1. Chapter 1 Introduction
   1. Introduction to E-government Platform

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

基于B／S(浏览器朋艮务器)结构的企业应用软

件也得到了快速发展，各种应用系统如财务系统、人事系统、生产管理系统等

已经应用到很多企业的生产管理活动中去，为企业提高工作效率和管理水平做

出了巨大的贡献，但由于企业应业务、自身条件、和当时软件技术的影响，这

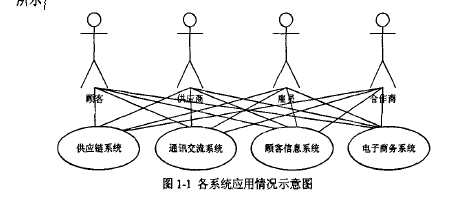
些不同的系统往往是在不同的时期建设起来的，运行在不同的平台上；也许是

由不同厂商开发，使用了各种不同的技术和标准，每个应用系统都有自己独立

的一套身份验证机制，各应用系统采取分散登录、分散管理。又出j：针对愈№

应用各个系统的联系又十分紧密，各用户需的要使用大多数的系统，如图1．1

所示：



* 1. Introduction to the PAAS Cloud Architecture Platform

提出政务云概念；政务云功能模块；UIAACS的需求与挑战

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

云计算作为现代网络信息化发展潮流中的新生产物，其以它数据大集中的建设模式，资源动态配置、信息共享、交付式服务等特点，在网络信息资源的海洋里益突显并逐渐发挥出不可取代的地位与作用。宁波市政务云计算中心是全市智慧城市和电子政务应用的支撑平台。在公共基础应用支撑的优化配置和集中管理的基础上，实现应用系统与中间件的松耦合布局。利用总线设计的思想，为智慧城市和电子政务应用提供业务互动基础生态环境，提高跨部门业务之间的协作能力。



# ESB模块功能概述

ESB全称为Enterprise Service Bus，即企业服务总线。它是传统中间件技术与XML、Web服务等技术结合的产物。它将企业内部开发的服务进行集成，提供了细粒度、标准化的服务统一管理，并包括服务的授权认证管理系统，以及服务监控系统。

ESB系统按照功能可以分为服务注册模块和服务申请模块以及监控模块，ESB系统会接入到统一身份认证系统进行统一认证管理。

。

系统功能模块图4-1如下：

使用人员



* 1. Introduction to the Existing Work

国外统一身份认证的研究比较活跃，目前最流行的技术主要有微软的

Passport技术和自由联盟规范，其中微软的Passport技术实际上是。种Web

Service，它是由微软公司控制的中央统筹式的单一登录服务。自由联盟(LibertyAlliance)是由SUN、IBM、HP、Intel、Oraele、Novell

等150多家公司和机构组成的一个围绕身份认证的技术，该联盟的宗旨是创建

一个具有开放性、联合的、单一身份识别的解决方案。与微软的Passport集中

认证不同，自由联盟采用的是一种称为联邦认证(Federated Identity)的机制，

即自由联盟中的身份提供者不唯一，而是可以相互独立存在的，使用联邦认证

的好处是在有可以避免因单点故障而导致的系统瘫痪。自由联盟相关协议的基

西南交通大学硕士研究生学位论文第3页

础和核心是SAML(S∞an-ityAssertionMarkup Language,安全断言标记语言)。同时随着统一身份认证研究的深入和较高的商业意义，目前已经出现了很

多商用软件和产品【2】，专门的SSO商业软件主要有：Netgdty的Sitemindef(已

经被CA收购)、Novdl公司的iChain、RSA公司的ClearTrust等。还有门户

产品供应商自己的SSO产品如：BEA的WLES、mM的TivoliAccessMarmger、

Sun公司的identity Server、Oracle公司的OID等。这些商业软件一般适用于

客户对SSO的需求很高，并且企业内部采用COTS软件omino、SAP、Sieble

的系统。采用这些商用软件一般都要对要集成的系统做些改造，如在要集成的系统

上安装代理，首先统一这些系统的认证方式(一般采用LDAP或数据库)，然

后才能实现SSO，相对比较麻烦。

另外，也有很多开源组织从事SSO的研究如JOSSO、OPENSSO、

SOURCEID等。

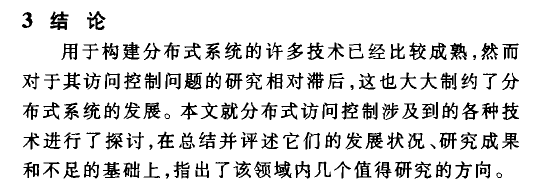
国内的很多机构和系统正在进行统一身份认证的改造，如电力、银行、政

府、高校等信息化比较先近的企业。

信息化的迅猛发展带动越来越多的企业开始了信息化之路。一些大型企业更是纷纷上马了 MIS、OA、ERP、Notes 等系统，逐步实现企业的生产、管理、电子商务等诸多业务无纸化。期望借助信息化的平台，提升企业的效率，加快企业的发展。然而，随着越来越多信息系统的部署，却发现企业生产效率并未达到预期的提高。相反，纷繁复杂的系统登录大量占用了员工的时间，而员工登录的混乱更是给企业信息化带来严重的安全漏洞。

据艾瑞调查发现有将近50%的用户有过网络帐号、密码被盗的经历，据Gartner调查发现网站客户服务工作量的40%来自用户密码问题；据 Forrester 调查估计网站每次为解决用户密码问题的开销是38美元。可以看出，不仅用户为密码问题而烦恼，而网站也在为密码问题而浪费了大量的人力和财力资源。

目前，国内有些网站为了解决用户需要多次输入用户名和口令而提供了登录通行证服务(如网易通行证、新浪通行证等) ，但是所有这些通行证都是基于非常脆弱的用户名/密码认证机制，因为一旦密码被泄露或木马程序非法窃取，则用户名和口令全都被泄露。此类解决方案无法回答用户所担心的两个问题:其一，一旦登录密码被非法窃取，那什么都泄密了；其二，用户把所有其他账号信息都存放在服务提供商的服务器上不放心服务提供商本身。解决这两个最大的问题，单点登录的解决方案应运而生。 国外已开展了单点登录系统的研究，微软的 passport 和美国在线时代华纳的“魔毯”（Magic Carpet）是可行的单点登录解决方案。



* 1. Main Work

本文分析比较了企业应用环境下相对稳定的的单点登录模型、租户管理模型与访问控制模型，结合政务云环境下从对统一认证与访问控制服务的实际需求，设计了服务于宁波市智慧城市和电子政务云的系统模型，最后根据设计，给出了系统关键实现并在测试上线。

* 1. Content Organization

本文的内容是这样安排的：第一章介绍了问题产生的背景，分析了国内外的研究技术现状，提出本文的研究目标：第二章介绍了统一身份认证需要用到的基础理论；第三章分析和比较了现在流行的统一身份认证模型；第四章行了企业统一身份认证需求分析并设计了具有一定实用性的统一身份认证模型；第五章根据模型设计的要求给出了一个具体的实现；最后是全文的总结。

