重庆大学

硕士学位论文

Oauth协议及其在社会化网络中的应用与研究

姓名：朱扬谷

申请学位级别：硕士

专业：计算机软件与理论（软件工程）

指导教师：陈林

20120527

摘 要

互联网的逐渐发展，网络规模的不断扩大，网络技术持续的推层出新，各式各样网络应用迅速地出现与融入到人们的平常生活与工作当中，并保持着越来越社会化的趋势而不断发展壮大。这一系列的变化，在使得人们的生活变得越来越轻松的同时，也呈现出了一系列越来越严重的问题。应用的增多导致人们的账号数量不断增长，用户名与密码越来越难记忆与维护，用户隐私被泄露的风险也越来越大。同时，各个应用之间用户所拥有的资源，也得不到有效的共享与互通，从而形成了一个个的孤岛，这对于社会化的人来说，也是相当不益的。

本文在探索这一系列问题的解决方案的路途中，经过分析绝对信任模式、二足模式与三足模式下的资源共享方案的利与弊，找到了 OAuth 协议这一个经过用户授权，允许第三方应用代表用户，操纵指定应用中的资源，从而实现应用之间的资源共享与互通的方案。

在国际知名的中国对外 B2C 电子商务服务提供商 DX 的相关应用开发的经验基础上，针对 OAuth 协议的 1.0 与 2.0 版本，本文首先分别详细阐述了在用户、客户端与服务提供方等角色的参与下，协议的具体工作流程与流程的每个步骤的详细参数说明。而后分别针对协议的两个版本，对协议中的各个参与者的功能进行了仔细地分析并得出了相应的用例图。在相应分析结果的基础上，本文提供了能够满足当今社会化网络的高并发与高扩展性需要的数据库与系统设计方案，而后依据此方案实现了 DX 电商系统的数据开放平台，并对方案的核心实现提供了 C# 源码示例与相应的源码说明。

针对 OAuth 的应用，本文也明确地阐述它在私有资源的跨界使用、畅通无阻地使用单一账户穿梭于各个网络应用之间以及让互联网更好地为人工作与服务等方面所发挥的巨大作用。

关键词：OAuth，社会化网络，资源共享，认证授权

ABSTRACT

With the gradual development of the Internet, the size of the network continues to expand, newthought and technology continues tocome up, more and moreweb applications are created and merged into people's daily sociallife. Because of these changes, people are enjoying a more and more easy life.But at the same time, we can't ignore that some problems are becoming more and more serious. Increasing number of application comes with the increasing account number. People are suffering from remembering and managing user names and passwords, which causes the increasing difficulty of protecting user privacies. Meanwhile, the lack of sharing users' resources between different applications is definitelya big problem for our social people.

By analyzing three kinds of resource-sharing mode, which are the full trust apps sharing mode, two-legged resource sharing mode and three-legged resource sharing mode, we findthat third-party application can act on behalf of resource owner to manipulate the resources in specified applications, after acquiring a user-authorized access tokenby using the OAuth protocol.

Based on the working experience in DX, which isthe largest foreign-oriented B2C Company in China, this paper first described the OAuth 1.0 and OAuth 2.0protocol, and their working procedures. And then this paper analyzes the actors in the protocols and works outsome use-case diagrams for the OAuth protocol. Based on the use-case diagrams, this paper continues to describe how tobuild a high performance and high scalable web applicationby using the C# language and the MongoDB database.

This paper also describes how we can benefit from the OAuth protocol, which includescross-border use of protected resources, unique identity for all web applications and also how can we put the Internet work for us.

Keywords: OAuth, Social Web, Resource Sharing, Authorization

1 绪论

1.1 研究背景

“Social tool for social life on the go.”，著名的照片分享应用软件 Instagram 的创始人兼 CEO Kevin Systrom 在接受洛杉矶时报的采访如是说[1]。

随着互联网的逐渐发展，网络规模的不断扩大，网络技术持续的推层出新，各式各样网络应用也迅速地出现在人们的视野中，并保持着从尝试融入到丰富、从丰富到渗透至人们生活的方方面面的趋势。从 Web 1.0 时代的最初的单方面的简单的资讯获取，到 Web 2.0 的产生，人们越来越注重在互联网上人与人之间的交互。从较早发展起来的博客、照片存储与共享服务、视频服务、网络交友、互动游戏等应用，到最近几年涌现出来的 SNS 社交网络、微薄等应用，都极大地满足了人们对信息分享与聚合的需求。同时，随着电子商务的兴起，B2B，B2C，C2C，团购等各种电子商务模式的不断成熟与相应模式下各种团队与企业的不断发展壮大，人们也都能极其方便地通过互联网满足对于衣食住行等生活与工作的需求。

与此同时，由于硬件技术与工艺的不断改进与完善，各种移动终端在其便携与续航性能上都有显著提升。而 3G 移动网络技术的出现，更使得移动互联网在速度与稳定性上也达到了前所未有的高度。在此基础上，移动终端由于其具有的移动性、即时性、便利性与私人性这四大特性，使得移动互联网也进入了一个崭新的时代。Apple、Google 等厂商，纷纷研发出了适用与当前移动互联网终端的操作系统。各大传统互联网应用提供方，也络绎不绝地发布着相应的支持各种移动操作系统移动应用。一批批新的移动互联网应用也应运而生。诸如照片实时分享应用 Instagram，数据显示，在该应用发布的 24 小时内，就拥有了 2.5 万用户，1 周后达到了 20 万用户，3 个月达到了 100 万用户，而在 18 个月后，这个数字达到了 3 亿！在其发布仅 1 年的时间里，人们通过该应用分享的照片数量达到了达到了 10 亿[2]。日前而言，每秒就有 15 张图片产生，每天有 130 万的图片会被通过此应用分享至 Facebook、Twitter、Tumblr 甚至是新浪微薄等第三方互联网应用中。另外一些基于地理位置服务的移动应用如 Foursquare、微信、移动地图等，使得人们不仅可以方便地与好友分享当前所在的地理位置，更能让人们方便地找出处在附近的好友、超市、学校、医院与银行等，这些服务都极其方便了人们的娱乐与生活。我们可以说，这些应用，在将移动互联网的特性表现得淋漓尽致的同时，更让人与人，人与互联网紧密地、无时不刻地联系在了一起。

因此我们看到，以前被认为是虚拟的网络，在与社会化的人联系得越来越紧密的同时，也逐渐显现出突出其社会化的作用。

然而，在社会化网络的发展过程中，随着形形色色的网络应用的出现，我们也发现了不少的问题。第一，对于应用的增加，账号管理成了一个麻烦的事情。人们经常需要在各自独立的应用里面注册一套属于自己的账号，这就导致了每一个人不得不管理几套甚至是几十套账号，这对一个普通人来说，是一件比较麻烦的事情。其次，应用的良莠不齐，也加大了用户隐私泄漏的可能性。保护用户隐私，应当说是每一款应用最基本的责任。然而，2011 年 12 月，中国的著名 IT 社区 CSDN600 万用户密码泄露事件与接踵而至天涯社区、人人、开心网等十几个国内大型网络或信息数据库被曝光而导致的总数将近 1 亿的用户隐私被泄露事件[3]，对整个网络造成了相当恶劣的影响，人们都纷纷议论并寻找着更加安全的账号运营方或者托管方。第三，不同应用之间的资源共享也成了一个亟待解决的问题。人是社会的人，人们总是乐于合理地运用其所拥有的一切资源来完成一系列的社会活动。这也就导致了，各个独立的应用，不太可能会处在一个完全封闭的环境中，而会是处在一个获得合理授权下，能够运用第三方应用中的特定资源的环境中。

1.2 课题来源与研究意义

本课题来源于国际知名的中国对外 B2C 电子商务服务提供商 DX（全称为 DealExtreme，网址为 http://dx.com）的数据开放平台项目与 Passport 项目。作为国内营业额最大的对外电子零售服务提供方的 DX，因其提供着高品质而又相对比较廉价的商品，在全球 200 多个国家和地区内拥有着相当广泛的用户群。

而伴随着社会化网络的发展，为了拓宽市场与满足客户需求以适应时代变化，DX 需要相应地允许诸如移动客户端的第三方应用的接入，以方便客人随时随地地进行商品浏览、下单购买、查单改单以及实时分享信息至 Facebook、Twitter 等应用的一系列活动。在此项目中，如何确保移动客户端能够正确而且安全地查询与操作指定账户的数据，以及如何将客人在 DX 拥有的私有信息与资源跨界分享至诸如 Facebook 等第三方网络应用中，则成了一个重要的问题。同时，由于 DX 在 Facebook 的公共页面上也拥有着庞大的粉丝群，为了方便 Facebook 用户快速地入住 DX，用作 DX 账户信息管理的 Passport 项目也需要能够提供使用 Facebook 账户直接登录 DX 的支持。因此，如何让 DX 能够获取并使用客人在 Facebook 的 Email、姓名、年龄、性别以及所属国家或地区等账户信息，又成了另一个重要的问题。总的来说，这一系列举动，对于 DX 来说，意义非常，而攻克这一系列的技术问题，又成了重中之重。如若这些问题不能得到妥善解决，则这一系列项目也不可能成功。

在 DealExtreme 的工作积累，加上一直使用 Gmail、Google+、Facebook、Twitter、QQ、微信、新浪微博等一系列社会化网络应用的实际经历，并结合长期关注并不

断参与社会化网络发展与建设所得的经验，本人通过理论结合实际的方式，深入地研究了支撑这一系列社会化网络应用协同工作所运用的最通用的交互与授权协议– OAuth 协议。

通过研究 OAuth 协议，对于解决社会化网络在账号管理比较麻烦、隐私安全泄漏问题以及跨应用资源共享等问题有着积极的作用。同时，这也能加深对于如何更安全、高效地开发社会化网络应用以及如何合理地使用社会化网络应用以帮助我们更好地学习、生活与工作等方面的理解。

1.3 国内外现状研究

就传统意义而言，人们所认知的网络应用普遍是指基于 B/S（Browser/Server）模式的应用，也即是通过浏览器发起访问请求后，网络应用服务器，如 IIS，Apache 等负责处理程序逻辑，进行数据读取操作，而后将相关数据返回至客户端，浏览器负责数据展示与用户交互的这一模式上的应用，亦即人们口头常说网站。而 C/S（Client/Server）模式普遍应用于桌面应用程序，即数据读取操作与大部分程序逻辑在服务端完成，客户端除了负责用户交互工作外，还需要完成一部分的业务逻辑。而近几年以来，我们看到，越来越多的传统网络应用服务提供方都纷纷推出了基于 C/S 模式的桌面客户端，这其中又以移动桌面客户端居多。由于移动客户端在计算能力上的限制性，越来越少的逻辑被包含在了客户端而使得客户端慢慢地转变成了只起到类似 B/S 模式中浏览器这一方所起的作用，即负责数据展示与用户交互工作。同时，我们也看到，这些客户端与传统的网络应用也结合得相当紧密，传统应用网络变成了是它们赖以生存的环境，同时大部分通过浏览器完成的操作也能通过这些客户端来完成。可以说，这种 C/S 模式与 B/S 模式的界限越来越小，故而，我们也可以把这一类的 C/S 模式的应用称之为网络应用。

那么在网络应用如此繁多的今天，在面对账户管理麻烦、客户隐私保护工作难做与资源的如何跨界使用等问题面前，业界的技术先驱提出过什么样的理念，实践者又尝试过什么样的解决方案呢？

首先从账户管理与保护隐私谈起。从统计学上来说，假定一个账户被曝光的概率是稳定的，那么一个人的账户越多，隐私被曝光的可能性也就越大。同时，进行相应的补救措施所消耗的成本也就越高。故而，一个人尽量保持较少的账户数据量，无论在降低账户维护成本方面，还是增强隐私保护方面，都能起到良好的作用。有鉴于此，2005 年 5 月，知名社区网站 LiveJournal 的创建人 Brad Fitzpatrick，发布了 OpenID 协议的初始版本。至今时今日，OpenID 已经发展到了 2.0 版本[4]。该协议认为，在网络中，每个人都可以用一个唯一 URL 地址来表示，每一个 URL 也代表着唯一的一个人。通常而言，这个代表着一个人的唯一的 URL，是由 OpenID

服务提供方颁发给用户的。人们凭借着代表他的唯一的 URL 作为用户账户，就可以使用着不同的网络应用而只需要管理这一个 OpenID 账户。故而，只要选择优质而且值得信赖的 OpenID 服务提供方，人们就可以在相当大的程序上降低账户管理的成本以及减小隐私被泄露的可能性。OpenID 协议的流程如图 1.1。

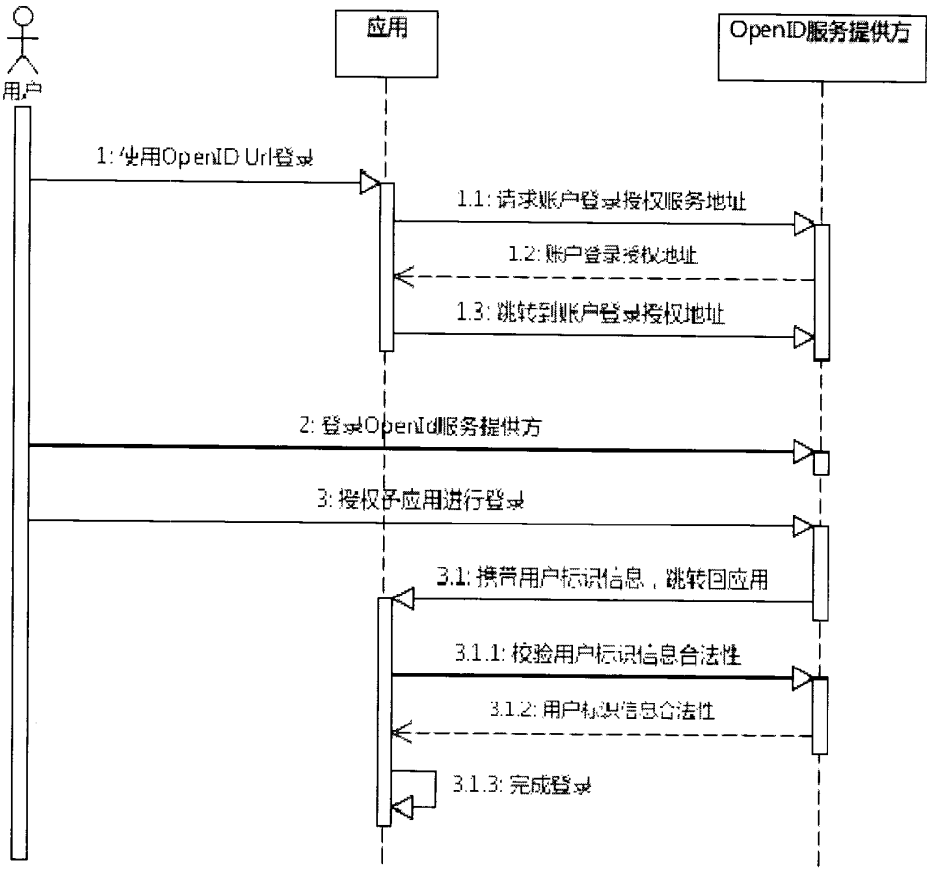

图 1.1OpenID 协议工作流程图

Figure 1.1OpenID Working Flow Chart

在 OpenID 协议[5]中，用户在应用中输入 OpenID 颁发的 URL 进行登录后，应用通过向该 URL 发起请求，而后获取到 OpenID 服务提供方返回的基于 XRDS 格式标准的登录授权地址，进而转向至处于 OpenID 服务提供方的该地址。然后，OpenID 服务提供方可以要求用户登录并授权予以应用使用当前账户进行登录，并携带着用户信息，跳转回应用（当然，如若用户已经登录或者已经授权予以过应用，OpenID 服务提供方可以免去相应的要求用户进行的操作以更快速引导用户登录至应用）。应用此时需要对用户信息的合法性进行一次校验，并完成登录。

在这个工作流程中，我们看到，每个人只需要关心唯一的一个账户登录凭证 - OpenID 的登录凭证，即可顺利地使用各种支持 OpenID 的应用。放眼业界，目前

OpenID 使用也不可谓是不广泛。就 OpenID 服务提供方而言，除了该协议的作者所处的 LiveJournal 外，Google、Yahoo、AOL 等国际知名厂商都支持第三方应用通过 OpenID 协议使用其账户登录。同时，我们也能看到专注于提供 OpenID 服务的网络应用，如 myOpenID.com。就支持 OpenID 协议进行登录的应用而言，其数量也可谓是数不胜数。以全球最著名、质量最好的 IT 技术问答问题社区 StackOverflow 为例，在其登录页面，使用 OpenID 服务登录的功能被摆在了最重要的位置，而其自身所带有的注册登录功能，却被放到了次要的位置。由此可见，OpenID 协议已经发挥着出了其相当大的作用。

再来看看资源跨界使用与共享的问题。由于受到应用所处的软硬件环境、运行模式、自身逻辑甚至是其实现语言的影响，不同的应用之间可能存在着无法通信、可以受限制地通信与自由地通信等三种情况。对于相互之间无法通信的应用，也就不存在资源跨界使用与共享的问题了。对于后两种情况，应用之间又存在着信任关系。这也就导致了针对不同信任的应用之间，资源跨界使用与共享的工作方式又不尽相同。

对于存在绝对信任关系的应用之间，资源的共享相对比较简单。如图 1.2 所示，如果应用 B 完全信任应用 A，应用 A 在请求应用 B 的资源时，应用 B 可以直接地将所需要的资源返回。

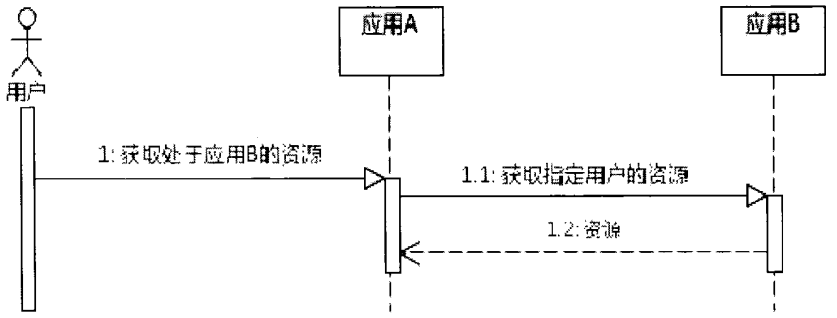

图 1.2 绝对信任模式下的资源跨界使用与共享工作流程

Figure 1.2 Resourcesharing flow between different appsinfull trust mode

绝对信任模式被广泛地应用在公共服务应用中以及同一服务商提供的内部应用当中。比如，我们手机中的天气应用，一般就是直接访问气象部门提供的公共服务平台即可直接获取，即这些公共服务信任任意的调用方。当然，企业的内部系统之间，由于各系统之间能够建立这种绝对信任的关系，故而也使得这种工作方式应用相当广泛。可以说，这种模式是最简单，也是最直接的资源共享模式。

