模表单表示另一种类构造型：«表单»。«表单»没有操作，因为可能在 <form> 标记中定义的任何操作是由客户端页面真正所有的。表单的输入元素都是 «表单»类的构造好的属性。«表单»可以与作为输入控件的 Applet 或 ActiveX 控件有关系。每个表单也都与服务器页面（处理表单提交的页面）有关系。这种关系被构造成 «提交»。由于表单是完整包含在 HTML 文档中的，因此它们用一种很强的聚集形式表示在 UML 图中。图 5 显示了简单的购物车页面，该页面定义了一个表单并显示了与处理服务器页面的提交关系。

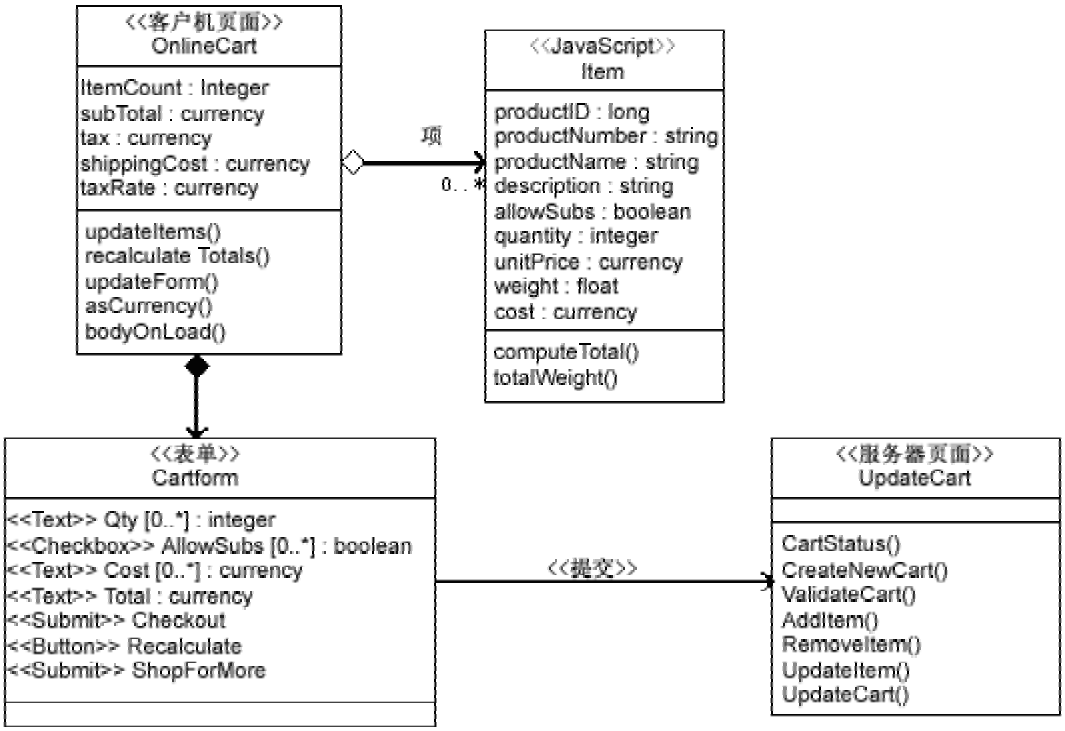

图 5. 表单提交给服务器页面

在图 5 中，«JavaScript»构造好的类是表示购物车中的商品的对象。在表单属性的描述中，对具有不定数量的实例的字段使用数组语法。在此购物车的情况中，它表示购物车中的许多商品都可为零，每件商品都具有 Qty、AllowSubs、Cost 和 Total <input> 元素。

由于客户端页面中的所有活动都是用 JavaScript 执行的，并且 JavaScript 是无类型的语言，因此为任何这些属性指定的数据类型只用于实施者说明。当用 JavaScript 或作为 HTML 输入标记实现时，忽略类型。这也适用于函数参数，这些参数虽然未全部显示在此图中，但也是模型的一部分。

框架

HTML 框架在 Web 站点或应用程序中的使用已成为一个由于它的引入而产生两级分化的争论的主题。框架可使多个页面在任意给定时间对用户来说都是激活和可视的。目前，最通用的浏览器的最新功能集也允许多个浏览器实例在用户机器上是激活的。使用动态 HTML 脚本和组件，这些页面可彼此进行交互。在客户端上进行复杂交互的可能性是很大的，而对此建模的需要甚至更大。