1. Chapter 2 Relevant concepts and theories
   1. B/S model and SSO

瀏覽器-服務器（Browser/Server）結構，簡稱B/S結構，與C/S結構不同，其客戶端不需要安裝專門的軟件，只需要瀏覽器即可，瀏覽器通過Web服務器與數據庫進行交互，可以方便的在不同平台下工作；服務器端可採用高性能計算機，並安裝Oracle、Sybase、Informix等大型數據庫。B/S結構簡化了客戶端的工作，它是隨着Internet技術興起而產生的，對C/S技術的改進，但該結構下服務器端的工作較重，對服務器的性能要求更高。

B/Ｓ是Brower/Server的缩写，客户机上只要安装一个浏览器（Browser），如Netscape Navigator或Internet Explorer，服务器安装Oracle、Sybase、Informix或 SQL Server等数据库。浏览器通过Ｗeb Server 同数据库进行数据交互。

B/S最大的优点就是可以在任何地方进行操作而不用安装任何专门的软件。只要有一台能上网的电脑就能使用，客户端零维护。系统的扩展非常容易，只要能上网，再由系统管理员分配一个用户名和密码，就可以使用了。甚至可以在线申请，通过公司内部的安全认证（如ＣＡ证书）后，不需要人的参与，系统可以自动分配给用户一个账号进入统。

　　B/S结构，即Browser/Server(浏览器/服务器)结构，是随着Internet技术的兴起，对C/S结构的一种变化或者改进的结构。在这种结构下，用户界面完全通过WWW浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现，形成所谓3-tier结构。

B/S结构，主要是利用了不断成熟的WWW浏览器技术，结合浏览器的多种Script语言(VBScript、JavaScript…)和ActiveX技术，用通用浏览器就实现了原来需要复杂专用软件才能实现的强大功能，并节约了开发成本，是一种全新的软件系统构造技术。随着Windows 98/Windows 2000将浏览器技术植入操作系统内部，这种结构更成为当今应用软件的首选体系结构。显然B/S结构应用程序相对于传统的C/S结构应用程序将是巨大的进步。

Single sign-on

Connected to: Authentication OpenID Login

From Wikipedia, the free encyclopedia

Single sign-on (SSO) is a property of access control of multiple related, but independent software systems. With this property a user logs in with a single ID and password to gain access to a connected system or systems without using different usernames or passwords, or in some configurations seamlessly sign on at each system. This is typically accomplished using the Lightweight Directory Access Protocol (LDAP) and stored LDAP databases on servers also called directory servers.[1] A simple version of single sign-on can be achieved over IP networks using cookies but only if the sites share a common DNS parent domain.[2] For clarity it is best to refer to systems requiring authentication for each application but using the same credentials from a directory server as Directory Server Authentication and systems where a single authentication provides access to multiple applications by passing the authentication token seamlessly to configured applications as SSO.

Conversely, single sign-off is the property whereby a single action of signing out terminates access to multiple software systems.

As different applications and resources support different authentication mechanisms, single sign-on must internally translate and store credentials for the different mechanisms, from the credential used for initial authentication.

Other shared authentication schemes not to be confused with SSO include OAuth, OpenID, OpenID Connect and Facebook Connect, which require the user to enter their login credentials each time they access a different site or application.

Benefits

Benefits of using single sign-on include:

Reducing password fatigue from different user name and password combinations

Reducing time spent re-entering passwords for the same identity

Reducing IT costs due to lower number of IT help desk calls about passwords[3]

SSO shares centralized authentication servers that all other applications and systems use for authentication purposes and combines this with techniques to ensure that users do not have to actively enter their credentials more than once.

* 1. SAML and Oauth

1. 用户试图访问SG

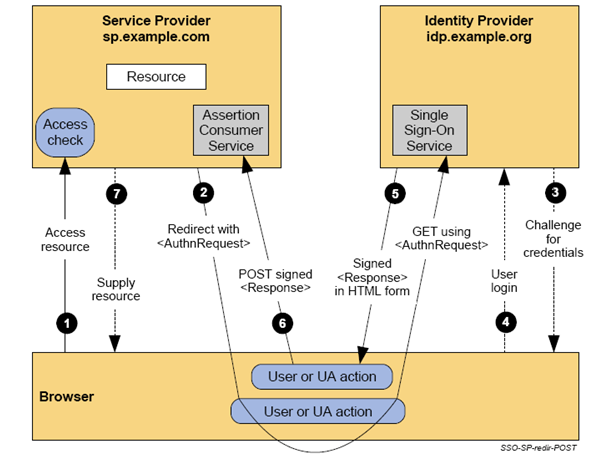
2. SG 发一个AUTHNrequest 到 IDG

3. IDG 要求用户登录

4. 用户登录

5. 登录后把SAML 发到浏览器

6. 浏览器再转发到SG

7. SG 验证SAML 请求后转到SG 应用

SAML 是一种认证协议， 主要用于SAAS的单点登录

SAML 过去， 现在和未来

由OASIS安全委员会制定的SAML, 逐渐成为一个流行的单点登录解决方案。 尽管传统上看SAML很复杂， 要很多花费, 但是有了SSO365 的产品和解决方案，企业不论规模如何， 从大公司到小公司， 都可以瞬间部署SAML 兼容的解决方案。

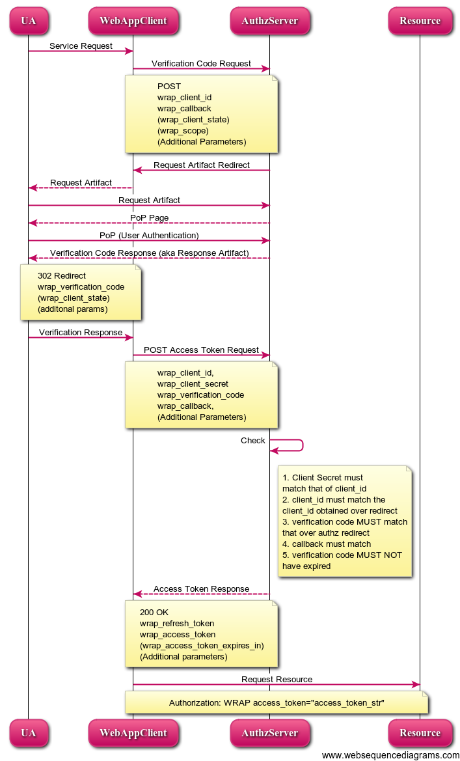
SAML又译作安全断言标记语言， 英文全称是Security Assertion Markup Language， 其实翻译成安全令牌标记语言更合适。 它是一个基于XML的标准，用于在不同的安全域(security domain)之间交换认证和授权数据。与传统的单点登录产品不同， 它主要解决的是跨域的单点登录问题。 比如一个公司的员工可以通过SAML访问saas 应用不需要登录。