然而，遗憾的是，能够相互之间建立绝对信任的应用，在所有的应用当中，占的比例实在是太少了。我们见到的绝大多数往往是在相互之间不能建立绝对信任

关系的应用。于是，我们又有了另外的一种模式：二足模式。由于基本上所有的应用都有要求用户建立登录凭证（登录名+登录密码）。有了合法的登录凭证，便有了使用相应资源的资格。于是，在跨界资源使用时，调用方携带着用户在被调用方的登录凭证，就可以获取得用户在被调用的应用的资源了。如图 1.3 所示，用户在将其在应用 B 的登录凭证授予给应用 A 后，应用 A 发起请求，在应用 B 通过了登录凭证校验后，就即可返回相应的资源信息给应用 A。

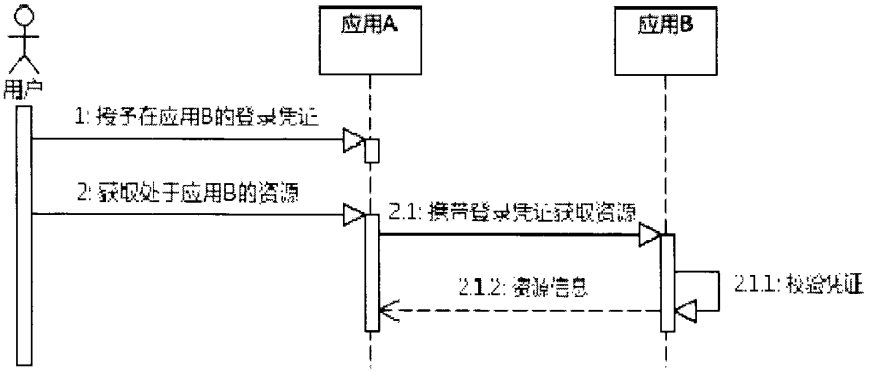

图 1.3 二足模式的资源跨界使用与共享工作流程

Figure 1.3 Two-legged resourcesharing flow

此模式应用得相当广泛。以我们常用的邮件系统为例，一般来说，邮件系统，除了使用浏览器访问以外，还支持 POP3 与 SMTP 协议。邮件客户端也就是在获取了用户的邮件系统登录凭证后，通过这些协议，就能够随意地完成发送，回复与删除邮件等操作。那么，这也就我们的网络应用之间跨界资源共享创建了条件基础。目前很多邮件系统提供的代收邮件功能就是一个很好的例子。例如在网易 163 在线邮件中，我们可以通过其代收邮件功能，在输入了 QQ 邮箱的登录名与密码后，就可以收取本属于 QQ 邮箱的邮件。另外，在博客系统中，此模式也得到了很好的运用。2010 年 9 月，微软宣布关闭其 Windows Live 博客服务，导致其 3000 万用户不得不选择将他们发布在 Windows Live 的博客迁移到诸如 WordPress 的其它博客服务中[6]。而这自动化迁移过程中所运用的便是基于此二足模式下依托于 XML-RPC（XML Remote Procedure Call）[7]通讯方式的 MetaWeblog 协议[8]。

然而，需要将用户凭证告知第三方应用的二足模式却给用户带来了很大的安全隐患。第一，第三方应用能否保护好用户凭证是个需要优先担忧的问题。第二，第三方是否会滥用用户凭证成了一个值得仔细思考的严重问题。用户登录凭证即完全代表着用户，拥有了用户用户凭证，即拥有着完全操作用户所拥有的资源的绝对权力。以网易邮箱收取 QQ 邮箱的邮件为例，我们知道，QQ 的一整套账户集

成功能做得相当优秀，使用 QQ 账户与密码即可登录任意的 QQ 服务与应用，包括 QQ 空间、QQ 微博、财付通以及 QQ 邮箱等。那么，我们将 QQ 的登录凭证交给网易邮箱后，是否意味着，网易邮箱就拥有了操纵我们所有 QQ 相关应用的权力呢？答案是肯定的。因此，一旦第三方应用没能妥善处理用户在其它应用中的登录凭证，其造成的后果将是相当可怕的。当然，这也是我们不愿意看到的。

通过分析二足模式，我们看到其最明显的弊端出现在了用户登录凭证的授予上。同时，在授予了登录凭证后，作为资源的拥有者，用户也不能对应用之间的交互做到有效的权力控制。故而，从业者思考着一种在无需将用户登录凭证授予给第三方应用的情况下，又可使得应用之间能够可控地操纵指定资源的解决方案。这便产生了在二足模式基础上添加了用户这一角色的三足模式。即在应用之间的交互过程中，需要用户参与并进行控制的一种模式。

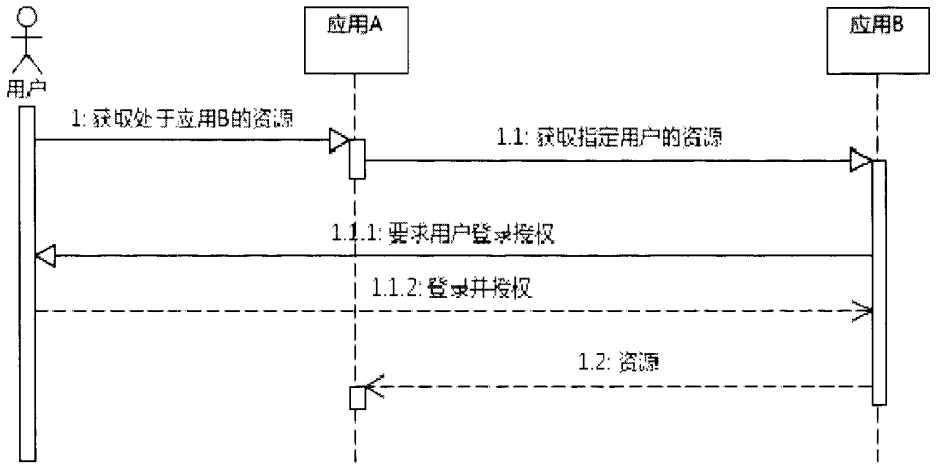

图 1.4 三足模式的资源跨界使用与共享工作流程

Figure 1.4 Three-legged resourcesharing flow

如图 1.4 所示，在应用 A 获取用户处于应用 B 的资源的过程中，应用 B 要求用户参与进来，并进行相应的授权，而后再根据用户的决定，选择是否将指定的资源共享给应用 A。

早在 2006 年，当 Google 公司发布出它的 Google Calendar API v1.0[9]以允许用户在第三方应用中操作其 Google 日历时，就使用的是这种三足模式。而它的统一授权的系统，被称为 AuthSub，因此，其协议也被称为了 Google AuthSub 协议。与此相似，Yahoo 公司在收购了图片分享网站 Flickr 后，也发布出了三足模式的 BBAuth 协议[10]。不幸的是，这些协议基本上都只能在其所属公司的系统中使用，而没能最终成为真正的互联网标准协议。然而万幸的是，有别于 AuthSub 与 BBAuth 协议，当 Twitter 的工程师在处理相同问题时，考虑到了要将其解决方案- OAuth

协议[11]标准化。于是2007年OAuth讨论小组成立，并于2008年11月，在明尼阿波利斯举行的互联网工程任务组（IETF）第73次会议上，OAuth被得到了广泛认可，并在IETF下成立了专门的工作组–OAuth工作组。2010年4月份，OAuth 1.0终于正式成为了互联网标准协议，具体为RFC 5849: The OAuth 1.0 Protocol[12]。有了标准的支持，OAuth很快就被推广了开了。包括Twitter、Google与Yahoo在内的越来越多的公司都支持该协议。当然，随着时间的发展，人们认为OAuth 1.0还是具有一定的复杂性，因而相对比较精简的OAuth 2.0协议也在规划与完善中，目前已经发展至第25个修改版并会在不久的将来发布正式版。自从Facebook率先表示支持OAuth 2.0协议并将其应用于Facebook Graph平台开始[13]，Google+、Yammer等社会化网络应用也纷纷开始了对OAuth 2.0协议的支持工作。同时，在国内我们也看到，业界呈现出了一片欣欣向荣的景象：2011年，在人人网率先升级人人开放平台并支持OAuth 2.0；而后的6月，腾讯发布开放平台战略，推出使用OAuth 2.0的微博、财付通、“Q+”等八大开放平台；9月上旬，百度亦推出其新的开放战略[14]。在这些开放平台中，OAuth 2.0协议作为第三方应用与开放平台进行协同工作的核心协议而得到了很好的运用。

可以这么说，OAuth协议的出现，加速了整个互联网应用由竞争与各自封闭的状态向相互合作的转变，同时也使得社会化网络迈上了一个新的台阶。故而，研究OAuth协议，对于完善包含电子商务在内的社会化网络应用与优化整个网络环境，有着重要的意义。

1.4 本文主要工作

本文首先研究了OAuth协议的具体工作流程，并基于此协议设计与实现了DX电子商务平台的数据开放平台项目与DX Passport项目中的使用第三方账户快速入住DX的功能。

其次，在对OAuth协议了解的基础上，本文针对如何解决社会化网络环境中资源跨界使用与共享问题、账户管理麻烦与隐私保护工作难等问题进行了研究，并提出了积极有效的意见与建议。

最后，本文对如何利用OAuth协议以促使互联网如何变得更加社会化，从而使其能够更好地服务于社会，提出了相当具有建设性的意见并做出了相应的技术可行性验证。

1.5 本文章节结构

在相关背景的大环境下，结合课题来源，并在仔细分析与研究了当前国内外现状后，特将本文章节分配如下：

第一章，绪论：介绍了相关研究背景，课题来源与研究意义，同时详细地分析了国内外现状，并对本文的主要工作进行了简要的阐述。

第二章，相关技术原理：首先详细阐述了 OAuth 协议及其工作流程，而后对 NoSQL 与 MongoDB 数据库、RESTful Web Service、JSON 等技术等进行了描述，为本课题提供技术基础。

第三章，基于 OAuth 协议的电商数据平台与应用的分析与设计：在对 OAuth 协议及其工作流程的学习与分析的基础上，提出相应的系统设计方案。

第四章，基于 OAuth 协议的电商数据平台与应用的实现：针对 OAuth 协议的 1.0 与 2.0 版本，分别细致地阐述了它们的工作流程与业务逻辑的实现。

第五章，OAuth 协议在社会化网络中的应用研究：有针对性地研究了 OAuth 在解决了社会化网络的相应问题后，对如何利用此协议以服务于社会进行了深入的研究。

第六章，结论与展望：通过对课题的研究与实践，深切地表达出了作者的感触以及对后续工作与生活的展望。

2 相关技术原理

2.1 OAuth 协议与其工作流程

经过几年的发展，OAuth 经历 1.0 版本的定型与 2.0 版本的 25 个修改版本。其实 OAuth 2.0 从它的第 12 个修改版本开始，就变得相当地稳定，Facebook 就是从这个时候开始支持 OAuth 2.0 协议的。后续的修改主要是在协议措词的修改上，而对于整个工作流程没有做太多变动。OAuth 2.0 版本的最终定稿也是指日可待的事情。目前大部分的开放平台上也都是基于 OAuth 2.0 版本，只有一少部分仍然在使用 1.0 版本而暂未进行升级，故而在介绍完 1.0 版本后，我们讲重点学习与研究 2.0 版本的工作流程。

2.1.1 OAuth 1.0 及其工作流程

① 协议概览

在 OAuth 1.0 协议中，由于它是基于三足模式的，故而参与者有用户，作为请求资源方的应用 A 与作为资源持有方的应用 B。我们分别称这三者为资源拥有者（Resource Owner），客户端（Client）与服务提供方（Server）。同时，我们称需要请求的资源为资源所有者的私有资源，也叫受保护的资源（Protected Resource）。例如，在 DX 移动终端项目中，移动终端应用为客户端，DX 数据开放平台为服务提供方，而用户在 DX 的订单数据等则被称为受保护的资源。

在整个工作流程中，我们需要使用到一系列的证书（Credential）。证书是由代表着证书的唯一标识（Unique identifier)与相应的共享密钥（Shared Secret）组成。证书始终是由服务提供方颁发给客户端，它总共可分为三类：客户端证书、临时证书与令牌证书。这三类证书会被适时地应用到整个工作流程中的客户端鉴别、授权申请和授予以及资源的获取等不同环节中。

同时，服务提供方会适时地授予一些唯一的凭证给客户端以支持客户端携带着凭证对服务提供方发起相应地操作。这些凭证，我们称之为令牌（Token）。

我们需要认识到，OAuth 是依赖于 HTTP 协议的，服务提供方只提供基于 HTTP(s)的请求，客户端向服务提供方发起的任何操作均靠 HTTP 请求来完成，而用户在服务提供方进行的相关操作也需要通过用户代理客户端（User-Agent，通常为浏览器或者客户端内置的浏览器）来完成。

OAuth 协议总共可分为准备阶段、授权阶段与资源调用这三个阶段。

② 准备阶段

在开始 OAuth 1.0 的工作流程之前，我们需要将我们的客户端在服务提供方进行登记注册，并获得相应的客户端证书，即客户端标识与客户端私钥。

在注册客户端应用的时候，除了填写客户端名称外，更为重要的是，我们需要选择客户端完成对受保护资源的操纵所需要使用的权限。

③ 授权阶段

OAuth 1.0 授权阶段需要经历临时令牌的请求、用户授权、访问令牌的请求等三个过程。大概过程如图 2.1 所示。

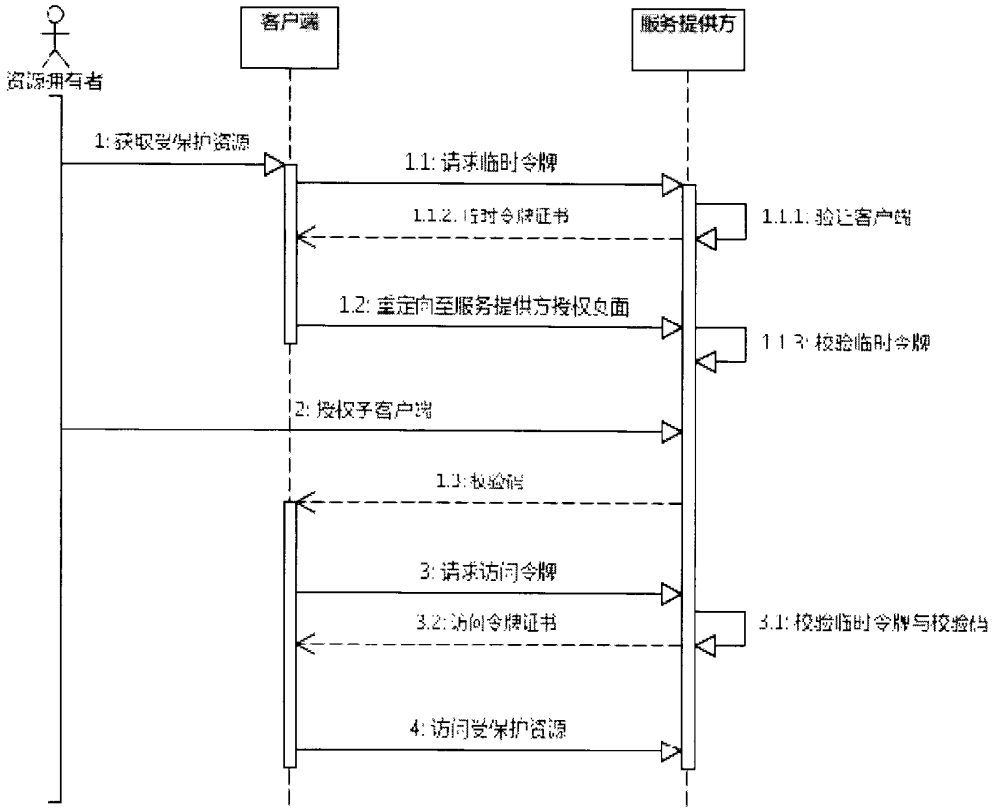

图 2.1OAuth 1.0 授权整体流程图

Figure 2.1 OAuth 1.0 general working flow

1) 临时令牌申请过程

临时令牌的请求过程中，客户端向服务提供方的临时令牌发放服务地址发起 HTTP 请求。如表 2.1 所示，在临时令牌请求过程中，客户端必须附加上服务提供方颁发给它的唯一标识、客户端回调 URL、时间戳，客户端请求标识等参数。同时，整个请求消息在进行了一次签名后，所使用的签名算法以及最终的签名也得传递给服务提供方。服务提供方在接收到请求后，需要验证客户端的合法性，并按照请求中的签名算法，重新将请求参数进行一次签名并确保签名结果与请求中的消息签名的一致性后，方可认为请求数据的是有效而未被篡改过的。当使用的签名算法不为 PLAINTEXT 时，协议规定，oauth_timestamp 与 oauth_nonce 参数均

不得为空，同时若 oauth_token、oauth_timestamp 与 oauth_nonce 三者的组合在历史请求中曾经出现过的话，服务提供方可以选择将拒绝当前请求。在后续的步骤中，我们将看到，对于任意一个从客户端发到服务提供方的请求都得进行签名，具体的签名算法原理将会在 2.1.1.5 小节中进行介绍。同时，由客户端向服务提供方传递参数及其值的具体方式，协议规定，可以使用如下三种方法中的任意一种：

a. 将参数与值附加到 HTTP 请求头的 Authorization 部分；
b. 使用 HTTP POST 方法；
c. 将其附加到 HTTP 请求 URL 的 Query String 部分。

表 2.1 OAuth 1.0 临时令牌请求参数表

Table 2.1 OAuth 1.0 temporaryrequest token request parameters

请求参数	是否必须	备注
oauth_callback	Y	客户端回调 URL，若客户端没有回调 URL，则必须设置为“oob”
oauth_consumer_key	Y	客户端唯一标识
oauth_signature_method	Y	消息签名算法，通常为 HMAC-SHA1、RSA-SHA1 与 PLAINTEXT 这三种方法中的一种
oauth_timestamp	N	时间戳，一般为 1970-01-01 00:00:00GMT 开始的秒数，当使用非 PLAINTEXT 签名时必须设置
oauth_nonce	N	请求的标识，用户防止重复攻击，当使用非 PLAINTEXT 签名时必须设置
oauth_version	N	如果设置的话，则必须为“1.0”
oauth_signature	Y	消息签名

在服务提供方对请求进行验证通过后，生成一个临时令牌证书，并按表 2.2 所示参数将此令牌返回，具体的返回方式为 HTTP 响应状态为表示成功的 200，响应内容参照 URL 中 Query String 部分的格式进行拼接，如表 2.3 所示。

如若验证不通过，服务提供方则返回HTTP状态码400以告知客户端请求无效。

表 2.2 OAuth 1.0 临时令牌请求服务响应参数表

Table 2.2 OAuth 1.0 temporary request token response parameters

返回参数	是否必须	备注
oauth_token	Y	令牌标识
oauth_token_secret	Y	令牌的共享私钥
oauth_callback_confirmed	Y	标识是否成功，始终为“true”

表 2.3 OAuth 1.0 服务提供方响应数据的格式样例

Table 2.3 OAuth 1.0 server response data sample

oauth_token=v8hzFFxXOJDsVgoKRfoCiH1BXtXz5AqGxCFuTLR7w&oauth_token_secret= wa-BuEtpB0z010SShmGWUqaWLRR6IjMRXFJ2Ikd7T7L4&oauth_callback_confirmed=true