是否在一个应用程序中使用框架或多个浏览器实例，这是软件设计人员的决策。如果使用框架或多个浏览器实例，那么基于与以前相同的原因，此客户端行为的模型需要表示在 ADM 中。要对框架用途建模，要再定义两个类构造型 — «框架集»和«目标»— 以及关联构造型 «目标链接»。框架集类表示一个容器对象并直接映射到 HTML <frameset> 标记。它包含客户端页面和目标。目标类是由其它客户端页面引用的已命名的框架或浏览器实例。目标链接关联是到另一个页面的超链接，但在特定目标中显示出来。在图 6 所示的示例中，通用的概观视图是用两个框架显示在浏览器中的。一个框架用目标（内容）命名，而另一个框架仅仅包含客户端页面。此客户端页面框架是书的目录（TOC）。目标是此页面中的超链接，因此它们显示在内容框架中。效果是左边显示静态目录，并且右边页面按章节显示书的内容。

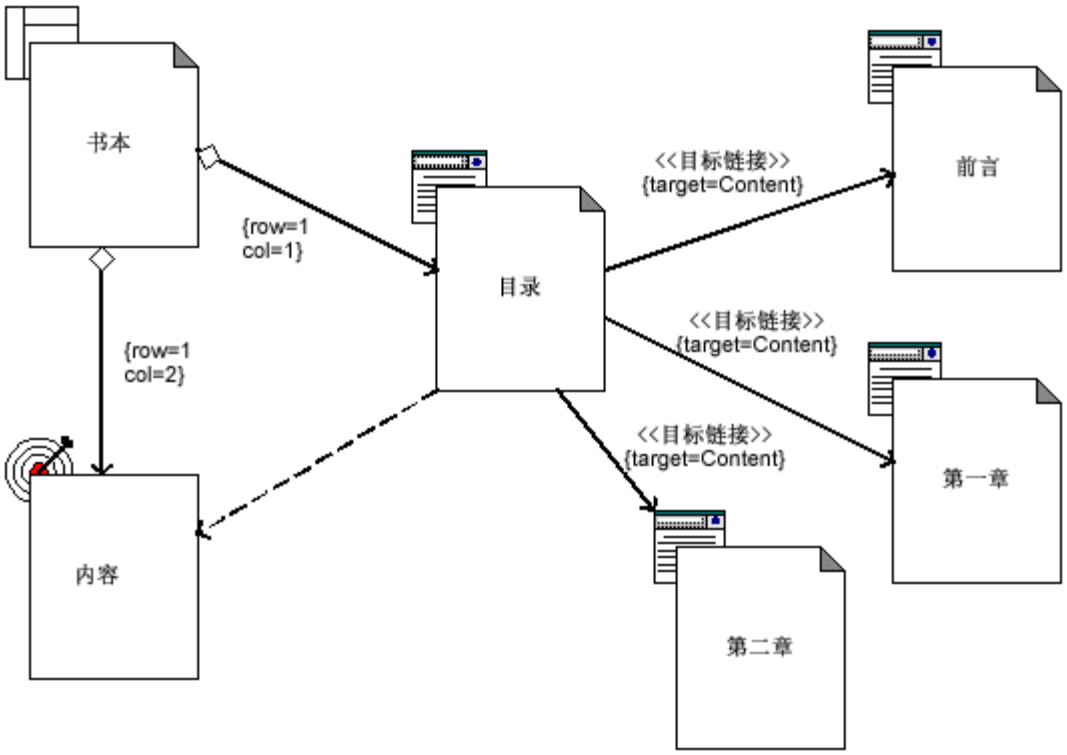

图 6. 框架示例

许多实际演示细节是由框架集和关联中的标注值捕获的。框架集和目标（或客户端页面）之间的聚集关系上的两个标注值指定目标或页面所属的框架集的行和列。目标链接关联上的标注值“目标”确定应显示页面的 «目标»。

当目标未与框架集聚集时，意味着使用单独的浏览器实例呈现页面。记住此表示法表示客户端的单个实例是很重要的。假设多个独立目标都在同一台机器上运行，并且此图表示一个客户端实例的客户端行为。为了更好地理解，需要在模型中详细记录任何其它部署配置。

结论

在本白皮书中讨论的想法和概念简介了使用 UML 为特定于 Web 应用程序的元素建模的问题和解决方案。此工作的目标是简介一种一致和完整的方法，对特定于 Web 的元素建模和应用程序的其它部分进行集成，使详细程度和抽象级别适合于 Web 应用程序的设计人员、实施者和设计人员。用于 Web 应用程序的 UML 的正式扩展的第一个版本已快要完成了。此扩展将为设计人员和设计人员提供一个通用的方法来用 UML 表示完整的 Web 应用程序设计。

关于此扩展的最新信息可在 Internet 上 Rational Software 的 Web 站点的 Rational Rose 和 UML 部分找到。

Rational®
the software development company

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

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

Toll-free: (800) 728-1212
E-mail: info@rational.com
Web: www.rational.com
International Locations: www.rational.com/worldwide