如左图所示， SAML标准定义了身份提供者(identity provider)和服务提供者(service provider)，这两者构成了前面所说的不同的安全域。 它主要解决了在IDG 中认证的用户， 可以访问SG 不需要重新验证。 原因很简单， 因为IDG 和SG 有一个互信的关系。

在大多数计算机连入互联网之前，各个系统中像身份验证和授权这类安全服务的实现完全是独立的。因此，执行身份验证所需的全部代码，以及密钥、口令，供授权决 策所用的用户信息，以及授权策略本身均存放于使用这些信息的系统上。最初，系统连接到网络上时情况变化不大。每个系统都是一个孤岛，各系统都要求用户拥有 一个账户才能访问该系统， 我们统称为系统为主（domain centric）的用户认证系统。随着互联网的出现，多台机器作为一个Web站点的主机成为一种普遍现象。但仅仅因为用户要使用不同的机器处理不同的请求而强迫他们多次在网络上进行登录，显然是令人无法接受的。在SAML 产生之前， 人们一直在寻找类似的解决方案。 因为越来越多的应用迁移到云中， 所以这个问题变的日益迫切。为了解决这个问题， 2001年， Sun 等公司联系了一些同行, 如Oracle, CA, Fedelity, Intel, AOL等， 成立了自由联盟（Liberty Alliance）, 试图制定一个标准。2003年， Liberty Alliance将他们制定的标准推荐个OASIS组织安全服务技术委员会(Security Services Technical Committee)， 从而使SAML 成为一个OASIS的标准。 2007年，Gartner认为SAML 已经成为事实上的工业标准。 到目前为止， 全球有70多家支持saml 的产品， 包括微软， IBM， Oracle, CA等。因为大公司关注的是如何整合自己的产品线， 相对兼容性较差。 这样就出现了象SSO365等专注SAML来解决用户身份安全 ， 同时有良好兼容性的公司。

综上所述， 开放成为SAML 流行的一个根本原因。 之前ibm和微软曾经主推过WS-federation， 但是因为它的封闭性，并没有得到众多厂商的支持， 巨人们最终也选择了二者兼而有之。 因为SAML 安全性高， 甚至被选择作为支付的凭证， 在金融，保险和电商得到了很好的应用。

随着移动设备的发展，SAML 也受到了不少的挑战。 因为整个协议是基于XML 的， 对WEB-SERVICE 有很好的支持， 但对基于JSON 的REST 支持较差。 未来的SAML ， 讲充分发挥其安全的一面， 和OAUTH 结合， 衍生出新的应用案例。

 1. UA(用户)试图访问WebAppClient

2. WebAppClient 要求AS 给予授权

3. AS 返回一个Authorization Code

4. 用户用这Authorization Code通过UA 传递给WebAppClient

5. WebAppClient换回一个Access Token

6. WebAppClient使用Access Token 来访问RS

7. RS 通过AS 再次验Access Token

8. RS 开放应用给Web

OAuth 是一种授权协议， 主要是针对移动应用

OAuth 过去， 现在和未来

OAuth得到众多大的社交网络的支持， 如谷歌， 雅虎， 脸书等， OAuth 协议采用了业界最好的Web 认证技术，统一形成一个开放的标准。 SSO365 的各种社交网络接口， 支持最终用户不间断地访问国内和国外的应用。

OAuth是IETF互联网工程任务组2010年制定的一个标准。 同SAML 产生的背景类似， 随着各种各样网站的不断增多, 人们在寻找减少用户名和密码, 同时解决访问授权问题的的解决方案。每个网站都有自己的用户名和码显然是件让人沮丧的事情， 但是， 整个用户信息的共享应该是受控的， 因为隐私的关系， 显然你不会把你的身份信息共享给一个给你的宠物找伴侣的网站， 但是给家庭朋友聚会的组织者就不同了。 用户觉得，比如一个登录到一个大的网站， 如google, yahoo等， 如果能把它作为身份认证机构， 在访问一些小的网站时， 使用同样的身份就好了。 于此同时， 用户应该自己可以决定信息共享的程度， 这就提出了另外一个概念以用户为主(user centric)的认证授权方式。

如左图所示,产生了两个新的概念AS (认证服务器， authorization server), 和 RS (资源服务器, resource server). 用户首先通过应用程序访问AS, 登录， 并询问用户是否允许把上面的用户信息开放给第三方网站RS。 如果用户选择了同意，那么就变得简单了。 AS会先向browser发放一个临时的token， 叫authorization code, 然后传递给应用程序。 应用程序用这个authorization code来换取一个access token， 然后放在header中去访问RS。 RS 收到后， 可以和AS 来重新验证， 也可以直接信任， 从而开放服务给用户。

之前， 人们有Flickr Auth、Google's AuthSub、Yahoo's BBAuth、 Facebook Auth等。2006年， Twitter 提出的 Twitter OpenID标准。 2007年， 人们开始坐下来讨论一个标准， 这样， OAuth core 1.0 就诞生了。 在2008 年，人们把它提交给了 IETF(Internet工程任务组 InternetEngineeringTaskForce )。 IETF又叫互联网工程任务组，成立于1985年底，是全球互联网最具权威的技术标准化组织，主要任务是负责互联网相关技术规范的研发和制定，当前绝大多数国际互联网技术标准出自IETF。这样在， 2010年， OAuth 作为RFC 5849 号标准发布了。2012 年， OAuth 进化成RFC 6749. 2.0 发布了。 OAuth 2.0是OAuth协议的下一版本，但不向后兼容OAuth 1.0. OAuth 2.0关注客户端开发者的简易性，同时为Web应用，桌面应用和手机，和智能家居提供专门的认证流程。

OAuth协议为用户资源的授权提供了一个安全的、开放而又简易的标准。任何第三方都可以使用OAuth认证服务(AS)，任何服务提供商(RS)都可以实现自身的OAuth认证服务，因而OAuth是开放的。业界提供了OAuth的多种实现如PHP、JavaScript，Java，Ruby等各种语言开发包，大大节约了程序员的时间，因而OAuth是简易的。

目前， OAuth 得到了众多社交网站的推崇， 包括国内的QQ, 人人， 新浪微博等。 SSO365的OAuth 解决方案， 可以在认证用户后， 充当AS, 给RS 提供授权令牌Access Token.