2) 用户授权过程

在客户端获得临时令牌后，即将用户引导到服务提供方的授权页面，从而进入了用户授权过程。

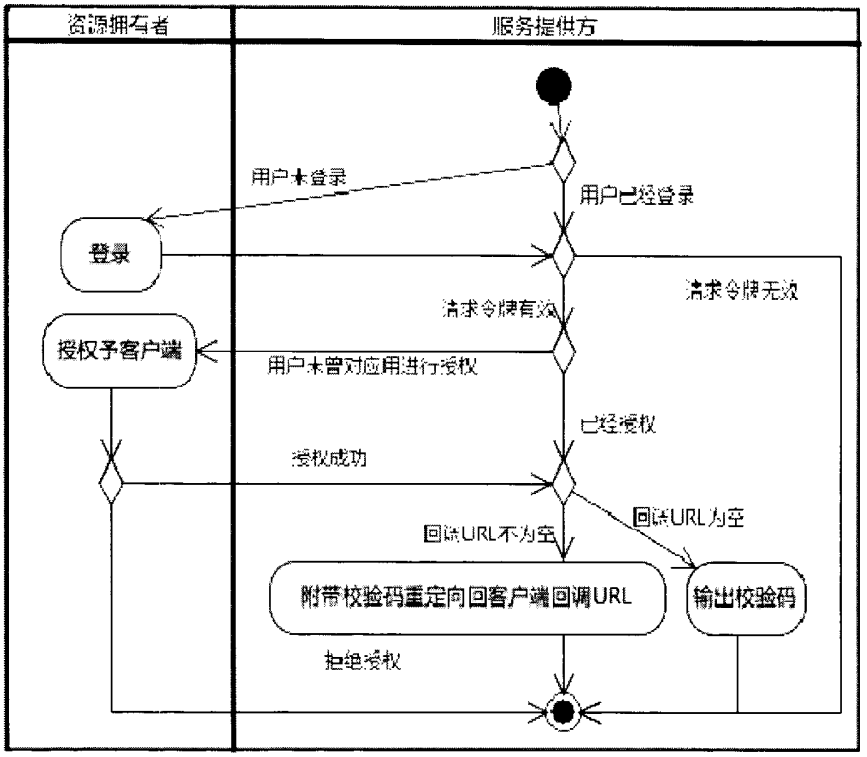

图 2.2 OAuth 1.0 用户在服务提供方进行授权流程图

Figure 2.2 OAuth 1.0 user authorization flow in server side

如图 2.2 所示，在此过程中，客户端必须将上一阶段中所获得的临时令牌标识作为 HTTP 请求 URL 的 query string 中的 oauth_token 参数通过用户代理程序（浏览器或者客户端内嵌的浏览器）传递给服务提供方，服务提供方在要求资源拥有者登录后，根据临时令牌，识别对应的客户端、客户端所需要的权限以及此次操作中客户端的回调 URL 后，等待用户授权或者拒绝。若用户授权成功，则需要根据在上一阶段中，客户端传递的回调 URL 是否为空，来决定是附带着校验码自动将用户引导至客户端的回调页面还是将相应的校验码输出至页面。如果是后者，

客户端一般需要与用户代理程序进行交互以获取校验码，或者直接让用户将校验码输入至客户端。该过程的返回参数如表 2.4 所示。

表 2.4 OAuth 1.0 用户授权服务响应参数表

Table 2.4 OAuth 1.0 user authorization server response parameters

返回参数	是否必须	备注
oauth_token	Y	临时令牌标识
oauth_verifier	Y	临时令牌校验码，用于下一过程中访问令牌的获取

3) 访问令牌申请过程

在获得用户授权的许可下，客户端便可以通过访问令牌请求流程申请访问令牌，以使得最终的资源调用成为可能。访问令牌请求的参数包括：客户端标识、临时令牌标识、用户授权过程中获得的校验码、时间戳、请求标识、签名算法以及消息的签名等。具体参见表 2.5。

表 2.5 OAuth 1.0 访问令牌请求参数表

Table 2.5 OAuth 1.0 access token request parameters

请求参数	是否必须	备注
oauth_consumer_key	Y	客户端标识
oauth_token	Y	临时令牌标识
oauth_verifier	Y	校验码，在上一过程中最终获得
oauth_signature_method	Y	消息签名算法
oauth_timestamp	N	时间戳
oauth_nonce	N	客户端向服务提供方发起请求的标识
oauth_version	N	如果设置的话，则必须为“1.0”
oauth_signature	Y	消息签名

服务提供方在接收到访问令牌请求时，先校验了请求数据的有效性，再确认令牌标识的校验码与客户端传递过来的校验码的一致性，而再确保用户确实已经进行过了授权后，将临时令牌做废处理，并最终生成全新的访问令牌并照表 2.6 所示参数返回。

表 2.6 OAuth 1.0 访问令牌服务响应参数表

Table 2.6 OAuth 1.0 access token response parameters

返回参数	是否必须	备注
oauth_token	Y	令牌标识
oauth_token_secret	Y	令牌共享私钥

客户端在获取到了访问令牌后，便可将此令牌存储起来。以供后续的对资源的相关操作的进行。同时，在用户也无需每次在客户端操作位于服务提供方的受保护资源前，都进行相应地授权。

至此，整个授权阶段完成。

④ 资源调用阶段

在获取到了访问令牌之后，凭借着访问令牌，再根据服务提供方的相应接口说明，就可以进行相应的资源访问了。除了服务提供方接口特别声明的参数外，表 2.7 中的参数为也必须进行相应的设置以方便服务提供方完成客户端与客户的识别，请求的有效性验证以及相应操作的鉴权等一系列操作。当然，为了避免与表中的基础参数产生冲突，协议规定，资源获取的接口参数均不得以 oauth 开头。

表 2.7 OAuth 1.0 资源获取基础参数表

Table 2.7 OAuth 1.0 basic resource operating request parameters

请求参数	是否必须	备注
oauth_token	Y	访问令牌标识
oauth_consumer_key	Y	客户端标识
oauth_signature_method	Y	消息签名算法
oauth_timestamp	N	时间戳
oauth_nonce	N	客户端向服务提供方发起请求的标识
oauth_version	N	如果设置的话，则必须为“1.0”
oauth_signature	Y	消息签名

当然，资源调用相关接口的返回数据格式已经超出了 OAuth 协议的范围。服务提供方可以自由选择返回数据的格式，而业界比较通用的有 XML 与 JSON 这两种格式。同时，客户端根据服务提供方的返回数据，完成进行相应的解析操作后，便能够为资源拥有者有效地服务了。

⑤ 签名过程

前面我们提到，为了确保每一个请求的有效性，客户端需要对请求进行签名。

服务提供方也需要进行相应的签名验证工作。

需要进行签名的数据由如下几个部分组成：HTTP 请求方式（GET 或 POST）、参照 RFC3986 中 URL 编码要求进行编码后的请求 URL 的绝对地址、格式化的请求参数及其值。各个部分通过“&”拼接起来。参数的格式化按照如下步骤进行：排除 oauth_signature 参数、将剩余的参数按其名称进行升序排列、将排序好的参数及其值按 URL 中 Query String 部分的格式逐一进行拼接、将拼接好的参数按 RFC3986 规范进行一次编码。表 2.8 所示为临时令牌获取时的示例请求及其签名数据。

表 2.8 OAuth 1.0 签名数据示例

Table 2.8 OAuth 1.0 signature data sample

HTTP 请求	POST /oauth/request-token HTTP/1.1 Host: open.dx.com Content-Type: application/x-www-form-urlencoded Authorization: OAuth oauth_consumer_key="GmDIWcLRiPzvbUoZRAbWiA", oauth_signature_method="HMAC-SHA1", oauth_nonce="q9RgxiHx", oauth_timestamp="1334652625", oauth_version="1.0", oauth_signature=""
待签名数据	POST&http%3A%2F%2Fopen.dx.com%2Foauth%2Frequest-token&oauth_consumer_key%3DGmDIWcLRiPzvbUoZRAbWiA%26oauth_nonce%3Dq9RgxiHx%26oauth_signature_method%3DHMAC-SHA1%26oauth_timestamp%3D1334652625%26oauth_version%3D1.0

在生成了待签名数据后，则需要选择相应的签名算法对签名数据进行签名。签名结果计算出来后，需要被赋予给原请求的 oauth_signature 参数。但签名过程不会对请求的其它的参数造成任何变化。

就签名算法而言，一般我们会使用 HMAC-SHA1[15] 与 RSA-SHA1[16] 与 PLAINTEXT 这三种方式。

选择 HMAC-SHA1 算法时，该算法所需要的 key 由客户端证书的共享私钥与令牌证书的共享私钥经过 URL 编码后通过“&”拼接而成。使用该 key 对待签名数据进行一次计算后，即得出签名。服务提供方是知道客户端证书及令牌证书的密钥的，在使用同样的方法计算出签名后与再请求中的签名进行比对来确认请求

是否有效。

若选择使用 RSA-SHA1 进行签名的话，需要确保客户端所使用的证书的公钥以某种形式在进行交互前提供给了服务提供方，以使得服务提供方能够完成对签名的验证。

确切地讲，PLAINTEXT 不算是签名算法，使用 PLAINTEXT 的话，客户端只需要简单地将客户端证书的共享私钥与令牌证书的共享私钥经过 URL 编码后通过“&”拼接而成即组成了整个请求的签名。我们认识到，请求参数的变化并不能直接导致签名的变化，即，签名与请求参数无关。故而，此方法并不能帮忙服务提供方验证请求是否被篡改过。有鉴于此，协议规定，在使用 PLAINTEXT 的情况下，整个请求必须在一个安全的通道中进行传输。例如使用 HTTPS 就可以确保请求数据在传输过程中不会被篡改。

2.1.2 OAuth 2.0 及其工作流程

在 OAuth 1.0 的基础上，互联网从业者经过详细地研究与仔细地思考，并结合相关实践经历，推出了一个在工作流程上更为简洁、授权方式更加灵活与多样化以及权限分割与控制做得更加细致的新协议，这便是 OAuth 2.0[17]。

① 协议概览

在 OAuth 2.0 协议中，原服务提供方这一角色被细分成了授权服务提供方（Authorization Server）与资源服务提供方（Resource Server）。即此协议中的角色分别是资源拥有者（Resource Owner）、客户端（Client）、授权服务提供方与资源服务提供方。同时协议对客户端的类别也进行了细致的划分，包括普通网络应用（网站）、基于用户代理程序的应用（如 flash 应用、基于浏览器的插件应用等）以及本地应用（如移动终端应用、桌面应用等）。

在 OAuth 1.0 中，客户端向服务提供方发起的任何请求，都得进行签名。而我们知道，只要通过安全的传输通道，请求数据在传输过程中就可以避免被篡改。因此，为了使客户端与服务提供方的交互更加简单，OAuth 2.0 协议规定，协议内部的请求通道都必须使用 TLS（Transport Layer Security）[18]。

OAuth 2.0 中对于授权阶段也进行了简化，它取消 1.0 中临时令牌的请求，而是让应用让资源拥有都对应用进行直接授权。授权的方式也相对比较灵活。共有如下四种：通过使用授权码（Authorization Code）走服务端流程的方式、通过走客户端流程进行直接授予（Implicit）的方式、通过使用资源拥有者登录凭证（Resource Owner Password Credentials）的以获得授权的方式以及通过使用客户端凭证（Client Credential）以获得授权的方式[19]。客户端可以自由选择适用于其自身环境的最佳授权方式以获得访问令牌证书。

对于权限的分割与控制，OAuth 2.0 细化到了每客户端、每用户这个粒度，而

不像早期版本中的只能按每客户端的粗粒度来控制。同时也为访问令牌添加了有效期限，从而结束了早期版本中的一旦授予，往往终身有效的情况。这对访问令牌被非法获取后，对资源进行的非法操作起到了一定的限制作用。当然，当令牌失效后，新增的刷新令牌流程可以用来帮忙获取新的令牌。

OAuth 2.0 协议也需要经过准备阶段、授权阶段与资源调用这三个阶段。

② 准备阶段

与 1.0 版本类似，客户端必须先通过某种方式在授权服务提供方进行注册以获取到相应的客户端证书。与 1.0 版本不同的是，在注册时客户端必须提供其所属的类别，而对于权限，前面我们提到，2.0 版本细化到了每客户端、每用户，故而此时客户端不需要进行权限的指定。权限的指定将在授权过程中进行。

③ 授权阶段

OAuth 2.0 协议提供了多样化的授权方式，每种授权方式适应于的客户端场景也不尽相同。授权服务提供方需要有统一的授权服务地址（Authorization Endpoint）来提供相应的授权服务，同时还需要提供统一的令牌服务地址（Token Endpoint）以进行令牌的申请服务。

1) 服务端流程

在此流程中，客户端需要有自己的重定向回调地址（Redirection Endpoint），同时，在客户端注册时，也需要在向授权服务提供方注册客户端使用的 URL 地址。故而，通过服务端流程进行授权的方式适用于有 Web 服务端的应用，如网站。

如图 2.3 所示，在用户触发授权流程后，客户端无须事先与授权服务提供方进行交互，而直接引导用户重定向至授权服务地址。在此过程中，客户端标识、请求的响应类型、待授予的权限、客户端状态标识以及客户端回调服务地址都必须以 Query String 的方式附加到授权服务地址后面。从表 2.9 中我们可以看到，对于客户端所需要的权限，通过使用服务提供方指定的权限代码，应用待申请的权限被包含在了授权过程中。而资源拥有者在使用客户端过程中，通过使用客户端的不同功能，可以逐步地让客户端进行相应权限的申请。从而使得 OAuth 2.0 对权限的控制达到了每客户端、每用户的粒度。

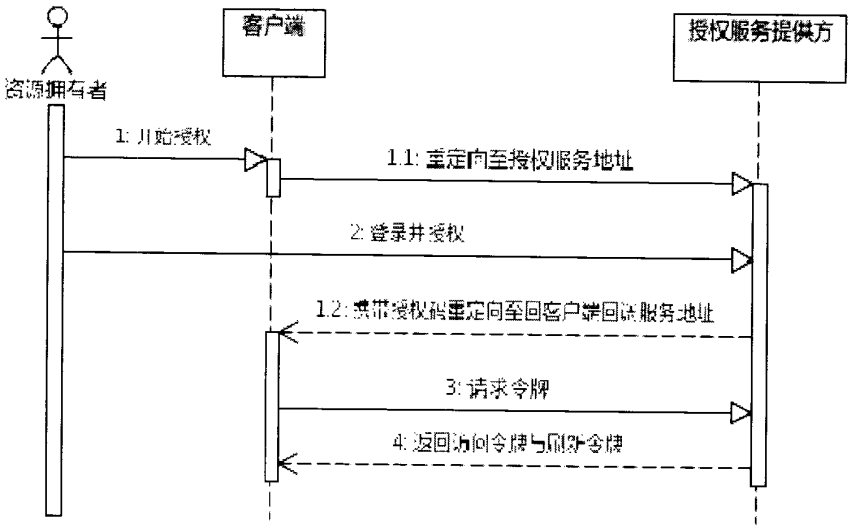

图 2.3 OAuth 2.0 服务端流程授权图

Figure 2.3 OAuth 2.0 server-side working flow

表 2.9 OAuth 2.0 服务端流程授权服务请求参数表

Table 2.9 OAuth 2.0 authorization request parameters in server-side flow

请求参数	是否必须	备注
response_type	Y	授权请求响应类型，必须为“code”
client_id	Y	客户端标识
redirect_uri	N	客户端回调服务地址，若为空，则使用客户端注册提供的客户端运行地址
scope	N	待授予的权限。授权服务提供方提供相应的权限代码，多个权限代码之间使用“,”隔开，留空则表示默认权限
state	N	客户端状态标识

授权服务提供方在接收到授权服务请求后，通过引导用户登录与授权，并在验证了回调服务地址后，生成一个全局唯一的授权码，并将其与客户端状态标识一起附加到回调服务地址的 Query String[20]部分并引导用户回到客户端。具体参数如表 2.10。

表 2.10 OAuth 2.0 服务端流程授权服务响应参数表

Table 2.10 OAuth 2.0 authorization response parameters in server-side flow

返回参数	是否必须	备注
code	Y	授权码
state	N	客户端状态标识。必须按照请求中的 state 值原样返回

在授权服务过程中，若用户拒绝授权，或是在发生其它异常的情况下，授权服务提供方返回给客户端回调服务的参数则有所变化，详细如表 2.11 所示。

表 2.11 OAuth 2.0 服务端流程授权服务异常响应参数表

Table 2.11 OAuth 2.0 response parameters for exception from server side

返回参数	是否必须	备注
error	Y	错误代码，包括 invalid_request、unauthorized_client、access_denied、unsupported_response_type 、 invalid_scope 、 server_error 与 temporarily_unavailable
error_description	N	错误信息描述
error_uri	N	错误信息的详细描述 URL，一般为授权服务提供的帮助文档地址
state	N	客户端状态标识。

客户端回调服务在接收到请求后，需要判断授权服务返回是否是异常情况下的返回结果，若是，则需要停止整个授权流程。若授权服务提供方返回的是正常情况下用户授权成功的结果，客户端则在获取到授权码后，需要将服务提供方返回的客户端状态标识后与本地标识进行一次校对。校对成功后，则可以凭借着授权码，向服务提供方申请访问令牌与刷新令牌。具体参数如表 2.12 所示。

表 2.12 OAuth 2.0 服务端流程请求令牌参数表

Table 2.12 OAuth 2.0 access token request parameters in server-side flow

请求参数	是否必须	备注
grant_type	Y	授予类型，必须为“authorization_code”
code	Y	从授权服务提供方获取到的授权码
redirect_uri	N	客户端回调服务地址，必须与请求授权码时的回调地址保持一致
client_id	Y	客户端标识
client_secret	Y	客户端私钥

授权服务提供方在接收到申请令牌的请求后，需要通过如下步骤完成整个验证过程：

1）客户端认证：即对客户端标识与客户端私钥进行认证；

2）授权码验证：确保授权码存在且未被使用过；

3）其它验证：确保授权码对应的客户端与本次请求为同一客户端，同时确保客户端回调地址与请求授权码时的回调地址为同一地址。

在校验通过之后，授权服务提供方需要将授权码标记为已经使用，并在创建访问令牌与更新令牌后，按表 2.13 所示返回。

表 2.13 OAuth 2.0 服务端流程请求令牌响应参数表

Table 2.13 OAuth 2.0 access token response parameters in server-side flow

返回参数	是否必须	备注
access_token	Y	访问令牌
token_type	Y	令牌类型
expires_in	Y	有效时间（秒）
refresh_token	Y	刷新令牌

由于使用 TLS 以确保了请求通道的安全性，OAuth 2.0 中的令牌不再需要生成相应的密钥。同时，与早期版本有所差异的是，OAuth 2.0 内服务提供方向客户端返回的数据优先采用了 JSON 格式。表 2.14 所示即为客户端流程中申请令牌的响应数据样例。

表 2.14 OAuth 2.0 服务提供方返回数据样例

Table 2.14 OAuth 2.0 authorization server side response data sample

```
{
    "access_token" : "ya29.AHES6ZSaMAWtm1C4tRbwvCQ2SnSoinSUmWfyaceFWvanZRE",
    "token_type" : "Bearer",
    "expires_in" : 3600,
    "refresh_token" : "1/66MuZf_kC97O4zLGTxcGBsbh8rPZ05vQOZoJK0irM14"
}
```