随着协议的不断推广， OAuth 一个缺点也越来越明显。 比如对于ACCESS TOKEN， OAuth 并没有严格的定义。 导致各个厂商生成的ACCESS token难以互换使用。 随着时间的推移， 人们在呼唤新的协议如OpenID Connect (注：非OpenID) 等的出现。

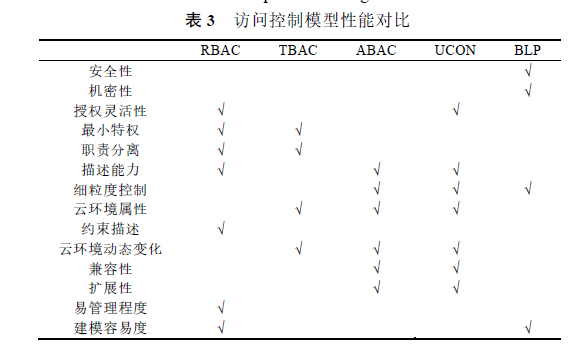
* 1. ABAC and XACML

Attribute-based access control

Connected to: Access control XACML Role-based access control

From Wikipedia, the free encyclopedia

Attribute-based access control (ABAC) defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together. The policies can use any type of attributes (user attributes, resource attributes, environment attribute etc.).[1] Attribute values can be set-valued or atomic-valued. Set-valued attributes contain more than one atomic values. Examples are role, project. Atomic-valued attributes contains only one atomic value. Examples are clearance, sensitivity. Attributes can be compared to static values or to one another thus enabling relation-based access control.



==

XACML

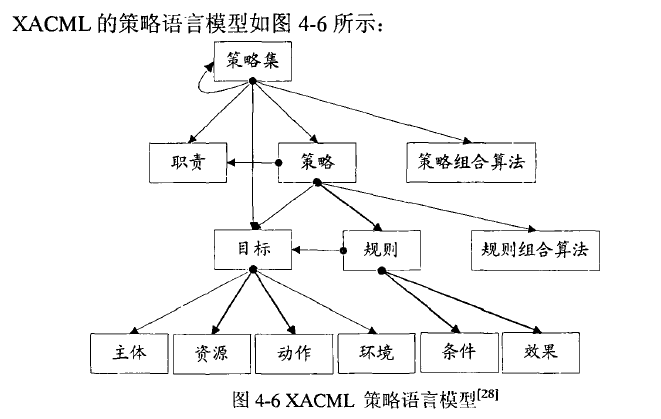
Connected to: Attribute Based Access Control ZXID Standards organization

From Wikipedia, the free encyclopedia

XACML stands for "eXtensible Access Control Markup Language". The standard defines a declarative fine-grained, attribute-based access control policy language,[1] an architecture, and a processing model describing how to evaluate access requests according to the rules defined in policies.

As a published standard specification, one of the goals of XACML is to promote common terminology and interoperability between access control implementations by multiple vendors. XACML is primarily an Attribute Based Access Control system (ABAC), where attributes (bits of data) associated with a user or action or resource are inputs into the decision of whether a given user may access a given resource in a particular way. Role-based access control (RBAC) can also be implemented in XACML as a specialization of ABAC.

The XACML model supports and encourages the separation of the access decision from the point of use. When access decisions are baked into client applications (or based on local machine userids and Access Control Lists (ACLs)), it is very difficult to update the decision criteria when the governing policy changes. When the client is decoupled from the access decision, authorization policies can be updated on the fly and affect all clients immediately.



* 1. REST and SOAP API

Representational state transfer

Connected to: Uniform Resource Identifier Hypertext Transfer Protocol University of California, Irvine

From Wikipedia, the free encyclopedia

In computing, representational state transfer (REST) is the software architectural style of the World Wide Web.[1][2][3] The purpose of REST architecture is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability.[4][1] More precisely, REST is an architectural style consisting of a coordinated set of components, connectors, and data elements within a distributed hypermedia system, where the focus is on component roles and a specific set of interactions between data elements rather than implementation details.[4][1]

The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine.[4][5] REST has been applied to describe desired web architecture, to identify existing problems, to compare alternative solutions and to ensure that protocol extensions would not violate the core constraints that make the web successful. Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI).[6][7]

To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers.[4] REST systems interface with external systems as web resources identified by Uniform Resource Identifiers (URIs), for example /people/tom, which can be operated upon using standard verbs such as GET /people/tom.

The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.[8]

SOAP

Connected to: Web service XML Information Set Simple Mail Transfer Protocol

From Wikipedia, the free encyclopedia

This article's lead section may not adequately summarize key points of its contents. Please consider expanding the lead to provide an accessible overview of all important aspects of the article. Please discuss this issue on the article's talk page. (June 2012)

SOAP, originally an acronym for Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It uses XML Information Set for its message format, and relies on application layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

Characteristics

SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework for web services. This XML-based protocol consists of three parts:

an envelope, which defines the message structure[1] and how to process it

a set of encoding rules for expressing instances of application-defined datatypes

a convention for representing procedure calls and responses

SOAP has three major characteristics:

extensibility (security and WS-routing are among the extensions under development)

neutrality (SOAP can operate over any transport protocol such as HTTP, SMTP, TCP, UDP, or JMS)

independence (SOAP allows for any programming model)

As an example of what SOAP procedures can do, an application can send a SOAP request to a server that has web services enabled—such as a real-estate price database—with the parameters for a search. The server then returns a SOAP response (an XML-formatted document with the resulting data), e.g., prices, location, features. Since the generated data comes in a standardized machine-parsable format, the requesting application can then integrate it directly.

The SOAP architecture consists of several layers of specifications for:

message format

Message Exchange Patterns (MEP)

underlying transport protocol bindings

message processing models

protocol extensibility

SOAP evolved as a successor of XML-RPC, though it borrows its transport and interaction neutrality and the envelope/header/body from elsewhere (probably from WDDX).[citation needed]

* 1. Java Web Application
* Java
* MyBatis

MyBatis is a Java persistence framework that couples objects with stored procedures or SQL statements using an XML descriptor or annotations.

MyBatis is free software that is distributed under the Apache License 2.0.

MyBatis is a fork of iBATIS 3.0 and is maintained by a team that includes the original creators of iBATIS.

* Spring

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an alternative to, replacement for, or even addition to the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.

* MVC

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.[1][2]

Traditionally used for desktop graphical user interfaces (GUIs), this architecture has become extremely popular for designing web applications.

Description

As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system.[3] Particular MVC architectures can vary significantly from the traditional description here.[4]

Components

A typical collaboration of the MVC components.

A typical collaboration of the MVC components.

The central component of MVC, the model, captures the behavior of the application in terms of its problem domain, independent of the user interface.[5]

The model directly manages the data, logic and rules of the application.

A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

The third part, the controller, accepts input and converts it to commands for the model or view.[6]

Interactions

In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.[7]

A model stores data that is retrieved according to commands from the controller and displayed in the view.

A view generates new output to the user based on changes in the model.

A controller can send commands to the model to update the model's state (e.g. editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g. by scrolling through a document).

* Bootstrap

Bootstrap (front-end framework)

Connected to: GitHub Website MIT License

From Wikipedia, the free encyclopedia

Bootstrap (front-end framework)

Bootstrap is a free and open-source front-end library for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications.

Bootstrap is a front end web framework, that is, an interface for the user, unlike the server-side code which resides on the "back end" or server.