客户端在获取到访问令牌与刷新令牌后，便可将之存储起来，以供后续的资源调用相关操作的进行。至此，服务端流程完成。

2) 客户端流程

相应服务端流程，客户端更加简洁与快速。它比较适用于运行于缺乏后 Web 服务器支持，运行于用户代理程序内部的应用，如运行于页面内的 JavaScript 程序或者 Flash 应用等。

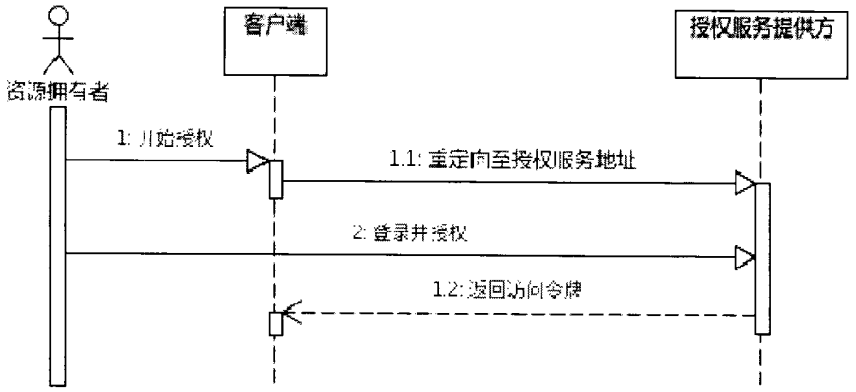

图 2.4 OAuth 2.0 客户端流程授权图

Figure 2.4 OAuth 2.0 client-side working flow

与使用授权码的服务端流程相比，客户端流程的最大差别在于在用户授权后，授权服务提供方直接将令牌信息返回给了客户端。如表 2.15 所示，该请求参数与服务端流程授权服务请求参数的唯一区别就是 response_type 的参数值由 code 改成了 token。

表 2.15 OAuth 2.0 客户端流程授权服务请求参数表

Table 2.15 OAuth 2.0 authorization request parameters in client-side flow

请求参数	是否必须	备注
response_type	Y	授权请求响应类型，必须为“token”
client_id	Y	客户端标识
redirect_uri	N	客户端回调服务地址，一般为当前页面地址。若为空，则使用客户端注册提供的客户端运行地址
scope	N	待授予的权限。
state	N	客户端状态标识

我们知道，URI 中的 Fragment[20]部分，即“#”后面的部分，是专门为客户端设计的，只有像 JavaScript 这类的客户端语言才有可能获取得到，而 Web 服务器程序中是无法获取的。基于这点，客户端流程中授权服务的响应方式是直接将访

问令牌以明文形式直接附加在客户端回调地址的 Fragment 部分，再引导用户至此回调地址。由于使用了这种明文方式的令牌传送，协议强调，需要确保客户端回调地址与客户端在注册时提供的运行地址的一致性。同时，在此过程中，不会返回刷新令牌，而只返回访问令牌（access_token）、有效期限（expires_in）、令牌类型（token_type）与客户端状态（state）等。示例数据如表 2.16 所示。

表 2.16 OAuth 2.0 客户端流程授权返回数据样例

Table 2.16 OAuth 2.0 authorization response data sample in client-side flow

http://example.com/#access_token=ya29.AHES6ZSaMAWtm1C4tRbwvCQ2SnSoinSUmWfyaceF WvanZRE&token_type=Bearer&expires_in=3600&state=xyz

3) 资源拥有者登录凭证流程

使用资源拥有者的登录凭证以获取授权方式是一种比较特殊方式，由于客户端能够接触到资源拥有者的登录凭证，故而服务提供方需要对客户端要持有足够的信任，才能允许其使用此流程来进行授权的获取。

如图 2.5 所示，在获得登录凭证后，客户端便可以直接向授权服务提供方的令牌服务地址（Token Endpoint）发起请求以申请令牌。授权服务提供方在接收到请求并完成验证后，就可以将令牌返回给客户端。

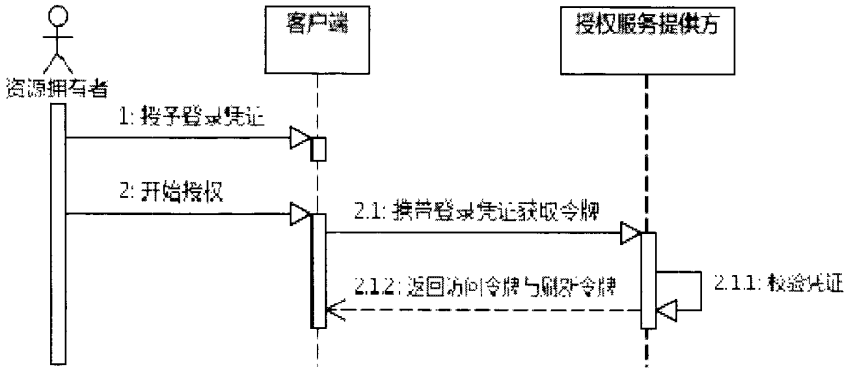

图 2.5 OAuth 2.0 使用资源拥有者的登录凭证获取授权流程图

Figure 2.5OAuth 2.0 user credentialauthorization working flow

在此请求过程，请求参数包括授权类型、客户端标识、客户端私钥、用户的登录名、登录密码以及请求的权限等等。详见表 2.17。

表 2.17 OAuth 2.0 使用资源拥有者登录凭证进行请求参数表

Table 2.17 OAuth 2.0 access token request parameters by using user credential

请求参数	是否必须	备注
grant_type	Y	授予类型，必须为“password”
client_id	Y	客户端标识
client_secret	Y	客户端私钥
username	Y	用户登录名
password	Y	用户登录密码
scope	N	待授予的权限

授权服务提供方需要进行验证客户端的标识及其私钥的合法性，再验证用户登录名与登录密码的合法性。在通过验证后，服务提供方可以认为用户同意授予权限权限给应用，而无须再经过用户的再次确认，然后生成相应的令牌并返回。返回的数据格式与服务端流程中令牌申请服务的响应数据格式相同，即为包含access_token、token_type、expires_in、refresh_token这四个属性的数据，详见表2.13。

4) 客户端凭证流程

客户端凭证流程即是通过使用客户端凭证，来获取访问令牌的方式。一般在资源拥有者即是客户端的时候，可以通过此流程获得访问令牌，而后再获取相应资源。

授权服务提供方的令牌申请服务在接收到客户端 POST 过来的客户端标识与私钥后，只要验证通过，即可以进行授权。请求参数详见表 2.18。令牌响应数据格式与服务端流程的令牌响应数据格式相同。

表 2.18 OAuth 2.0 客户端凭证流程授权请求参数表

Table 2.18 OAuth 2.0 access token request parameters by using client credential

请求参数	是否必须	备注
grant_type	Y	授予类型，必须为“client_credentials”
client_id	Y	客户端标识
client_secret	Y	客户端私钥
scope	N	待授予的权限

④ 刷新访问令牌

在 OAuth 2.0 中，访问令牌具有有效期限。在令牌失效或者即将失效的时候，客户端可以根据刷新令牌，请求授权服务提供方的令牌服务，以更新访问令牌。

请求参数如所示。

表 2.19 OAuth 2.0 刷新访问令牌请求参数表

Table 2.19 OAuth 2.0 refresh token request parameters

请求参数	是否必须	备注
grant_type	Y	授予类型，必须为“refresh_token”
client_id	Y	客户端标识
client_secret	Y	客户端私钥
refresh_token	Y	刷新令牌
scope	N	待授予的权限

令牌申请服务在验证了客户端标识及私钥的有效性后，再验证刷新令牌的有效性后，创建新的访问令牌并返回。

⑤ 资源调用阶段

OAuth 2.0 的资源调用请求比 1.0 版本也有所精简，除了免去签名外，也省去了客户端标识的传递，这就使得除了资源服务提供方明确指定的参数外，客户端只需要额外地附加访问令牌（access_token）即可发送请求。资源服务提供方，通过访问令牌，即可获知请求的客户端与指定的用户。只要访问令牌没有过期，而且相应的授权也没问题的话，资源服务提供方就可以自由地地提供相应的服务以实现资源跨界使用与共享。

2.2 NoSQL 与 MongoDB

今时今日，传统的关系型数据库在应对社会化网络应用，特别是超大规模和高并发类型的网络应用已经显得力不从心，并暴露了很多难以克服的问题，主要表现为以下几点：

第一，关系型数据库对高并发读写的支持越来越艰难。当今的社会化网络应用中，用户个性化的程度相当的高，而应用一般也需要根据用户的个性化信息实时地提供动态信息而无法使用将相关信息做静态化处理。故而，应用需要实时地对数据库进行读写操作，导致数据库并发数量会相当地高，往往能达到每秒上万次的请求。而如此之高的请求频率下，尤其是在高频率的写操作下，硬盘 IO 往往都消耗了太多的时间，而导致整个应用的响应越来越慢而将用户的耐心消磨殆尽。

第二，关系型数据库难以达到对海量数据的高效率存储和访问的要求。社会化网络应用是面向整个社会大众的。而随着世界是平的这一概念的提出，每一个网络应用在其诞生之刻就不得不应对来自全球的几十亿的用户。像在 Facebook、

Twitter 这样的全球性的社会化网络中，每天都有上千万级别的用户数据的产生。对于传统的关系型数据库来说，不要说插入，仅仅是在如此海量的数据的数据中找出一条数据，其效率也不会是很乐观的。

第三，关系型数据库无法达到高可扩展性和高可用性的需求。由于海量数据的产生，数据库对于硬盘的需求也相当地高。往往都需要经常地为数据库增加新的硬件来实现数据层的负载均衡。而在传统关系型数据库中，对数据库进行升级与扩容往往会是一件痛苦的事情，有些数据库甚至需要通过进行停机维护来完成，这对于一个全球化的社会化应用来说，是无法承受的。

由于传统的关系型数据库存在的种种缺陷，近几年以来，用以解决这诸多问题的非关系型的数据库（NoSQL）[21]成了一个极其热门的新领域，而相应的数据库产品的发展非常迅速。主要可以归为以下几类：

1) 满足极高读写性能需求的 Kye-Value 数据库；
2) 满足海量存储需求和访问的面向文档的数据库；
3) 具有足高可扩展性和可用性的面向分布式计算的数据库。

MongoDB[22]就是一种分布式的面向文档的数据库。它主要实现保证海量数据存储的同时，具有良好的查询与扩展性能。在 MongoDB 中，所有数据都是以文档的形式存储，同时文档的格式也相当松散，从简单到各种复杂的数据，都可以统一以一个文档的形式存储在 MongoDB 中。由于 MongoDB 是分布式的，故而它在可扩展性上表现得也相当优越。

同时，MongoDB 有着功能强大的查询语法，能够满足各种常见数据存储与查询要求。它对于各种语言的驱动也支持得相当完善。包括官方驱动以及社区推出的非官方驱动等。

这一系列的特性，以及在社区的广泛影响，使得 MongoDB 成了当今应用得最广的非关系型数据库。

2.3 RESTful Web Services

REST（REpresentational State Transfer，表述性状态转移）是一种架构风格，是由 Roy Thomas Fielding 博士在他的论文《Architectural Styles and the Design of Network-based Software Architectures》[23]中提出的一个术语，也是今世界最成功的互联网超媒体分布式系统架构，并使得人们真正理解了 HTTP 协议本来面貌。

REST 认为，在网络中，任何事物都可以被抽象成资源，而每个资源对应着唯一的资源标识，这个资源标识便是它的 URI（Uniform Resource Identifier）。对于资源的操纵，必须通过通用的转换器接口（generic connector interface）来完成。由于 REST 是基于 HTTP 协议的，故而这些转换器接口就是 HTTP 协议中的 GET、POST、

PUT、DELETE、HEAD 与 OPTIONS 等[24]。同时，REST 认为，对于资源的任何操作，都不能导致资源标识的改变，而且所有的操作都是无状态的。

由于 REST 还原了 HTTP 的本来面貌，这使得它快速地被应用在了基于 HTTP 协议的网络服务中，这便是 RESTful Web Service。比起 SOAP 协议的一大堆复杂的 XML 描述，REST 来得更加直接与明确，故而也更加简练。这也就使得服务的客户端的受益不少。我们可以直接使用浏览器，或者 curl，wget 等工具直接向 RESTfull 的服务发起 HTTP 请求。这也大大降低了我们的开发、调试以及对系统运行状态的监控等工作的复杂度，也就相应地提高了效率并节省了成本。

2.4 JSON

JSON（JavaScript Object Notation）[25]是一种轻量级的数据交换格式。JSON 对象是一个无序的键值对的集合，非常易于人阅读和编写，同时也易于机器解析和生成。它以“{”（左括号）开始，“}”（右括号）结束。每个“键”之后跟一个“:”（冒号）；键值对之间使用“,”（逗号）分隔。例如使用如下字符即可以用来表示成立于 1929 年 10 月 12 日的重庆大学这一对象：

```
{“Name”: “Chongqing University”,“Founded”:{“Year”: 1929,“Month”: 10,“Day”: 12}}
```

由于不同于 XML 数据格式的复杂性，JSON 应用得相当的广泛，特别是在基于 HTTP 请求以进行数据传递的应用之中。

各语言对于 JSON 数据都提供了良好的支持。而一些现代的浏览器（如 Firefox，Google Chrome）甚至内置了对 JSON 格式数据的序列化与反序列化操作，以提高基于 JavaScript 语言的相关应用的速度与性能。

2.5 本章小结

本章节首先详细介绍了 OAuth 1.0 与 OAuth 2.0 协议的原理与工作流程，这对于解决我们项目与社会化网络中的资源跨界使用与共享、账户管理麻烦等问题提供了强劲而有力的技术保障。

然后我们对支撑高并发与海量数据的非关系型数据库 MongoDB 进行了相应的了解。这是使得当今社会化网络应用能够快速响应并服务于全社会的基础。

而后我们对如今世界最成功的互联网超媒体分布式系统架构，REST 风格架构，进行了相应的介绍。这为我们如何架构社会化网络应用服务，具有重大的指导意义。

最后，我们学习并了解了被广泛应用于当今网络应用之间进行数据传输的 JSON 格式。

3　基于OAuth协议的电商数据平台与应用的分析与设计

DX的数据平台是专门为所有愿意接入DX电商系统的应用提供数据支撑服务的，这些数据包括但不限于商品信息数据、客人的账户信息、订单信息、积分信息、优惠信息、博客、帖子等。无论这些应用是由公司其他部门设计与研发的，还是由第三方公司或个人热心研发的，都不能对数据平台的接口设计造成影响。DX数据平台在平等地对待这些应用的同时，也能够保持数据平台的标准性与精简性。同时，DX数据平台也不能对各种应用的实现语言与运行环境进行假定或者限制。

作为互联网标准的OAuth协议，在HTTP标准协议的基础上，确保了应用的运行环境与实现语言都不会是障碍。而该协议规定的统一的帮助用户授权给予应用的授权流程，就确保了数据平台无须过多地关心应用的开发商，而平等地为各种应用提供授权服务支持，以及为应用在获得授权后获取并且操纵指定用户的订单等私有资源数据等操作提供服务。

OAuth协议经过几年的发展，如今已经有1.0正式版与2.0的稳定成型版（第25个修改版本）。虽然这两个版本的名称都叫OAuth协议，但是OAuth 2.0协议实际上是一个全新的协议，它并未保留对OAuth 1.0协议的兼容性，故而在本章我们将分别对基于这两个版本的数据平台与应用进行相应的分析与设计工作。

3.1　基于OAuth 1.0的平台与应用的分析与设计

3.1.1　用例分析

通过第2章中对OAuth 1.0协议及其工作流程的相关了解，我们得到，在整个流程过程中，共有客户端、服务提供方与资源拥有者这三个参与者。

如图3.1所示，资源拥有者在整个应用中，需要参与账户管理与授权管理这两大功能。账户管理包括账户的注册、登录、登出以及重置密码等用例。授权管理包括同意授权给客户端、拒绝授权给客户端以及在授完权后，返回给客户端的授权这三个功能。同时，整个协议工作流程的开启，需要资源拥有者来触发。

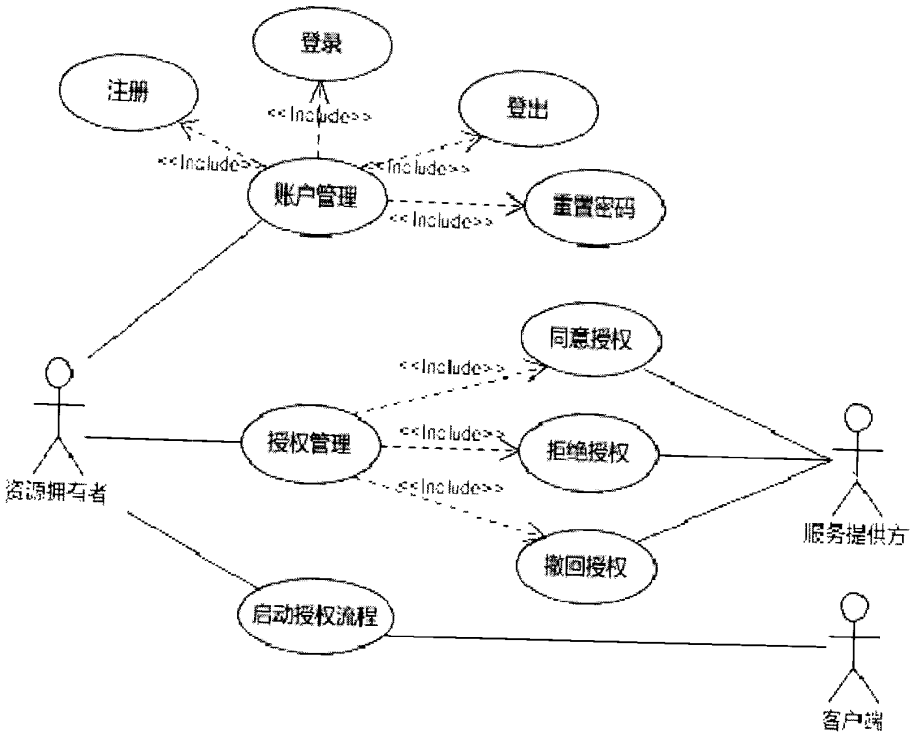

图 3.1 OAuth 1.0 中资源拥有者参与的用例图

Figure 3.1 OAuth 1.0 resource owner related use cases

在协议的核心的工作流程是在客户端与服务提供方这两个参与者之间完成的。如图 3.2 所示，在客户端与服务提供方参与的用户有客户端管理、申请临时请求令牌、申请返回令牌与资源的调用。客户端管理包括客户端的注册、注销与所需要的权限管理等子用例。对于临时请求令牌与访问令牌，均是客户端负责提出申请，服务提供方负责管理与颁发。