Bootstrap is the second most-starred project on GitHub, with over 95 thousand stars and more than 40 thousand forks

* Axis

From Wikipedia, the free encyclopedia

Apache Axis

Apache Axis (Apache eXtensible Interaction System) is an open-source, XML based Web service framework. It consists of a Java and a C++ implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, developers can create interoperable, distributed computing applications. Axis development takes place under the auspices of the Apache Software Foundation.

Axis for Java

When using the Java version of Axis there are two ways to expose Java code as Web service. The easiest one is to use Axis native JWS (Java Web Service) files. Another way is to use custom deployment. Custom deployment enables you to customize resources that should be exposed as Web services.

See also Apache Axis2.

* 1. Clustering and Encryption algorithm

1. Chapter 3 System Analysis

Identity Must Meet the Cloud’s Needs

Five areas of the identity stack must evolve to realize a cloud-scale identity fabric: access control and autho­rization; authentication, federation, and single sign-on (SSO); user account management and provisioning; auditing and compliance; and cloud platform architectural requirements.

Access control and authorization

Managing access control to premises-based and cloud apps is a complex undertaking. In the cloud, outside the firewall, perimeter controls cannot be relied upon to con­trol even binary access. Today, many users are outside the private network and access SaaS apps over the Internet with no need to go through the company network. Autho­rization, therefore, must evolve to a distributed model to support users outside the network firewall.

The authorization in depth concept captures the vary­ing granularity of authorization policies across three levels. Level 1 is a coarse-grained access control policy that governs users’ access to an application or resource. Level 2 is more fine-grained, controlling access to the data level - typically the URL. Level 3 is the most fine-grained level, controlling access to functions or views, sometimes referred to as “entitlements.” Any scalable authorization model must reflect the need to address multiple degrees of granularity or depth. Tying this back to scalability - the greater the granularity, the greater the volume of authori­zation transactions. Another key to scaling access control is grouping access. The earliest forms of access control on the mainframe were based on manually maintained access control lists (ACLs). ACLs initially worked well because few people used the mainframe, but as user bases grew, they became un­wieldy, leading to the use of groups. Group access control scaled well but still required manually managing group memberships. This led to dynamic group management using rules to determine membership and therefore access. Today, organizations use role-based access control (RBAC) instead of groups to reflect the enterprise’s member­ship and attribute-based access control (ABAC) to handle dynamic permissions. In the cloud, these attributes and role memberships are decoupled from the operating system and can be distributed across systems via federation.

Authorization can be scaled by using a distributed, federated model. By breaking the authorization pro­cess down into its core policy elements - management, decision, and enforcement - it is possible to federate authorization across technical and organizational do­mains. Policy management points, policy decision points, and policy enforcement points must run in distributed locations, especially across the cloud.

Identity decision points provide identity data for autho­rization rule evaluation and can utilize Security Assertion Markup Language (SAML) assertions or HTTP headers today and OAuth 2.0 in the future. For example, OAuth leverages a delegated trust model to realize the benefits of abstracting user identity data from user credentials and supports tokenization of authorization. It does require an OAuth-aware architecture of entitlements enforcement. Regardless of the technology used, these authorization decisions must happen quickly and support high volumes of traffic.

Authentication, federation, and SSO

The federation concept is familiar inside the firewall, perhaps best exemplified by the ubiquitous Microsoft Windows domain model. Enterprises can link multiple Windows domains by defining the trust model between different organizations within the firewall and allowing authentication to be delegated to the “local” domain and trusted by a remote domain using Kerberos, making login transparent to the end user.

Modern federation takes this model beyond the pro­prietary Microsoft approach to make seamless SSO work across the Internet using SAML, an XML-based open stan­dard for exchanging authentication and authorization data between security domains - that is, between an identity provider and a service provider - instead of Kerberos.

The problem with federation and SSO is that, after more than a decade, SAML adoption has not risen above 10 percent of enterprise apps - apparently due to the exces­sive costs of infrastructure software. There simply is not enough return on investment for most service providers to implement, expand, and manage a complex federation network. The industry must therefore go beyond SAML and focus on the core HTTP authentication standard. It requires no change to the target app and no coordination between users and the application. HTTP is the gold stan­dard in authentication, with nearly 100 percent of Web apps supporting it.

User account management and provisioning

Today’s apps, even those that are federated, need a local account for user identity management. The challenge is managing data about users, especially routine changes like password resets and account registrations. With cloud-enabling user management, every app performs user management differently and usually does it internal to the application; user management APIs are neither con­sistent nor standardized.

Ideally, developers will use the SAML equivalent for provisioning, the Service Provisioning Markup Language (SPML), but there are only a handful of real-world SPML implementations. Without federated provisioning APIs to enable automated synching of local accounts, SAML adoption will remain limited. There is also a lack of sup­port for integrating SAML attributes for personalization, session context, or just-in-time provisioning. The absence of universal user schemas for directories makes building general-purpose management tools difficult.

Auditing and compliance

A key challenge in auditing in the cloud is overcoming the lack of visibility into user access of SaaS apps. Using the public Internet rather than connecting to a company network puts users beyond the scope of network monitor­ing tools.

Unlike most enterprise networks, the cloud is globally accessible. However, regulatory compliance require­ments vary by jurisdiction and are complex and often contradictory. The industry needs a framework to meet global jurisdictional challenges. Identity is central to such a framework because many regulations center on user privacy and access.

Cloud platform architectural requirements

The cloud has brought with it new architectures and platforms that service providers have yet to make identity-aware. Specifically, many cloud service providers offer storage- or database-as-a-service via hosted hypervi­sors like KVM or those from VMware and Xen, but such IaaSs currently lack identity and access management as a service.

With their high utilization rates, virtualized platforms cannot handle the overhead associated with precloud Web access management (WAM) use of webserver plug-ins and agents. The tight coupling of WAM with plug-ins has proven to be brittle, and the “burstable,” elastic nature of virtualized cloud platforms makes the plug-in model infeasible. The industry requires a proxy-based approach that does not place load on the virtualized Web and ap­plication servers.

In the case of SaaS apps, the identity integration challenge of enforcing access control and supporting audits stems from multitenancy and the fact that the SaaS provider owns and operates the underlying infrastructure, making it impossible to install dedicated agents or plug-ins for each application instance. Also, with most SaaS apps, collecting audit logs is problematic because they are often comingled with other tenant data. In some cases, the audit details are insufficient for answering key forensic ques­tions. What is needed is a loosely coupled, noninvasive identity management platform that can enforce policy upstream from the SaaS apps themselves.