同时，从图 3.2 中我们还看到，客户端有义务引导资源拥有者到服务提供方进行授权。鉴于运行环境的差异，客户端引导授权的方式也有所不同。对于 B/S 结构的客户端应用，可以直接将页面跳转至服务提供方进行授权。而对于桌面或者移动客户端应用，可以选择调用操作系统内置的浏览器程序，打开服务授权地址从而引导授权，或者是通过应用内置的浏览器进行引导授权。一般来说，由于资源拥有者可能已经使用操作系统内的浏览器访问过服务提供方，从而可能保存了相应的会话信息，故而对于桌面与移动客户端来说，选择通过调用操作系统内置浏览器以引导授权是一种比较推荐的做法。

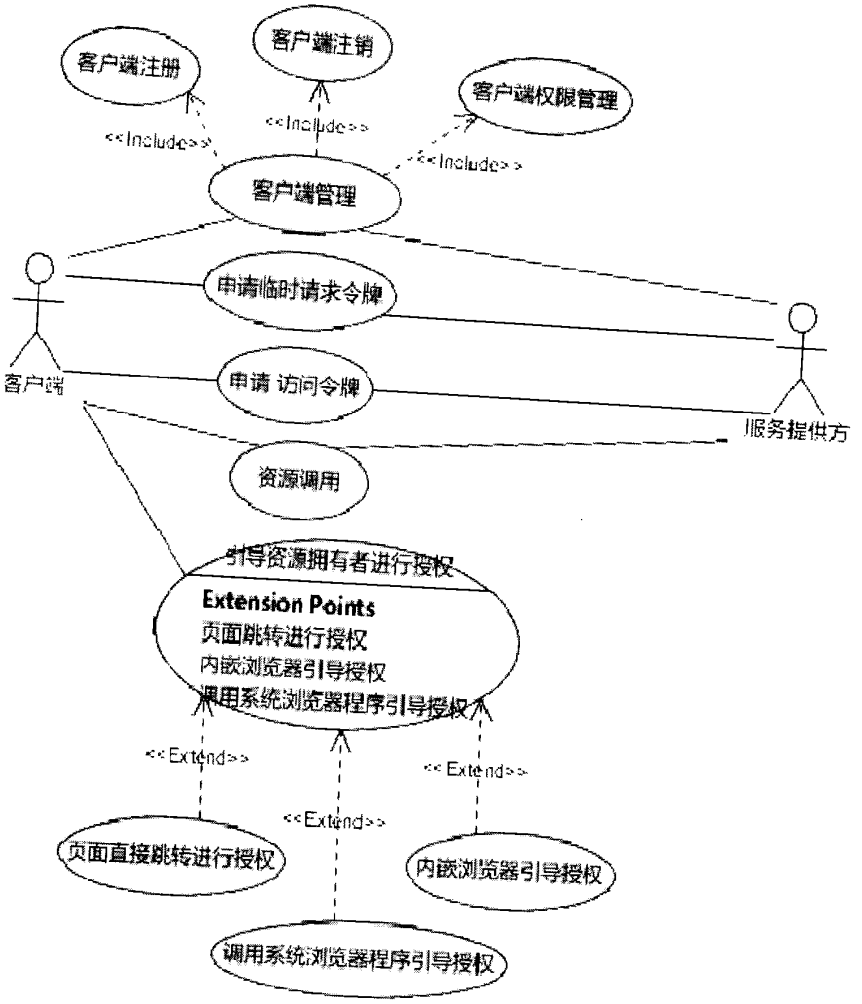

图 3.2 OAuth 1.0 中客户端与服务提供方参与的用例图

Figure 3.2 OAuth 1.0 client and service provider related use cases

3.1.2 系统设计

鉴于协议中的服务提供方与客户端是两个相互独立的应用，在系统设计时，我们将分别对这两个应用进行设计。

① 服务提供方设计

1) 类图设计

通过对协议中的服务提供方所涉及用例的名词分析，我们对其进行抽象后，得出了如图 3.3 所示的服务提供方类图。

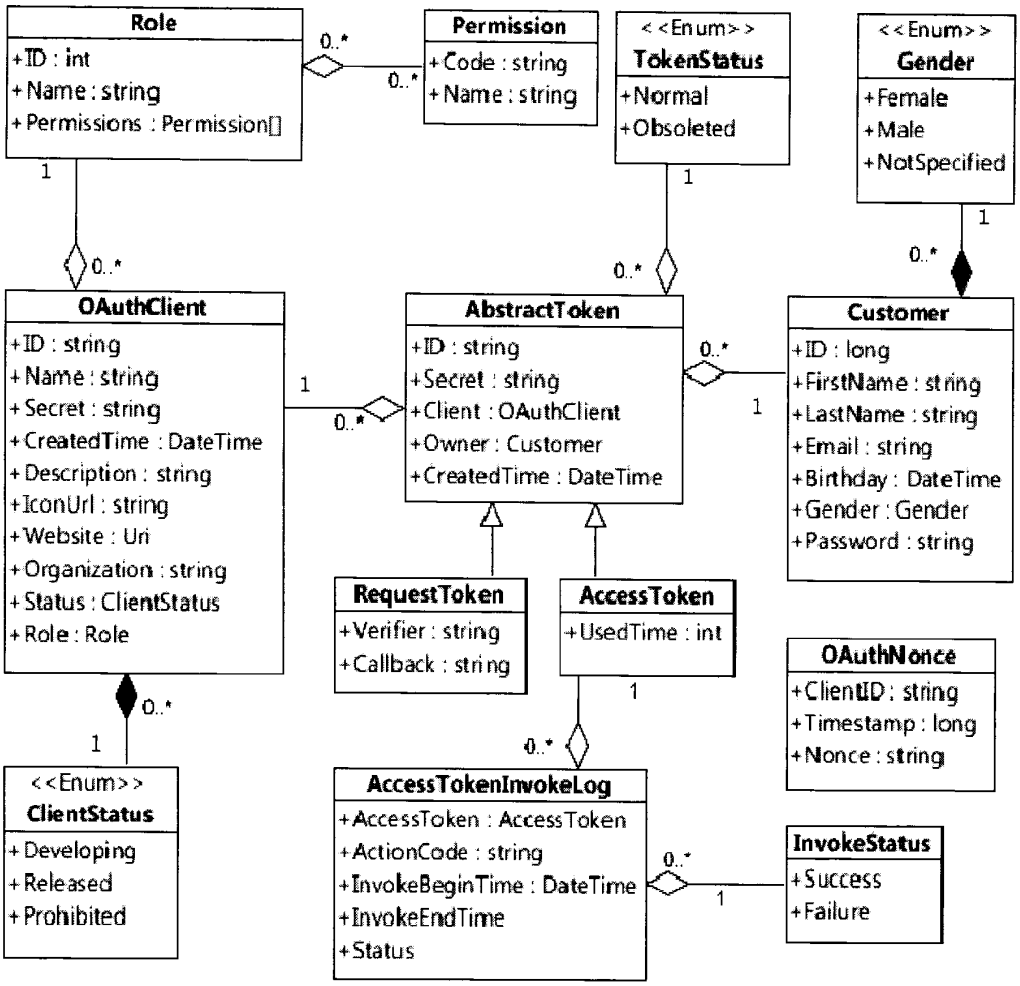

图 3.3 OAuth 1.0 服务提供方类图

Figure 3.3 OAuth 1.0 class diagrams for service provider

在服务提供方，首先需要有客户端这一抽象，客户端具有着 ID（也可以作为 Client Key）、名称、共享密钥、描述、发布的网站、应用图标、创建时间、创建组织以及状态等信息。客户端的状态可以分为研发中、已发布与被禁用等。

每个客户端对应着一个角色，每个角色被赋予了零或多个权限。权限由代码权限唯一标识的权限代码与其名称组成。每个角色可以被指定给多个不同的客户端。角色与权限也是多对多的关系。

而对于资源拥有者，他拥有着邮箱、密码、姓名、性别等属性。

对于令牌而言，临时请求令牌与访问令牌都有着标识、共享密钥、所属的客户端、所属资源拥有都、创建时间以及令牌状态等属性，故而可以将这些属性放至抽象类 AbstractToken 中。一个令牌只能被一个客户端与一个资源拥有者使用。而一个客户端与资源拥有都可能需要使用多个令牌。令牌的状态分为正常与过期失

效这两种。而对于临时请求令牌来说，它有着额外的校验码与客户端回调地址这两个属性。访问令牌需要记录它的使用次数与使用日志。对于访问令牌的每一条使用日志，我们需要分别记录它的请求开始与结束时间，以及请求的操作代码与请求结果成功与否等。

2) 系统关键架构设计

为了使得系统具有良好的可维护性与扩展性，系统采用模块化设计，并进行多层分离。系统详细的系统架构见图 3.4。

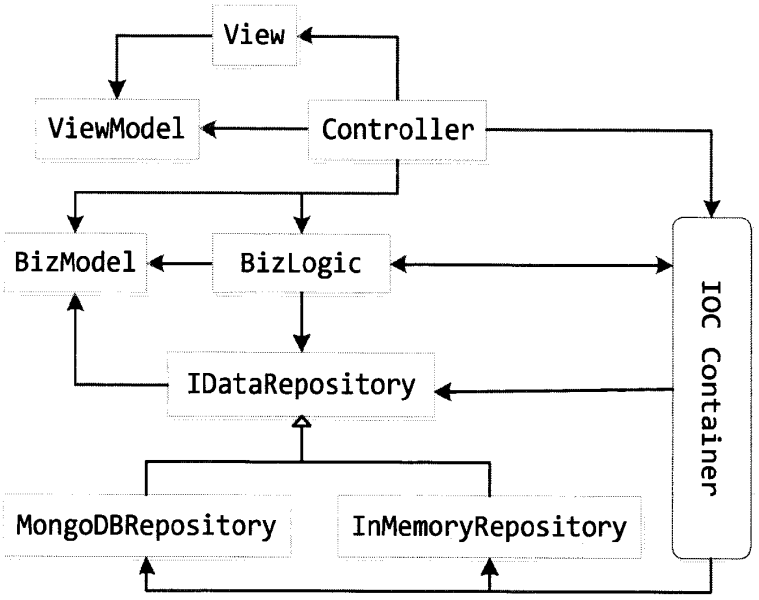

图 3.4 应用程序系统架构图

Figure 3.4 Application architecture

在系统架构图中，自底向上分别是：

1）数据访问层：提供具体的数据存储、访问、修改与删除的功能。数据可以存储在诸如 MongoDB，SqlServer 等数据库中，也可以是驻留在服务器内存或者诸如 Memcached 之类的第三方缓存中；

2）数据访问抽象层：定义数据访问层的接口原型与规范；

3）业务逻辑层：完成系统内部的具体业务逻辑的实现，以及完成对数据存取的相关操作。业务逻辑层与数据访问层不发生直接的关联关系，而只是依赖于数据访问抽象层。在程序的运行过程中，业务逻辑层通过 IOC（Inversion Of Control）容器，获取到具体的数据访问对象实例。需要在应用程序启动时，通过读取配置文件或者其它方法，将具体的数据访问层在 IOC 容器进行注册。

4）页面展示层：采用 MVC 模式，控制器（Controller）负责调用相应的业务逻辑，生成相应的视图模型（ViewModel），而后将视图模型传递给指定的视图，并由视图负责完成页面的渲染。

3) 基于 AOP 的请求验证设计

AOP（Aspect-Oriented Programming，面向切面编程）[26]是消除代码重复的一种方法。它可以自由地将第三方代码注入到程序的运行过程中。

在 OAuth 1.0 协议中，我们认识到，在客户端直接向服务提供方发起的任意请求，我们都需要进行客户端验证、请求重放验证以及签名验证等操作。为了降低编码的复杂性与提高系统的灵活性，这些验证将通过 AOP 注入来实现。

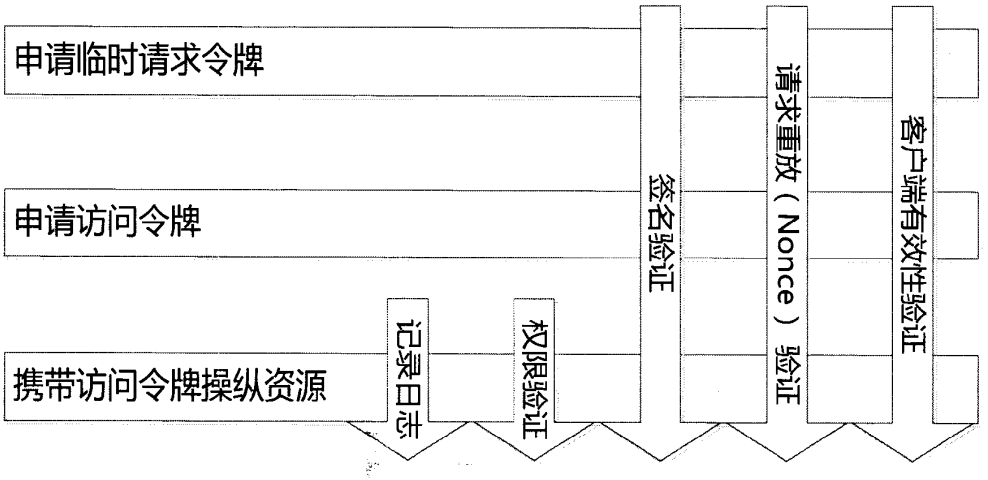

图 3.5 OAuth 1.0 服务提供方基于 AOP 的客户端请求验证

Figure 3.5 OAuth 1.0 AOP style for request verifications in service provider

如图 3.5 所示，对于客户端发起的申请临时请求令牌、申请访问令牌与后续的操纵资源等操作，通过 AOP 的方式，分别将客户端有效性验证、请求重放验证及签名验证等逻辑注入到请求的执行过程中。同时，对于资源的操作请求，权限的验证与日志的记录也以 AOP 的方式注入至其执行过程。

② 客户端设计

基于 OAuth 1.0 协议的客户端，需要关注临时请求令牌、访问令牌以及相应的服务器设置等。抽象后得出如图 3.6 所示类图。

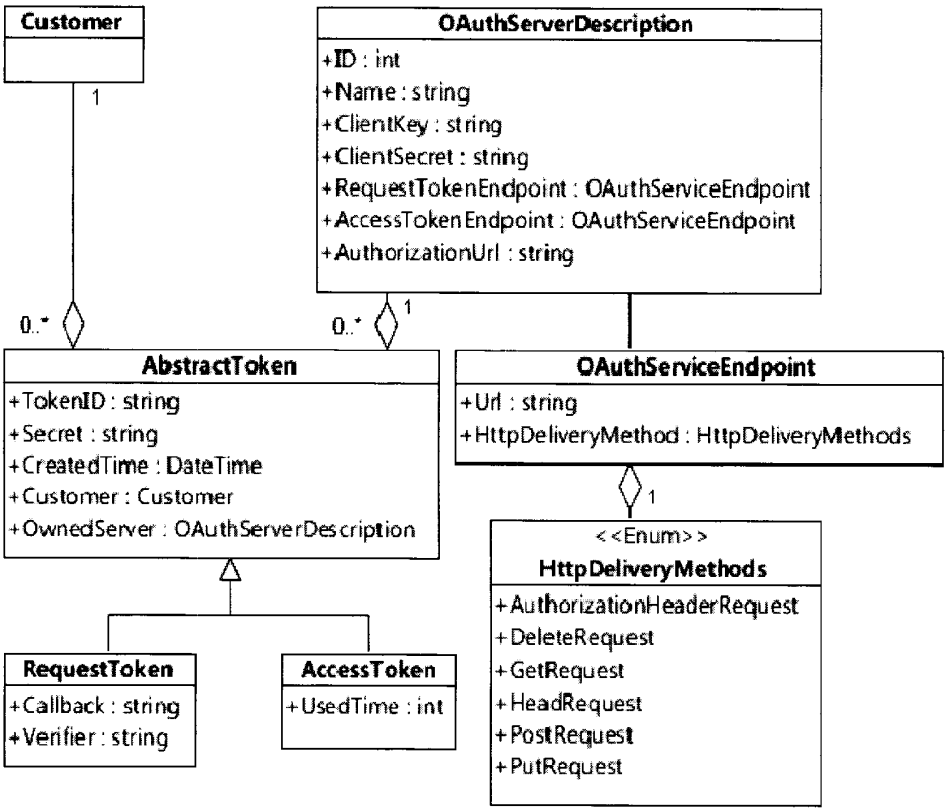

图 3.6 OAuth 1.0 客户端类图

Figure 3.6 OAuth 1.0 class diagrams for client

在客户端中，需要知道服务提供方以及相应的配置。服务提供方的抽象 OAuthServerDescription 包括 ID、名称、颁发给客户端的标识与共享密钥、临时请求令牌服务地址、访问令牌服务地址与授权服务地址。对于临时令牌服务与访问令牌服务地址，客户端还需要知道服务提供方支持使用何种方式发起 HTTP 请求，故而将这个 URL 地址与支持的 HTTP 请求方式，抽象成 OAuthServiceEndpoint 类。而具体的 Http 请求方式则为枚举 HttpDeliveryMethods，有代表将参数放至 HTTP header 的 Authorization 部分的 AuthorizationHeaderRequest 方式、将参数附加 URL 的 Query String 部分的 GetRequest 方式与使用 HTTP POST 方式的 PostRequest 方式等。

对于令牌而言，同样可以抽象出 AbstractToken。令牌具有标识、共享密钥、创建时间，所属的客户端用户以及颁发的服务提供方等属性。临时请求令牌与访问令牌从 AbstractToken 中继承出来，再附加上各自的特有属性。如请求令牌具有回调地址与校验码属性，而访问令牌可以有使用次数等属性。

对于客户端具体系统架构，视客户端类型、运行环境、及其实现语言而定。这

不是本课题讨论的重点。

3.2 基于 OAuth 2.0 的平台与应用的分析与设计

3.2.1 用例分析

在 OAuth 2.0 中，参与者由 1.0 版本中的三个变成了四个，即资源拥有者、客户端、授权服务提供方与资源服务提供方。在这四个参与者之间，除了资源拥有者不会直接与资源服务提供方发生直接关联外，其它的两两之间均会涉及到交互操作。

对于资源拥有者参与的用例而言，与 1.0 版本没有太多的变化，同样需要参与账户管理与授权管理这两大功能。账户管理依然包含账户的注册、登录、登出以及重置密码等功能点，虽然这些不是协议的主要内容，但也是协议能够正常动作的必要保障。授权管理也依然包含同意授权、拒绝授权与返回授权这几个功能点。唯一的变化是，授权管理的另一个参与都变成了授权服务提供方。同样，协议授权流程的开启也需要资源拥有者来完成。

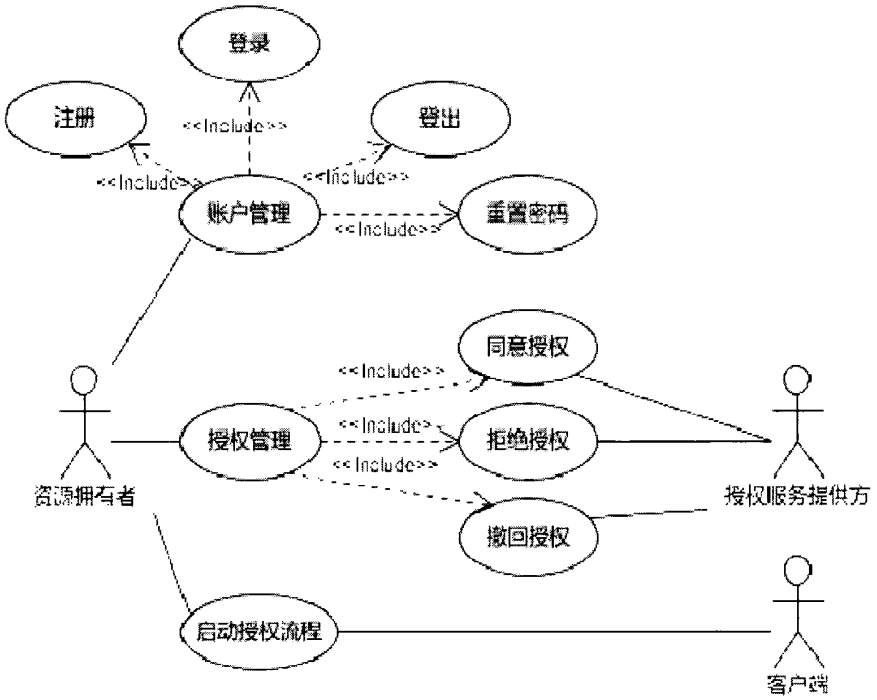

图 3.7OAuth 2.0 资源拥有者参与的用例图

Figure 3.7 OAuth 2.0 resource owner related use cases

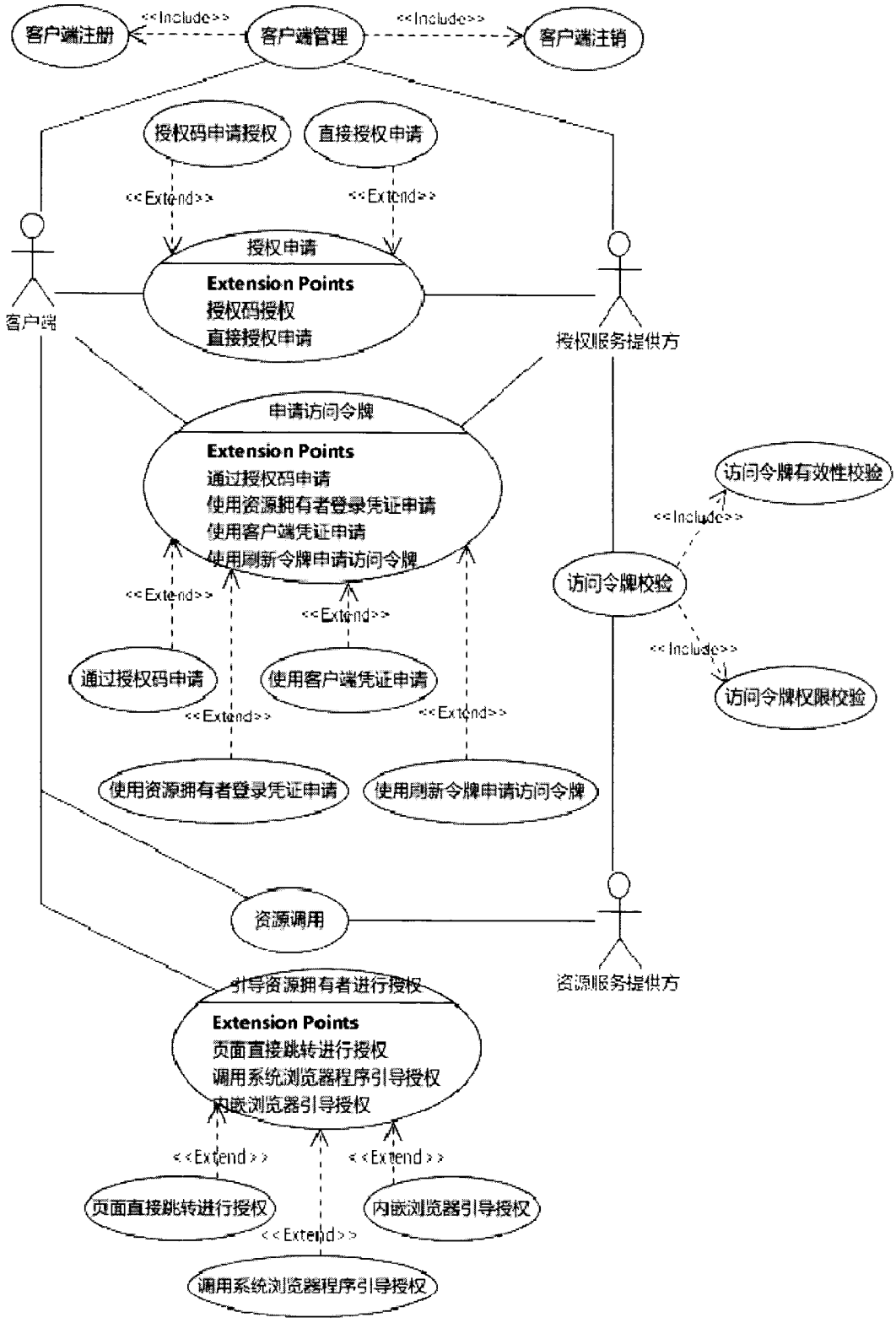

图 3.8 OAuth 2.0 客户端与服务提供方参与的用例图

Figure 3.8 OAuth 2.0 client and server related use cases

由于OAuth 2.0协议的简洁性与灵活性，作为协议工作流程的主要参与者，客户端与授权服务提供方之间参与的用例，相比之间版本，发生了很大的变化。

如图3.8所示，客户端依然需要在服务提供方进行客户端注册与注销等。而由于OAuth 2.0对权限的控制得更加细致而做到了每客户端、每用户，故而在客户端管理中无须对客户端所需要的权限进行管理。

客户端需要向服务提供方进行用户授权申请，而申请授权的方式包括使用授权码与直接申请访问令牌的方式。这分别是OAuth 2.0的授权流程中的服务端流程与客户端端流程中的一部分。

同时，客户端需要向服务提供方发起访问令牌申请。根据协议中规定，申请访问令牌的方式共包括：使用授权码申请、使用资源拥有者的登录凭证申请、使用客户端证书进行申请以及使用刷新令牌进行新访问令牌的申请这四种方式。

在获取到了访问令牌后，客户端可以向资源服务提供方发起资源的调用功能，资源服务提供方需要请求授权服务提供方帮忙校验客户端的访问令牌的合法性以及相应的权限等。

当然，与1.0版本中相同，客户端依然需要能够引导用户至授权服务提供方进行授权，具体的引导方式也没有任何变化，这里不再详细赘述。

3.2.2 系统设计

① 授权服务提供方设计

1) 类图设计

经过对OAuth 2.0的分析，得出授权服务提供方的类图如图3.9。

客户端对象（OAuthClient）依然有ID、私钥、名称、描述、客户端状态等信息。与1.0不同的是，客户端增加了回调服务地址与客户端类型这两个属性。客户端类型可分为桌面应用类型（PC 机桌面应用与移动终端应用）、网站以及基于浏览器的应用（JS/Flash应用、浏览器插件应用等）。

对于授权请求（AuthorizeRequest），它有授权码、所属客户端、客户端重定向URL、客户端状态码、请求的权限、所属的资源拥有者、是否过期与响应类型等属性。响应类型包括使用授权码的Code响应方式与直接返回访问令牌的Token方式。

访问令牌（AccessToke）拥有 ID、所属客户端、被颁发给的资源拥有者、创建时间，失效时间、对应的更新令牌、拥有的权限、被使用次数与令牌类型等属性。当然，对于访问令牌的使用记录，也可以进行相应的记录。具体的属性与1.0一致，这里也不再赘述。

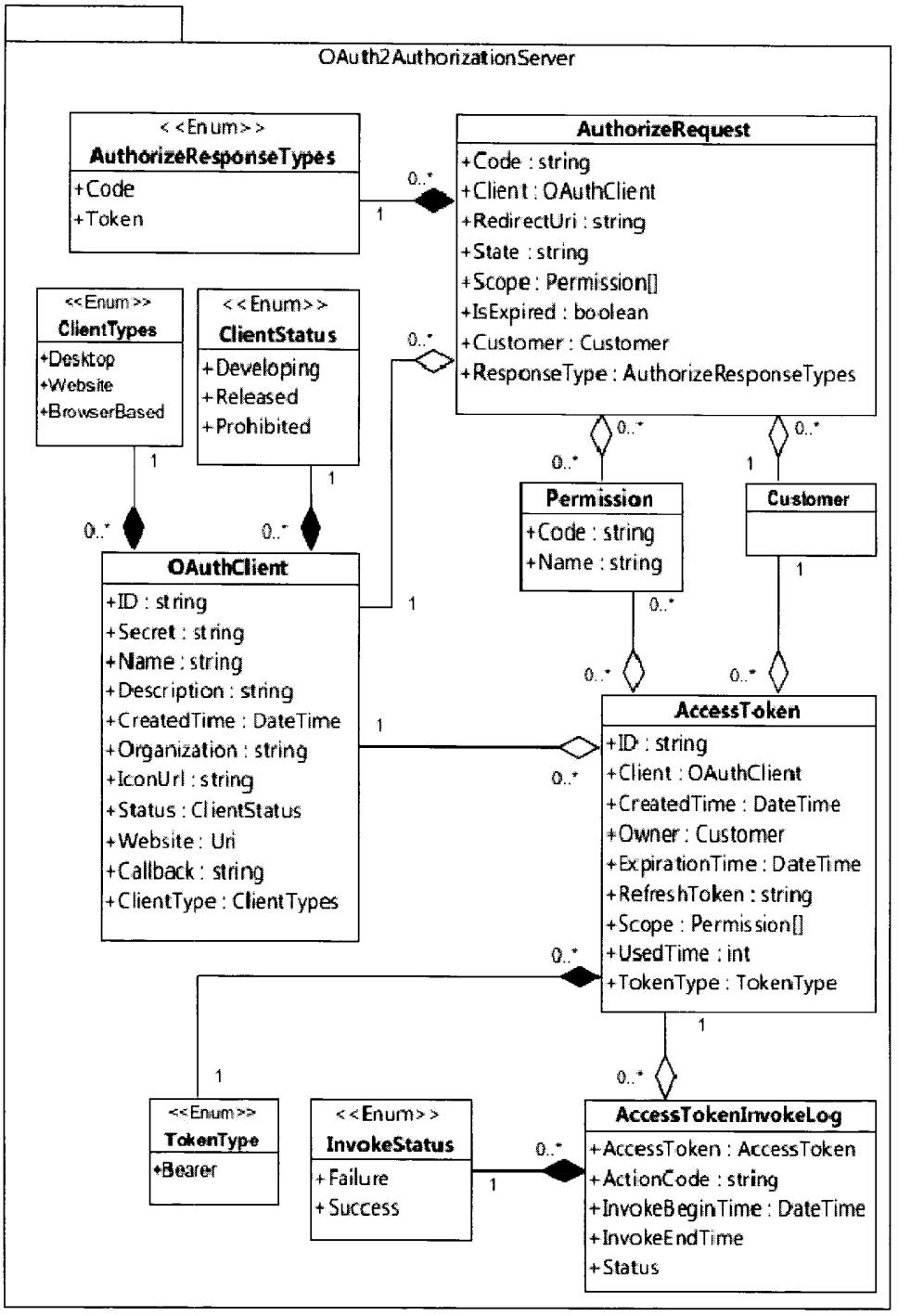

图 3.9 OAuth 2.0 授权服务提供方类图

Figure 3.9 OAuth 2.0 authorization server class diagrams

2)　系统关键架构设计

对于服务提供方，系统架构依然可以采用 3.1.22)所述进行，这里不再赘述。

3) 基于 IOC 的访问令牌请求处理设计

IOC（Inversion of Control）[27]，也叫控制反转，是一种面向接口编程的方式，即依赖于接口，而不是一个具体的类。在程序的实际运行过程中，接口的具体实现类的实例，通过 IOC 框架，实时注入至持有接口引用的实例中去，这个注入过程叫做依赖注入。

为了适应 OAuth 2.0 的申请访问令牌的灵活与多样性，我们可以将各种请求的具体响应方式通过 IOC 依赖注入的方式，来灵活地处理请求并做出响应。如图 3.10 所示，使用授权码、使用刷新令牌、使用客户端证书以及使用资源拥有者登录凭证的响应处理方式均实现了 IAccessTokenRequestHandler 接口。IOC 容器内装载着针对此接口的各种实现。而作为主程序的 Controller 持有对接口与 IOC 窗口的引用。在请求到达时，Controller 通过 IOC 容器，获得具体的响应处理实例，从而完成对请求响应。

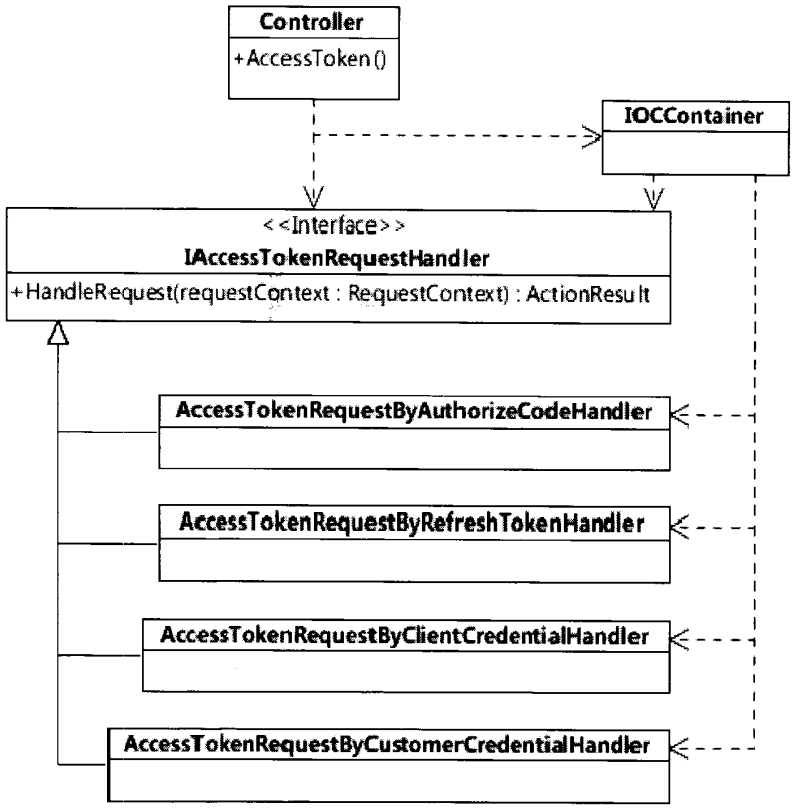

图 3.10 基于 IOC 的访问令牌请求处理设计图

Figure 3.10 IOC style for token request handling

② 客户端设计

客户端主要需要关注授权服务提供方的配置信息与授权请求与访问令牌这三

个对象。如图 3.11 所示，授权服务提供方包括 ID、名称、颁发给客户端的标识与共享密钥，以及相应的授权地址与访问令牌地址等属性。授权请求包括授权码，客户端重定向地址、请求的处理类型、请求的权限范围以及客户端状态等属性。访问令牌具有 ID、颁发的授权服务提供方、颁发时间、失效时间、权限与使用次数统计等属性。对于授权请求，客户端可以选择暂时缓存在内存中，而访问令牌，一般可以存储至数据库中，以供用户下次继续使用，同时，也可以根据令牌中的刷新令牌，在访问令牌过期后，请求新的访问令牌。

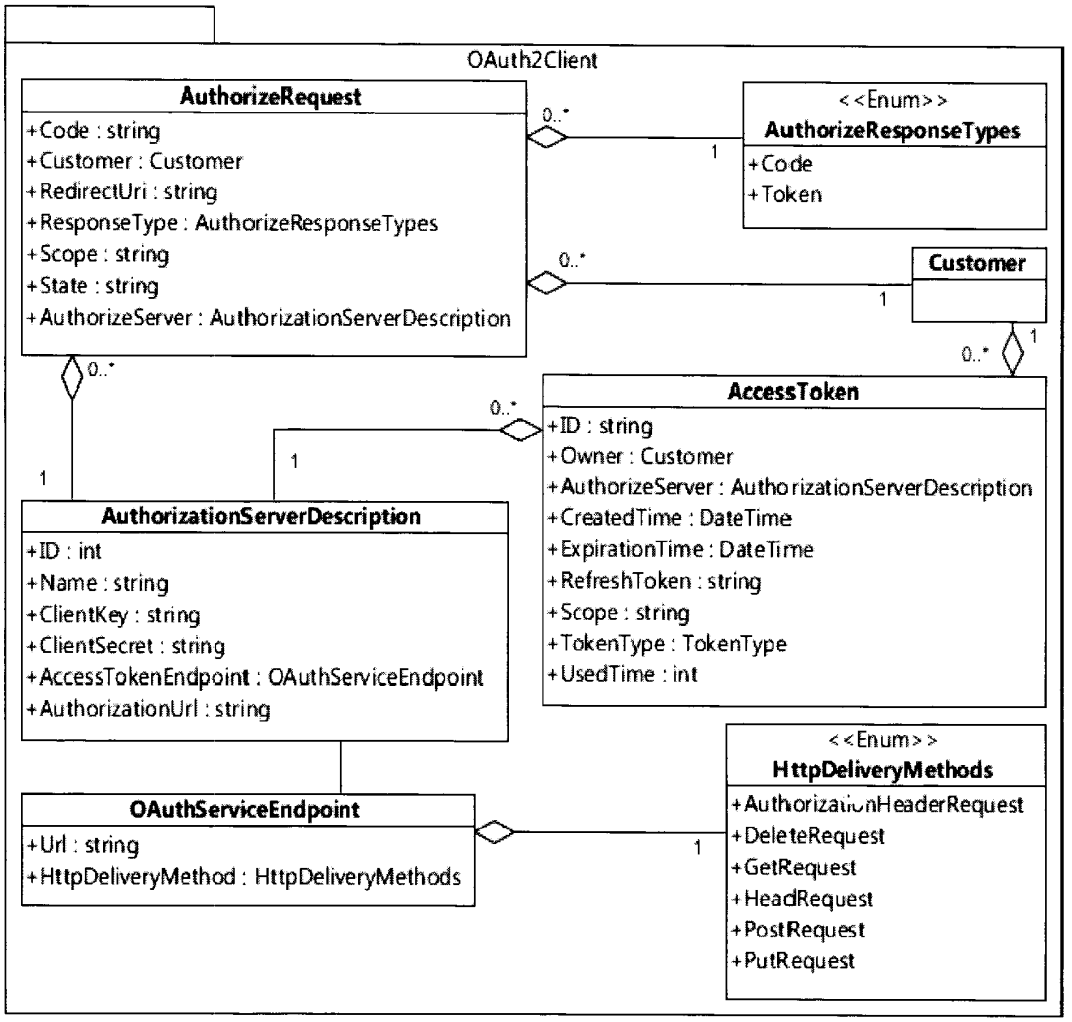

图 3.11 OAuth 2.0 客户端类图

Figure 3.11 OAuth 2.0 class diagrams for client side

3.3 本章小结

本章先后对 OAuth 协议的 1.0 版本与 2.0 版本进行了具体的用例分析，阐述了协议中涉及到的参与者以及各个参与者在协议的工作流程中所参与的活动。

而后，在用例分析的基础上，抽象出了协议各个版本的类图。而后，设计出了具有可维护性与高扩展性的系统架构。对于协议1.0版本，提出了基于AOP的编程方法，来完成客户端验证、签名验证、重复请求校验等一系列重复性的工作，从而使得降低了项目的开发难度，也使得代码能够保持得足够简洁。对于2.0版本，本章提出了基于IOC的访问令牌请求处理模式，从而使得系统能够灵活地处理各种方式的访问令牌请求的同时，也提高了系统的扩展性与可维护性。