* 1. Architecture Summarize
     1. Function Architecture



2.1 登录

统一身份认证系统对应用提供 SSO（Single Sign On，单点登录）功能，即在多个应用系统中，用户只需登录一次就可访问所有相互信任的应用。

2.2 登出

统一身份认证系统为应用提供 SSO（Single Sign Out，单点登出）功能，即在多个应用系统中，用户只需完成一次登出操作就能登出所有相关联的应用。

2.3 统一认证

应用接入统一身份认证系统后，便可把用户认证模块从应用中解耦出来，相关联的应用之间不再需要交差访问用户，只需通过统一身份认证系统即可完成认证操作，以此实现用户认证的统一化管理。同时，统一身份认证系统支持多协议认证方式（如 SAML、OpenID、

OpenID Connect、Passive STS），以满足多元化应用多样化认证方式的需求。

2.4 与已有认证系统对接

在特定应用场景下，一些部门应用或门户网站提供了独立的认证系统供自己的应用认证，当用户访问权限开放时，第三方应用可直接访问用户接口以实现认证操作。然而，一些安全部门应用或门户网站对诸如用户信息之类的敏感数据把控较严，不对外开放用户访问权限，第三方应用想接入到这些认证系统，则必须修改认证系统以支持安全认证协议，且需要对第三方应用支持的多样化认证方式做匹配，大大提高了认证系统的复杂性和维护难度。

通过统一身份认证系统与已有认证系统的对接，有效的解决了上述问题。第三方应用接入已有认证系统，首先，第三方应用向统一身份认证系统发起认证请求，统一身份认证系统在收到请求后，解析请求信息并寻找对应的IdP（Identity Proveider，即身份提供商），已有的认证系统作为统一身份认证系统的IdP 向统一身份认证系统提供认证信息，接着统一身份认证系统把认证信息返回给第三方应用以实现认证操作。同样，已有认证系统的应用接入统一身份认证系统，首先，应用向已有认证系统发起认证请求，已有认证系统解析请求信息并寻找对于的IdP，统一身份认证系统作为已有认证系统的IdP 为已有认证系统提供认证信息，接着已有认证系统把认证信息返回给应用以实现认证操作。在统一身份认证系统和已有认证系统对接的整个过程中，只需要确定两个认证系统间的安全认证协议，避免了用户数据的交差访问，不仅保证了用户信息的安全，还大大降低了认证系统的复杂性，实现了应用认证的统一化管理。

2.5 多租户支持

统一身份认证系统为企业应用提供多租户支持，不同部门可申请成为统一身

份认证系统的一个租户，每个租户下的应用享有一套完整的统一认证体系，从而

实现多个租户间应用和数据的完全隔离，在节省计算资源的同时为企业应用认证

的统一管理带来便利。

2.6 认证 API

API（Application Programming Interface,应用程序编程接口）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。在统一身份认证系统范畴内，由于需要认证的系统在语言、功能、形式上各有不同，导致接入方式及接入程度也不尽相同。由认证系统提供基本的操作API，如登陆、登出、用户查询等60 余种，被认证方根据自己需求合理整合后接入认证系统。

2.7 策略管理

XACML 是一种用于决定请求/响应的通用访问控制策略语言和执行授权策略的框架，它在传统的分布式环境中被广泛用于访问控制策略的执行。在典型的访问控制框架中，有策略执行点PEP(Policy Enforcement Point)和策略决定点PDP(Policy Decision Point)。PEP 用于表达请求和执行访问控制决定。PDP 从PEP处接受请求，评估适用于该请求的策略，并将授权决定返回给PEP。在同一身份认证系统中，主要完成对XACML 做相应的支持。

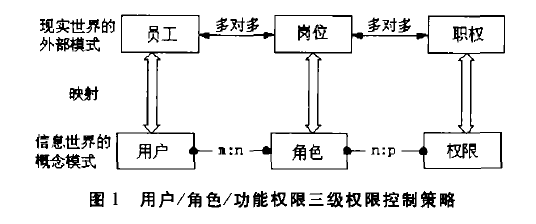
2.8 权限管理

权限管理，一般指根据系统设置的安全规则或者安全策略，用户可以访问而且只能访问自己被授权的资源，不多不少，按用户身份及其所归属的某项定义组来限制用户对某些信息项的访问。在统一身份认证系统中，需要能够对RESTful或者SOAP API 进行权限管理，支持高级权限审计和管理等。在单点登录后能够对访问资源进行一定程度上的管理。

2.9 系统管理与监控

系统管理和监控，需要对统一身份认证系统进行统一的管理与健康，比如支持内置标准的访问和性能统计信息的收集和监控，提供对一些关键度量进行监控和管理，支持企业级日志系统的集成等。

对于在软件系统设计和开发中遇到的重要问题，应该有相关的文档记录



策略存在冲突是怎么，要求提出解决策略冲突算法

* + 1. Description of Module Function
  1. Non-functional Requirements

## 性能需求

为提供良好的用户体验，性能方面需要满足如下需求：

1. 低延时：由于系统需要使用WSO2的登录认证模块并调用WSO2的API，应该尽可能优化保证系统的低延时；
2. 高并发：由于系统在未来可能会有较大的用户量，应该能够支持高并发；
3. 低耦合：在系统的设计中，应该尽可能地降低系统中不同模块的耦合度以提高扩展性；
4. 高内聚：在系统的设计中，应该尽可能地提高系统的子模块的内聚性，遵循单一责任原则；

## 安全性需求

1. 由于系统的用户分为多种角色，不同角色对应的权限也不同，因此不同角色对应的权限不能有任何差错。
2. 系统中的部门信息、人员信息不能被泄露，系统用户的密码不能被破解。

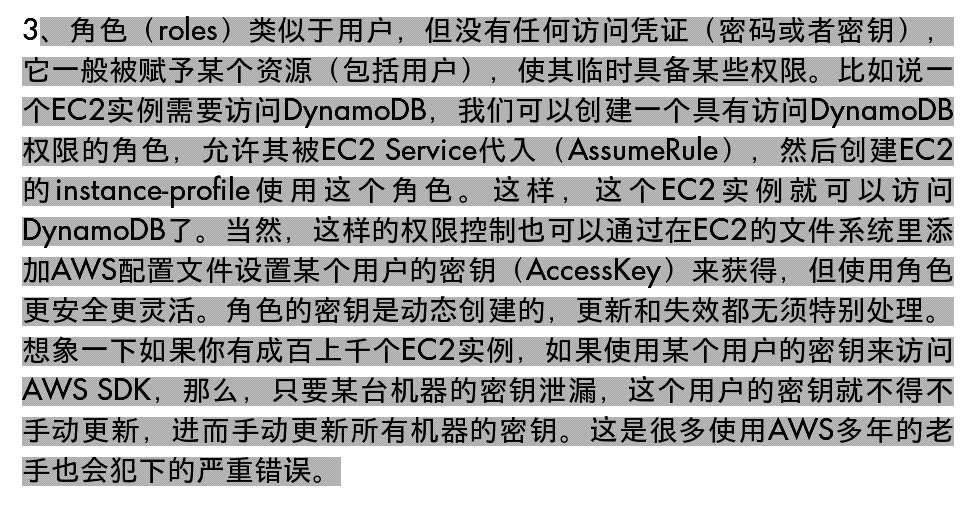
## 质量需求

1. 系统所有功能都能正常使用；
2. 系统中的前端网页所有链接要求都能进入；
3. 在系统上线之后，为了保证良好的用户体验，系统需要不间断运行，因此系统的维护升级需要在用户访问量最小的时候进行。