最后，对于客户端的设计，本章节提供了相应的类图设计，以及对相关令牌的存储方式的意见与建议。

4　基于OAuth协议的电商数据平台与应用的实现

在前一章的分析与设计的基础上，本章将分别针对不同的 OAuth 协议版本，阐述相应的 DX 数据平台与应用之间进行沟通的实现。

4.1　开发环境介绍

本课题将基于.Net 平台，使用 C#语言阐述基于 OAuth 协议的社会化网络应用的实现。开发工具为 Microsoft Visual Studio 2010，并使用 Asp.Net MVC 3 技术作为 MVC 的技术解决方案。数据库使用 MongoDB 2.0，相应的数据库驱动使用 MongoDB 官方提供的 C#驱动。除此之外，还需要一些第三组件来辅助开发。完整的开发环境说明如表 4.1 所示。

表 4.1 开发环境说明

Table 4.1Development environment specifications

组件名称	用途
Microsoft Visual Studio 2010[28]	开发的 IDE
MongoDB 2.0	数据库，官网地址 http://www.mongodb.org/
MongoDB CSharp Driver	MongoDB 的 C#驱动 可从 https://github.com/mongodb/mongo-csharp-driver 下载
Asp.Net MVC 3[29]	.Net 平台上的一个 MVC 技术解决方案
Log4Net[30]	日志记录组件
DotNetOpenAuth[31]	基于.Net 平台的开源的 OpenAuth 辅助类库，支持 OpenID、OAuth 等协议，用于辅助客户端完成授权申请的整个流程
StructureMap[32]	IOC 框架
NUnit	单元测试框架

4.2　基于 OAuth 1.0 的实现

4.2.1　服务提供方实现

在本小节中，我们将对服务提供方将对数据的存取、临时请求令牌的申请、授权页面、访问令牌的申请以及基于 AOP 的请求验证设计提供具体的实现方案。

① 针对 MongoDB 的数据存取实现

由于 MongoDB 是一个比较松散的文档型数据库，而并不像传统的关系型数据库一样有严格的数据库规格说明。MongoDB 内部都是一个个的基于 BSON 格式的

文档。这些文档按开发者的意愿，被归类到具体的集合（collection）中。故而，针对 MongoDB 的数据存取方案，也有别于针对传统的关系型数据库的使用 ADO.Net 进行数据存取的方式。

在 MongoDB 的 C#驱动中，主要有如表 4.2 所示的几大对象。

表 4.2 MongoDB C#驱动中的主要对象表

Table 4.2 Major classes in MongoDB C# driver

类名	说明
MongoServer	代表一个 MongoDB 的数据库服务器，比如安装在 mongodb://localhost:27017 的数据库
MongoDatabase	MongoDB 服务器中的一个数据库实例
MongoCollection	MongoDB 数据库中的一个用户自定义的集合
IMongoQuery	MongoDB 查询命令
BsonDocument	MongoDB 数据库中的一个文档

在对 MongoDB 的数据进行读取前，首先需要获取到 MongoServer 对象的实例，然后获得该 MongoServer 下的一个具体的数据库，再之后获得该数据库下的具体集合的实例。再获得了集合的实例后，普通 C#对象就可以序列化至集合中，或者是将集合内的数据反序列化成一个普通的 C#对象。序列化存储数据如代码 4.1 所示，读取数据并进行反序列化如代码 4.2 所示。

代码 4.1MongoDB 数据存储示例

Code 4.1Save data example in MongoDB

```
MongoServer server = MongoServer.Create(this.ConnectionString);
MongoDatabase database = server.GetDatabase(this.DataBaseName);

using (server.RequestStart(database))
{
    MongoCollection collection = database.GetCollection(this.CollectionName);

    // 序列化 AccessToken 对象
    BsonDocument doc = new BsonDocument();
    doc.Add("ID", accessToken.ID);
    doc.Add("Secret", accessToken.Secret);
```

```
    ...
    doc.Add("Status", accessToken.Status.ToString());

    collection.Insert(doc); // 插入新的文档至数据库
}
```

代码 4.2MongoDB 查询数据并反序列化为 C#对象示例

Code 4.2Read and de-serialize data into C# object from MongoDB

```
MongoServer server = MongoServer.Create(this.ConnectionString);
MongoDatabase database = server.GetDatabase(this.DataBaseName);

using (server.RequestStart(database))
{
    // 构建 mongo 查询命令
    IMongoQuery query = Query.EQ("ID", "2dda1db53b6c47e182e8f9340728dbf0");
    doc = collection.FindOne(query); // 查找指定的文档

    if(doc != null)
    {
        // 反序列化 BSON 文档至 c# 对象
        AccessToken token = new AccessToken();

        token.ID = MongoDBUtils.GetString("ID", doc);
        token.Secret = MongoDBUtils.GetString("Secret", doc);
        ...
        token.Status = Enum.Parse<TokenStatus>(MongoDBUtils.GetString("Status", doc));
    }
}
```

② 临时请求令牌申请的实现

在 Web 程序中，添加 OAuthController 类，作为 MVC 的控制器。在该类下添加 RequestToken 方法，而后客户端通过访问/OAuth/RequestToken 就可以申请临时令牌。该方法核心实现如代码 4.3 所示。

代码 4.3 OAuth 1.0 临时令牌请求实现代码

Code 4.3OAuth 1.0 temp request token handling

```
public ActionResult RequestToken(string oauth_consumer_key, string oauth_callback)
{
    ...

    // 创建新的临时请求令牌
    var requestToken = RequestTokenBiz.CreateNewToken(client, oauth_callback);
    requestTokenBiz.SaveToken(requestToken); //存储令牌

    var responseDic = new Dictionary<string, string>() {
        {"oauth_token", requestToken.ID},
        {"oauth_token_secret", requestToken.Secret},
        {"oauth_callback_confirmed", "true"}
    };

    return Content(responseDic); //按协议指定格式，返回临时令牌
}
```

③ 授权页面实现

授权页面对应的方法为 Authorize。该方法需要能够引导用户登录，并且在用户已经授权给应用时，直接将页面跳转回客户端的回调地址。核心实现如代码 4.4 所示。

代码 4.4OAuth 1.0 授权页面代码

Code 4.4 OAuth 1.0 authorization handling

```
public ActionResult Authorize(string oauth_token)
{
    ...
    if (ActiveCustomer == null) // 客户未登录，引导其至登录页面
    {
        UrlHelper urlHelper = new UrlHelper(this.Request.RequestContext);
        var returnUrl = urlHelper.Action("Authorize", new { oauth_token = oauth_token });
        return RedirectToAction("LogOn", "Account", new { returnUrl = returnUrl });
```

```
    }

    var accessTokenBiz = SocialServerApp.Container.GetInstance<AccessTokenBiz>();
    // 获取已经颁发给客户的令牌
    var accessToken = accessTokenBiz.LoadToken(requestToken.Client.ID, ActiveCusto-
mer.Email);
    if (accessToken != null && accessToken.Status == TokenStatus.Normal)
    {
        // 已经颁发过，直接返回令牌
        return CustomerAuthorizeClientResult(true, requestTokenBiz, requestToken);
    }

    // 显示授权页面
    return View(new AuthorizeModel() { RequestToken = requestToken, Customer = ActiveCus-
tomer });
}
```

④ 申请访问令牌的实现

申请访问令牌时，需要验证临时请求令牌的校验码，验证通过后，就可以创建新的访问令牌，并按协议规定格式返回给客户端了。逻辑相对比较简单。为了节省篇幅，这里就不展示具体的代码了。

⑤ 基于 AOP 的请求验证设计的实现

在 Asp.Net MVC 中，我们可以很好地利用 ActionFilterAttribute，来将相应的代码注入到 Action 的执行过程中。通过 ActionFilterAttribute 中的 OnActionExecuting 方法与 OnActionExecuted 方法，分别表示了在被注入方法执行之前与执行完之后，需要注入的代码。在 OAuth 1.0 中，我们需要有 NoReplayRequestAttribute、ValidSignatureAttribute、AuthenticateClientAttribute 与 AuthenticateAccessAttribute 等注入的类。以防止重复请求的 NoReplayRequestAttribute 为例，它确保了请求参数中的 oauth_consumer_key、oauth_timestamp 与 oauth_nonce 的组合只能出现一次。核心实现如代码 4.5 所示。

代码 4.5OAuth 1.0 中防止重新请求代码

Code 4.5 OAuth 1.0 request replay detecting

```
public class NoReplayRequestAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        ...
        var nonce = nonceBiz.LoadNonce(clientID, strNonce, timestamp);
        if (nonce != null) // 之前存在过此请求
        {
            throw new HttpException(400, "Replay request is prohibited.");
        }

        nonce = new OAuthNonce()  {
            ClientID = clientID,
            Nonce = strNonce,
            Timestamp = timestamp
        };
        nonceBiz.SaveNonce(nonce);

        base.OnActionExecuting(filterContext);
    }
}
```

而后，将这些需要注入的类，以 Attribute 的形式，添加到指定方法中去。如代码 4.6 所示。

代码 4.6 AOP 注入方法代码

Code 4.6AOP injecting sample

```
[AuthenticateClient]
[NoReplayRequest]
[ValidSignature(TokenType = TokenTypes.None)]
public ActionResult RequestToken(string oauth_consumer_key, string oauth_callback)
{
    ...
}
```

4.2.2 客户端的实现

借助 DotNetOpenAuth 组件，我们可以快速地完成整个 OAuth 协议在客户端的工作流程。以 WWW 网络应用为例，在用户启动授权流程后，通过如代码 4.7 所示简单地页面，就可以引导用户至服务提供方进行授权，并在返回至本地的回调地址后，完成访问令牌的获得。

代码 4.7 OAuth 1.0 客户端代码

Code 4.7 OAuth 1.0 client side sample

```
var serviceDescription = new ServiceProviderDescription {
    RequestTokenEndpoint = new MessageReceivingEndpoint(
        serviceRootUrl + "/OAuth/RequestToken", HttpDeliveryMethods.PostRequest),
    UserAuthorizationEndpoint = new MessageReceivingEndpoint(
        serviceRootUrl + "/Oauth/Authorize", HttpDeliveryMethods.GetRequest),
    AccessTokenEndpoint = new MessageReceivingEndpoint(
        serviceRootUrl + "/Oauth/AccessToken", HttpDeliveryMethods.GetRequest),
    TamperProtectionElements =
        new ITamperProtectionChannelBindingElement[] {
            new HmacSha1SigningBindingElement()
        }
};
WebConsumer client = new WebConsumer(serviceDescription, this.TokenManager);

var accessTokenResponse = client.ProcessUserAuthorization();
if (accessTokenResponse != null) {
    // 获取到了访问令牌
    this.AccessToken = accessTokenResponse.AccessToken;
} else if (this.AccessToken == null) {
    // 暂未获得访问令牌，开始授权流程，回调服务地址为当前页面地址
    client.Channel.Send(client.PrepareRequestUserAuthorization());
}
```

4.3 基于 OAuth 2.0 的实现

4.3.1 授权服务提供方实现

在 OAuth 2.0 服务提供方，主要有授权服务与申请令牌服务这两个方法。

① 授权服务的实现

授权服务与③介绍的授权页面逻辑一样，需要引导客户进行登录，并且在客户已经拥有授权的情况下，需要直接返回相应的结果，否则展示授权页面。相对而言，OAuth 2.0 内需要额外关注的是客户端请求的响应是 code 还是 token，如果是 code 则需要在授权服务提供方对返回新的 AuthorizeRequest 并进行记录。如果是 token 则需要将访问令牌直接返回。总体而言，逻辑不算太复杂，具体实现代码也就无须展示了。

② 基于 IOC 的访问令牌请求处理设计的实现

在 3.2 中，我们提出了基于 IOC 的访问令牌请求处理的设计，我们已经有了 IAccessTokenRequestHandler 接口与相应的实现类。那么我们就需要在程序启动的时候，将这一些具体的实现类注册到 IOC 容器中。如代码 4.8 所示。

代码 4.8 注册实现类至 IOC 容器

Code 4.8Register concrete classes to IOC container

```
protected void Application_Start()
{
    Container = new Container();
    Container.Configure(x =>
    {
        x.For<IAccessTokenRequestHandler>()

    .Use<AccessTokenRequestByAuthorizeCodeHandler>().Named("authorization_code");
        x.For<IAccessTokenRequestHandler>()
            .Use<AccessTokenRequestByRefreshTokenHandler>().Named("refresh_token");
        x.For<IAccessTokenRequestHandler>()

    .Use<AccessTokenRequestByClientCredentialHandler>().Named("client_credentials");
        x.For<IAccessTokenRequestHandler>()
            .Use<AccessTokenRequestByCustomerCredentialHandler>().Named("password");
    });

    ...
}
```

在我们的访问令牌请求响应方法中，我们通过容器获取到指定的具体的令牌请求处理类的实例，而后调用HandleRequest方法，即可以完成请求的响应，见代码4.9。

代码4.9 通过IOC容器获取具体的实现类实例

Code 4.9 OAuth get specified instance from IOC container

```
public ActionResult AccesToken(string grant_type)
{
    IAccessTokenRequestHandler handler =
        Container.GetInstance<IAccessTokenRequestHandler>(grant_type);

    return handler.HandleRequest(this.Request.RequestContext);
}
```

4.3.2 资源服务提供方的实现

资源服务提供方在进行访问令牌验证的同时，需要专注与资源的服务。令牌的校验，通过授权服务提供方提供的SDK dll来完成。所示为获取用户详细资料的接口。

代码4.10 OAuth 2.0 资源服务提供方服务样例

Code 4.10 OAuth 2.0 resourceoperation request handling sample in resource server side

```
public ActionResult CurrentUser(string access_token)
{
    AccessToken token;
    AuthorizationServer.SDK.TokenValidations.ValidToken(access_token, out token);

    return Json(token.Owner);
}
```

4.3.3 客户端实现

DotNetOpenAuth组件也支持OAuth 2.0协议。通过使用它提供的WebServerClient类，以及该类下的ProcessUserAuthorization以及RequestUserAuthorization方法，就可以顺利地完成整个协议的授权流程。如代码4.11所示。

代码 4.11 OAuth 2.0 客户端授权流程代码

Code 4.11 OAuth 2.0 authorization code in client side

```
var serviceDescription = new AuthorizationServerDescription
{
    TokenEndpoint = new Uri(serviceRootUrl + "/oauth2/AccessToken"),
    AuthorizationEndpoint = new Uri(serviceRootUrl + "/oauth2/Authorize"),
};
var client = new WebServerClient(serviceDescription);

IAuthorizationState authorization = client.ProcessUserAuthorization();
if (authorization == null) // 暂未获得授权
{
    // 开启授权, redirect_uri 为当前页面
    // 页面跳转至授权服务提供方的授权页面
    client.RequestUserAuthorization(
        scope: new string[] { "user_profile", "order" },
        state: Session.SessionID);
}
```

在获取到了授权码之后，客户端就可以向资源服务提供方发起服务请求了。该请求中，需要额外添加 access_token 参数。代码 4.12 所示为获取当前资源拥有者详细资料的请求。

代码 4.12OAuth 2.0 客户端请求资源服务代码样例

Code 4.12 OAuth 2.0 resource operation request in client side

```
UriBuilder ub = new UriBuilder(resourceServerRootUrl + "User/CurrentUser");
ub.AppendQueryArgument("access_token", authorization.AccessToken);

var request = WebRequest.Create(ub.ToString());
string resonseString;
using (var response = request.GetResponse())
{
    responeString = GetWebResponseString(response);
}
```

```
// 反序列化 JSON 格式的响应数据
...
```

4.4 本章小结

本章首先介绍了开发环境以及需要用到的相关组件。

而后，本文分别针对 OAuth 的不同版本，阐述了基于 OAuth 协议的社会化网络应用的实现。

在 OAuth 1.0 中，针对服务提供方的临时令牌请求、授权页面、访问令牌的请求等提供核心实现代码的同时，也针对如何读取与插入数据至 MongoDB 数据库提供了示例代码。在 OAuth 2.0 中，针对基于 IOC 的访问令牌请求设计的实现也做了详细的介绍。对于客户端的实现，介绍了基于 DotNetOpenAuth 组件，如何实现 Asp.Net 客户端，并提供了访问资源服务提供方的示例。

5 OAuth协议在社会化网络中的应用研究

DX数据平台是整个社会化网络应用中的一员。在基于OAuth协议实现了该平台之后，本章将着眼于整个社会化网络应用，针对OAuth协议的用途进行深入地探讨与研究。

5.1 私有资源，跨界使用

通过OAuth协议，客户端在获取到了相应的访问令牌后，就可以代表资源拥有者，操作处于服务提供方的私有资源。故而有了行业标准的OAuth协议，私有资源的跨界使用不再是问题。

如图5.1所示，由于解决了私有资源的跨界使用问题，OAuth协议被广泛地应用在了我们学习生活、工作交友以及游戏与娱乐等方方面面的应用中。

就学习方面而言，通过OAuth协议，精彩的博文可以被分享到第三方应用中。而同时，许多博文的作者都同时使用着多个的博客服务应用，通过OAuth应用，博文在其中的一个应用中被发布了之后，可以被其它博客应用获取到，并同步过去。

就生活方面而言，像Foursquare这种应用就是通过OAuth协议来实时地交个人地理位置分享到诸如Facebook之类的社交网站中去。Instagram通过OAuth协议实现了将照片推送到指定用户的Facebook、Twitter、Flickr等账户中。当然，一些第三方的照片冲印的应用，通过OAuth协议，也能够获取用户在照片服务应用中的照片等。

就工作方面而言，OAuth协议能够辅助企业内部各个系统之间的系统调用过程的完成。

OAuth也能够有效地为人们的日常交友提供服务。如依托于Facebook的一款名为My Calendar - Birthdays的应用，通过OAuth协议，获取到了笔者在Facebook的好友之后，就会定期的帮助提醒好友的生日，并在好友生日的那天自动发送贺卡并送上生日祝福。

基于OAuth协议，依托于大型社交平台的游戏也具有相当高的互动性。Zynga公司推出的社会化网络游戏，依托于Facebook、Google+之类的社交网站，可以使得玩家可以方便地邀请其好友一起加入游戏并产生互动。同时，玩家在游戏中的成绩与战果也可以实时地分享到这些社交网站当中去。在使得游戏的可玩性提高了不少的同时，这也游戏的推广成本也降低了许多。

当然，OAuth协议在音乐与视频分享等人们的日常娱乐生活中，也起到了很大

的作用。

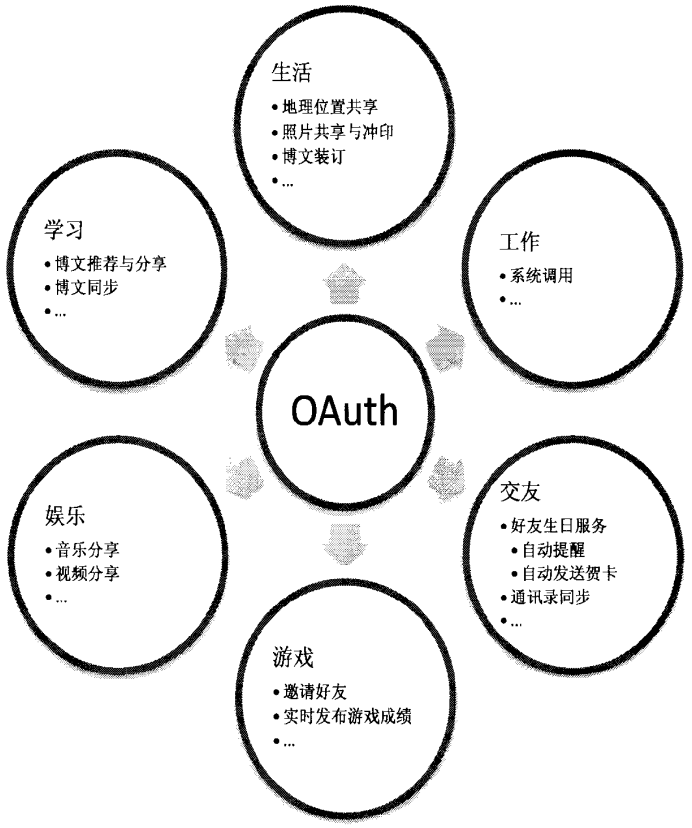

图 5.1OAuth 辅助资源跨界调用的应用分类概览图

Figure 5.1Categories of applications that use OAuth to access private resources

5.2 单一账号，到处通行

一般面言，用户在使用网络应用之前需要登录。而登录的过程，其实就是通过用户登录凭证（登录名+登录密码）来鉴别用户以获取用户唯一标识的过程。

我们知道个人资料也可以算是一种私有资源，而个人资料中总会有一些能够作为用户唯一标识的属性，如 Email。故而，如果获得了这个唯一标识，社会化网络应用之间，就可以进行用户的绑定，进而可以使用第三方的账号中的用户唯一标识，获取到本地账户系统中用户的唯一标识，从而达到完成登录的效果。由于 OAuth 能够有效地解决这种跨界获取私有资源的问题，因此 OAuth 协议也可以用作是以单一账号，通行于各种网络应用之间的方案。

如图 5.2 所示，在通过 OAuth 协议获取到了相应的授权后，通过获取资源拥有者的资料，获取到资源拥有者在服务提供方的唯一标识（Email），而后，通过此标识与本地账户系统进行比对，若本地账户系统不存在此标识，则可以以此标识创建一个无需密码的账户，若已经存在，则可以直接进行用户绑定，并进行相应的

登录会话设置，完成整个登录过程。

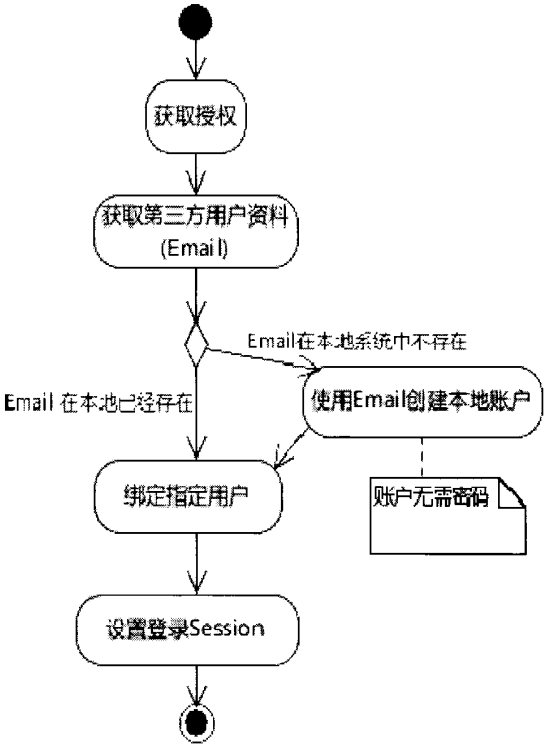

图 5.2 借助 OAuth 协议使用第三方账户登录流程图

Figure 5.2Use third-party account to login flow by using the OAuth protocol

通过对 OAuth 协议的了解，并结合上述的过程，用户真正需要关心账户的登录名与登录密码的时刻，是发生在授权过程中，而在本地系统中，用户无需关心登录凭证（既无登录名，也无登录密码），甚至是可以完全忽略本地账户系统的存在。故而，OAuth 协议在有效地解决了社会化网络应用中账户管理麻烦的问题的同时，也对加强用户隐私保护工作起到了相当重要的促进作用。

5.3 让互联网为人工作

5.3.1 生活智能化

在普通生活中，人们使用社会化网络时，各应用之间的交互基本上都需要用户人为地触发。比如前面提到的使用第三方账户登录功能，或者说电子商务系统移动终端的下单与查询订单等功能，都必须要由用户人为操作才能触发。而很少有应用，除非其本身支持，会自动地与第三方应用主动沟通，从而来更好地为人们服务。

以我们的电子商务系统为例，在我们系统中下单支付之后，用户可能希望这次

购买所花费的金额，能够被自动地添加到某个记账应用中。而我们的订单系统是不会知道用户所使用的记账应用的存在的，这也就导致了用户不得不在他所喜好的记账应用中，手动地添加一条记录。这对用户来说，体验是相当差的。故而，如果有一个应用，在侦测到订单系统中，多了一条订单记录后，能够自动地为用户到记账应用中添加一条相关的记录，将使得人们的生活变得更加智能化。

依托于 OAuth 协议的 http://ifttt.com 就是这样的一个应用。它提出的让互联网为人工作的理念，就是这种无需人为直接触发，而是应用自动帮助人们在各个应用之间协同作业模式的完美的精神写照。社会学家认为，人的所有活动，都是选择的结果。Ifttt.com 很好地践行这一理念，这也是它名称 If This Then That 的由来。在该应用中，这一系列的选择而产生的相应活动，被定义为一个触发器。用户在一个应用中的某个操作，都有可能自动触发在任意其它应用中的相应的活动。而能够监测到这种应用中的操作与触发其它应用中的相应活动，OAuth 协议起到了不可忽视的作用。

5.3.2 工作自动化

OAuth 协议也可以用来促进人们日常工作的自动化。

以 DX 这一电商平台为例，除了优质的产品与服务，电商的市场推广与营销也是电商得以持续发展与繁荣的主要动力之一。而电商的市场推广与营销方式，除了常见的 SEO 优化、在 Google 与 Facebook 等大型平台的广告投放外，依托于 Twitter 之类的微博平台的营销也成了它的一个重要组成部分。例如，当用户在 Twitter 上发了一条关于想要购买数码相机的微博之后，我们的自动化营销工具，通过 OAuth 协议，定期获取 Twitter 上的微博，通过对微博内容进行分析，就可以通过 OAuth 协议，对微博进行评论以提供处于 DX 的数码相机的专题页的链接，以及相关的折扣与优惠信息等。

可以说，是 OAuth 协议打破了各式各样的社会化网络应用之间的壁垒，从而使得互联网能够自由地连通、并默默地为人们工作着，才能使得人们的生活与工作越来越智能与轻松。

6 结论与展望

6.1 结论

针对社会化网络中的资源跨界共享问题，账户管理与维护困难以及隐私保护工作难做等问题，本课题研究了 OAuth 协议，并依据此协议，分析、设计并实现了 DX 电商系统的数据平台。本课题取得的主要成果有：

① 概括、分析并比较了社会化网络中资源跨界共享与使用的三种模式。

1) 绝对信任模式在企业内部系统中应用得比较广泛，但不适合互联网大环境下任意应用之间的资源跨界使用；

2) 授予用户登录凭证的二足模式在邮件、博客等开放性互联网应用中得到了普遍的应用，但使得用户隐私泄露的风险变得过高，而且相应的资源获取的权限也得不到有效的控制；

3) 三足模式在用户参与并控制授权下，有效地解决了互联网资源跨界使用的问题。

② 深入地研究与学习了基于三足模式的 OAuth 授权协议，完成了相应的需求分析。

OAuth 协议它依赖于客户端与服务提供方相互通信，并适时引导用户进行授权。OAuth 1.0 具有如下鲜明的特点：

1) 携带令牌证书以确保请求合法性：客户端与服务提供方的所有通信都需要在携带令牌的前提下进行，令牌证书由令牌标识与共享私钥组成。令牌的种类有临时请求令牌、访问令牌证书与客户端证书这三种。

2) 使用签名以确保通信数据有效性：通信的数据都需要通过使用相应证书的共享私钥在指定的算法下进行签名以确保通信请求的合法性。签名算法有 HMAC-SHA1、RSA-SHA1 与 PLAINTEXT 这三种。

3) OAuth 1.0 要求不得回放请求。客户端向服务提供方发起的请求，需要携带客户端标识、timestamp 与 nonce 这三个参数，而针对这三者的组合，在整个应用生命周期中，不得出现重复。

相比 OAuth 1.0，OAuth 2.0 有着明显的改善，主要体现为：

1) 简洁性：OAuth 2.0 版本抛弃了 1.0 中繁杂的签名工作，而改为使用 TLS 技术以确保通信数据的有效性。

2) 授权方式的灵活与多样性：OAuth 2.0 的授权流程包括通过使用授权码的服务端流程、客户端流程、使用用户登录凭证获取授权的流程以及使用客户端证书获取授权的流程这四种，每一种授权流程都有其各自的适用场景。

③ 基于 OAuth 协议，设计并实现了具有高性能与可扩展性的 DX 电商数据平台。

1) 对于 OAuth 1.0，本课题设计出了基于 AOP 的请求验证设计，从而有效地降低了开发难度与节省了开发成本，并在一定程度上提高了代码的可读性与可维护性。

2) 对于 OAuth 2.0，为了适应其授权方式的灵活与多样性，设计出了基于 IOC 的访问令牌请求设计，从而使得使用一些简洁的代码，就可以有效地处理不同方式的授权请求，并且保持了程序的可扩展性。

3) 对于协议的客户端，本课题也为其引导用户进行授权的方式提供了三种可行方案，即页面直接跳转引导授权、调用系统浏览器引导授权与内嵌浏览器引导用户授权这三种方案。

根据相应的设计方案，本课题在使用 C#语言，在 Asp.Net MVC、MongoDB、SturctureMap、DotNetOpenAuth 等应用程序框架与组件的基础上，实现了基于 OAuth 协议的服务提供方- DX 数据开放平台，并为基于 B/S 模式的客户端，提供了相应的协议工作流程示例代码。

④ 本课题在最后研究了 OAuth 的作用与应用前景。

1) 对于其在解决跨界资源的使用的问题上，做出了充分的认可与肯定。

2) 在对使用 OAuth 协议，来实现使用单一账号而通行于所有网络应用上，提供了可行的技术解决方案，从而有效地缓解了当今网络中账号维护麻烦与隐私保护工作难做的问题。

3) 最后，本课题还对如何使用 OAuth 从而使得互联网能够自动为人们工作与服务，以促进生活的智能化与工作的自动化，进行了相应的分析与阐述。

作为互联网标准的 OAuth 协议的出现，在解决了相应的网络中呈现出的问题的同时，也为促使整个产业生态环境，由竞争向相互合作的转变，做出了突出的贡献。这也使得互联网的社会化进程，迈上了一个新的台阶，进而开启了一个崭新的时代。

6.2 展望

在本课题中，发展中的 OAuth 2.0 版本，目前处在第 25 个修改稿中，我们殷切地期待着它最终的定稿。

我们也希望越来越多的网络应用能够加入到对 OAuth 协议支持的阵营中，从而使得我们的网络生态环境更加的开放与相互合作，并为社会的发展与变革做出更大的贡献。

致 谢

本人首先需要感谢我的父母与家人，感谢他们二十多年来对我的养育与关心。是他们的辛苦劳作，负担起了这么些年来的学习与生活开支，才使得我能够专心地研究并完成本课题成为可能；是他们的微言细语，使得我能够健康地成长，并养成了乐观向上，积极健朗的性格；更是他们的慈眉善目，使得我能够懂得学习与尊重他人，使得我学会了如何在社会环境中适应并融入进去，从而能够做一个健康的对社会有用的人。

其次，本人需要特别感谢我的导师陈林教授。从本科入校起，他的温文儒雅就深深地打动了我。而研究生的三年，更让我能够亲身体会到他那有如春风般的温暖。没有他，我不可能在研究生期间专注于互联网行业，从而也不可以能够接触到本课题，而相应的研究与实践工作则会无从谈起。在他的指导下，我的研究生生活才会变得如此地有意义。

再次，我要感谢我在研究生期间接收我实习的公司：Intel 与易宝（北京）信息技术有限公司。在 Intel 的实习的一年，使我真正地沉淀了下来。Intel 公司的六个核心的企业文化，将会是我一生的财富。借此机会，我要特别感谢 Intel 的 George, Liu 与 Lei, Li 两位博士，你们是我的经理、是我的同事、更是我一生的好友。同时，我要感谢易宝电脑系统有限公司，正是在易宝的工作经历，才使得我有机会将此课题研究的内容付诸实践。故而，我也要特别感谢在易宝一起参与到项目中的同事，没有你们的支持，我的实践经历也不可能如此丰富。

最后，在即将告别重庆大学之际，七年时间，点点滴滴的学习与生活经历不断地涌现在眼前，而我不禁泪眼婆娑。我需要感谢软件学院，以及所有指导过我的老师，能够做你们的学生，是我莫大的荣幸。同时，我还需要特别感谢我的本科与研究生期间的两位辅导员，郑晗琳老师与刘援朝老师，谢谢你们的热心辅导！同时，我需要感谢所有陪伴过我的同学们，是你们的认真学习，不断激励着我不断上进；是你们的欢声笑语，不断陪伴着我前行；更是你们的和平友善，使我强烈地感受到了如兄弟姐妹般的热情，以及如家般的温暖。

感谢一切！

朱扬谷

二 O 一二年四月

参考文献

[1] LATimes. How Instagram founder Kevin Systrom became insta-rich [EB/OL]. http://www.latimes.com/business/la-fi-instagram-systrom-20120411,0,7316011.story. 2011.

[2] TechFrom 科技源. 创业故事-仅 18 个月价值$10 亿的图片分享 Instagram [EB/OL]. http://www.techfrom.com/25491.html. 2012.

[3] 百度百科. 密码泄露门 [EB/OL]. http://baike.baidu.com/view/7167245.htm. 2011.

[4] Wikipedia. OpenID [EB/OL]. http://en.wikipedia.org/wiki/OpenID.

[5] OpenID Authentication 2.0[S].

[6] DigitalTrends.com. Microsoft closing Live Spaces, moving users to WordPress [EB/OL]. http://www.digitaltrends.com/buying-guides/microsoft-closing-live-spaces-moving-users-to-wordpress/. 2010.

[7] Wikipedia. XML-RPC [EB/OL]. http://en.wikipedia.org/wiki/XML-RPC.

[8] Dave Winer. RFC: MetaWeblog API [EB/OL]. http://xmlrpc.scripting.com/metaWeblogApi.html. 2003.

[9] Google. Google Calendar API v1 Developer's Guide: Protocol [EB/OL]. https://developers.google.com/google-apps/calendar/v1/developers_guide_protocol. 2006.

[10] Yahoo. Browser-Based Authentication (BBAuth) [EB/OL]. http://developer.yahoo.com/bbauth/.

[11] Wikipedia. OAuth [EB/OL]. http://en.wikipedia.org/wiki/OAuth.

[12] RFC 5849. The OAuth 1.0 Protocol[S]. IETF

[13] Facebook. Authentication - Facebook Developers [EB/OL]. http://developers.facebook.com/docs/authentication/.

[14] 新浪科技. 中国互联网协会评出 2011 年互联网十大重要事件 [EB/OL]. http://tech.sina.com.cn/i/2012-01-11/14496627567.shtml. 2012.

[15] Wikipedia. HMAC [EB/OL]. http://en.wikipedia.org/wiki/HMAC.

[16] RSA-SHA1 Signature Suite - Version 1.0[S]. World Wide Web Consortium

[17] draft-ietf-oauth-v2-25 - The OAuth 2.0 Authorization Framework[S]. IETF

[18] Wikipedia. Transport Layer Security [EB/OL]. http://en.wikipedia.org/wiki/Transport_Layer_Security.

[19] Ryan Boyd. Getting Started with OAuth 2.0 [M]. O'Reilly Media, 2012.

[20] RFC2396. Uniform Resource Identifiers (URI): Generic Syntax[S]. IETF

[21] Wikipedia. NoSQL [EB/OL]. http://en.wikipedia.org/wiki/NoSQL.

[22] Wikipedia. MongoDB [EB/OL]. http://en.wikipedia.org/wiki/MongoDB.

[23] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures [D]. University of California, Irvine, 2000.

[24] RFC2616. Hypertext Transfer Protocol -- HTTP/1.1[S]. IETF

[25] Douglas Crockford. JSON [EB/OL]. http://json.org/.

[26] Wikipedia. Aspect-oriented programming [EB/OL]. http://en.wikipedia.org/wiki/ Aspect-oriented_programming.

[27] Wikipedia. Inversion of control [EB/OL]. http://en.wikipedia.org/wiki/Inversion_of_control.

[28] Microsoft. Microsoft Visual Studio 2010 [EB/OL]. http://www.microsoft.com/ visualstudio/en-us.

[29] Steven Sanderson, Freeman Adam. Pro ASP.NET MVC 3 Framework, 3rd Edition [M]. Apress, 2011.

[30] Apache. log4net [EB/OL]. http://logging.apache.org/log4net/.

[31] Andrew Arnott. DotNetOpenAuth [EB/OL]. http://www.dotnetopenauth.net/.

[32] StructureMap [EB/OL]. http://docs.structuremap.net/.

[33] 刘镝，张智江，张尼．基于国内开放平台的 OAuth 认证框架研究[J]．信息通信技术，2011(6).

[34] 时子庆，刘金兰，谭晓华．基于 OAuth2.0 的认证授权技术[J]．计算机系统应用，2012，21(3)：1-4.

[35] Jess Chadwick. Programming Razor[M]. O'Reilly, 2011.

[36] Jon Skeet. C# in Depth, 2nd Edition[M]. Manning Publications, 2010.

[37] Joe Albahari. C# 4.0 in a Nutshell, 4th Edition[M]. O'Reilly, 2010.

[38] Douglas Crockford. JavaScript: The Good Parts[M]. O'Reilly, 2008.

[39] Ibrahim Hashimi. Inside the Microsoft Build Engine: Using MSBuild and Team Foundation Build[M]. Microsoft Press, 2011.

[40] Eelco Plugge, Hawkins Tim, Membrey Peter. The Definitive Guide to MongoDB[M]. Apress, 2010.

[41] Gavin Bell. Building Social Web Applications[M]. O'Reilly, 2009.