## 其它需求

1. 在系统开发完成之后，需要编写系统使用手册，提供给不同角色的用户使用；

为什么有加密需求



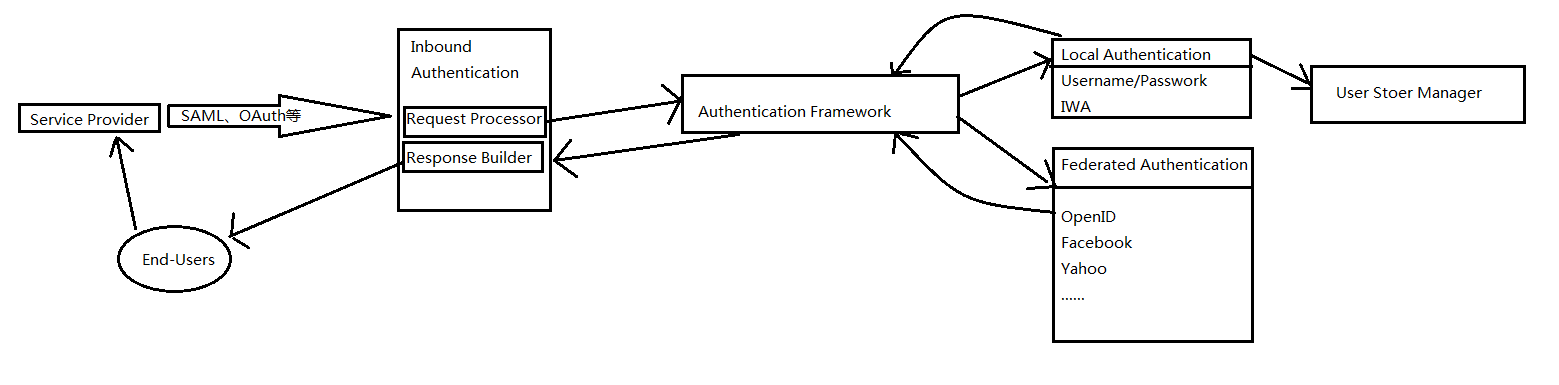
* 1. Development Environment

1. Chapter 4 System Design

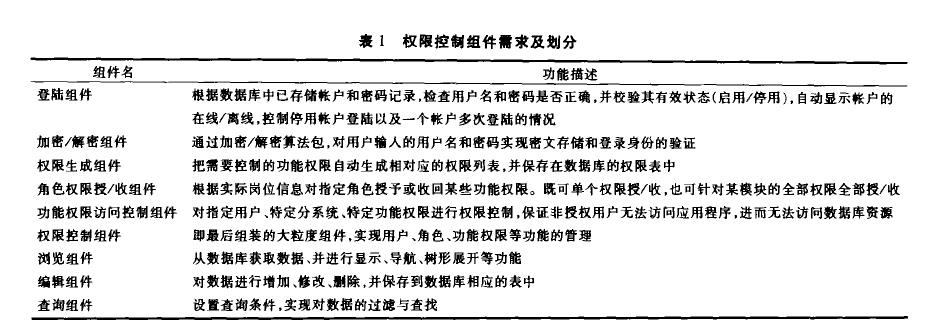


* 1. Preliminary Design
     1. The Architecture and Module

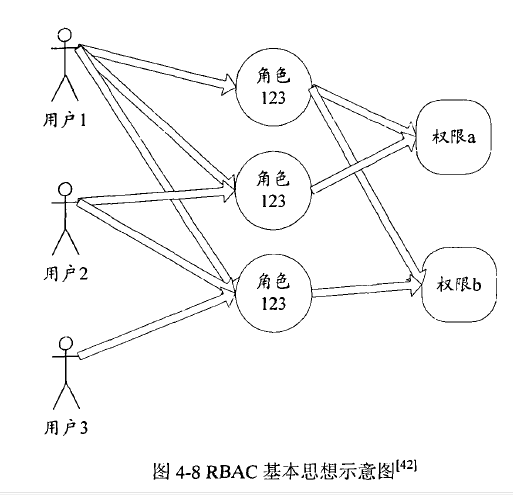
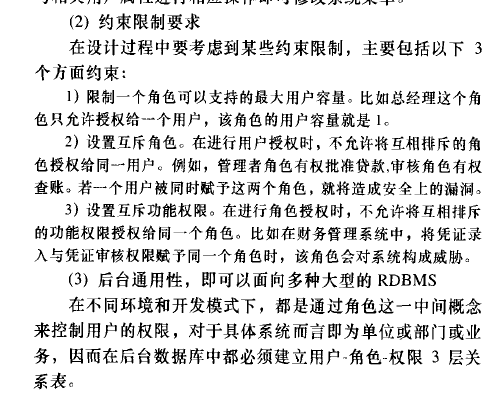
单点登录模型



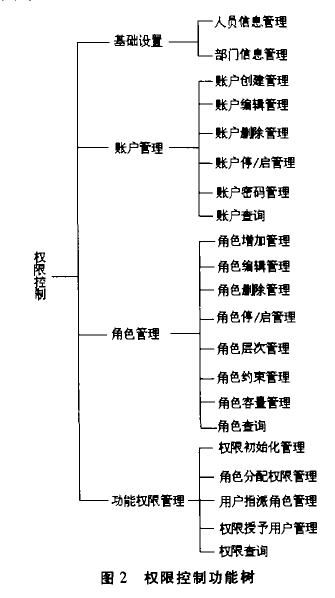
组件设计



* + 1. 用ABAC实现RBAC

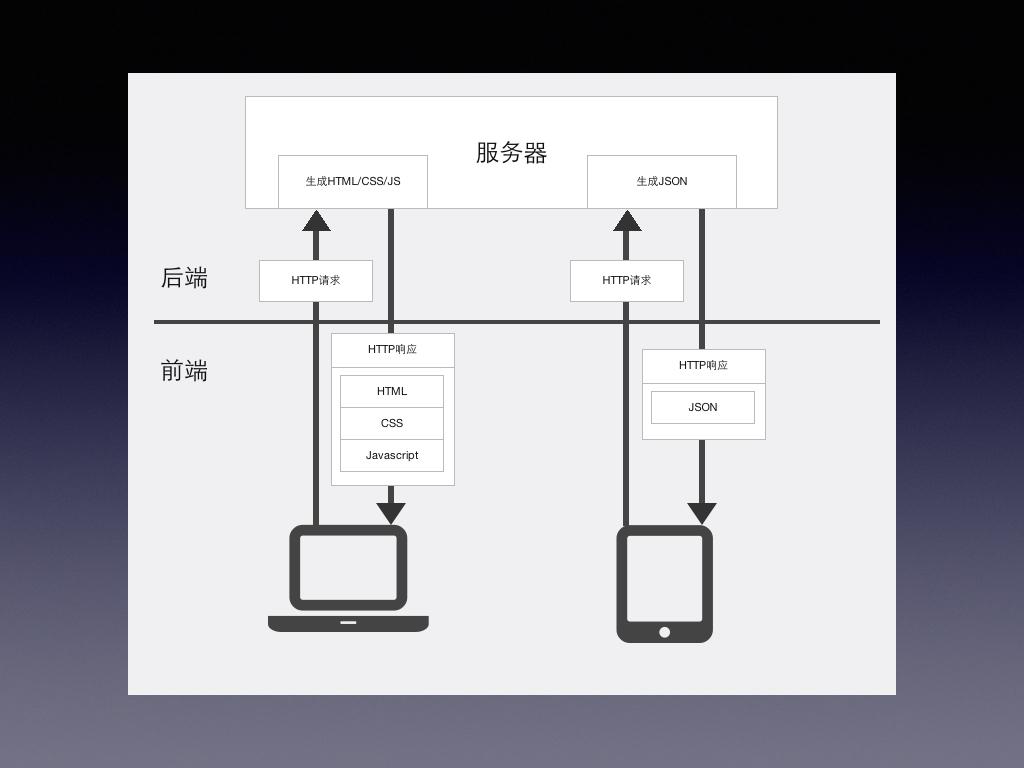
 

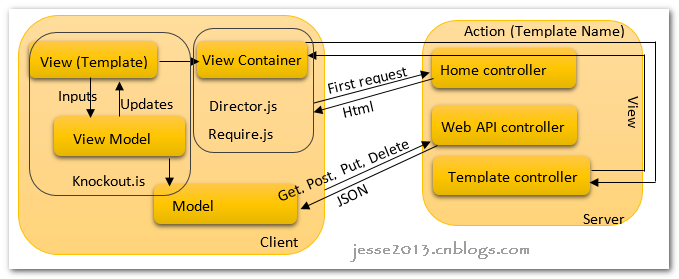
权限控制

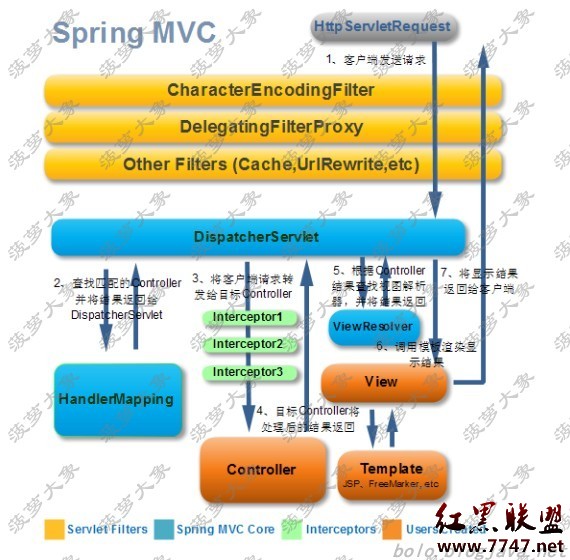


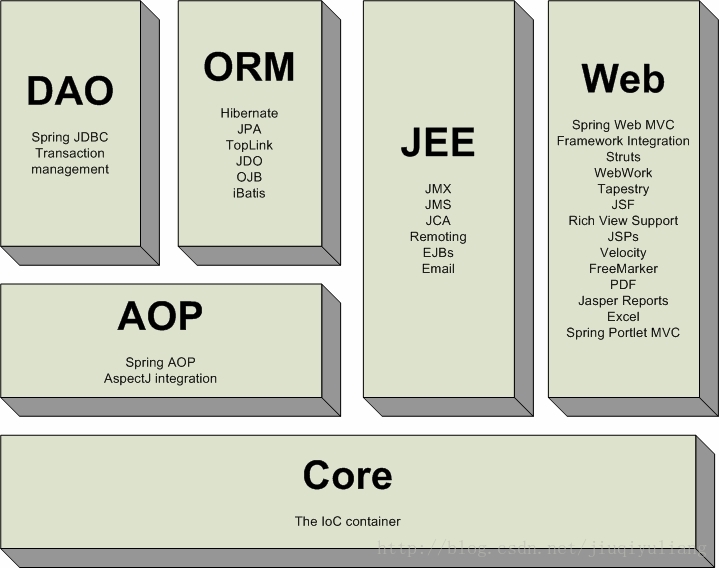
策略冲突时访问控制中授权一致性问题：

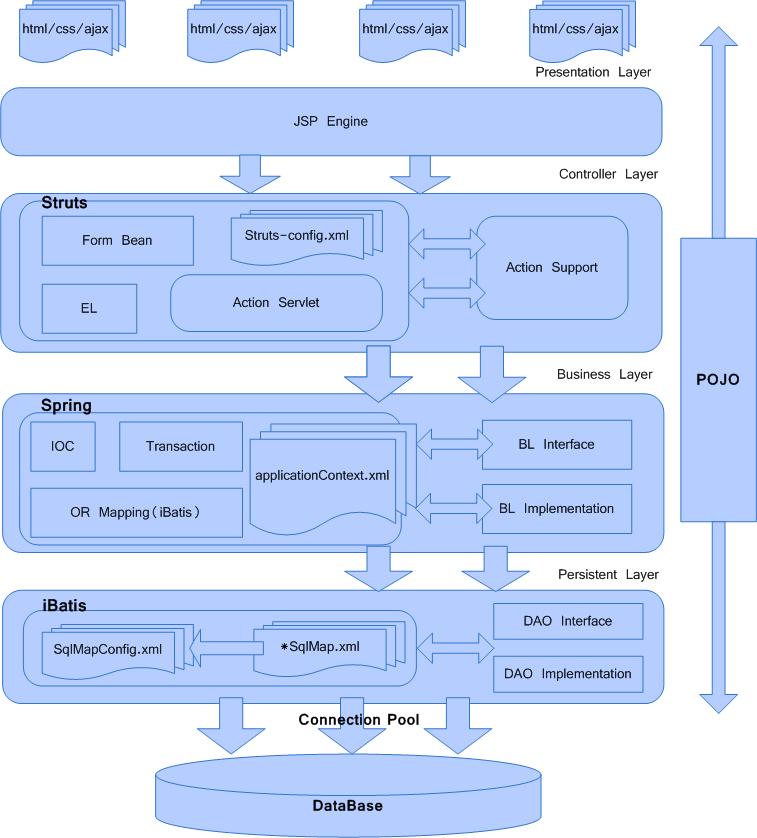
Jave WBE设计

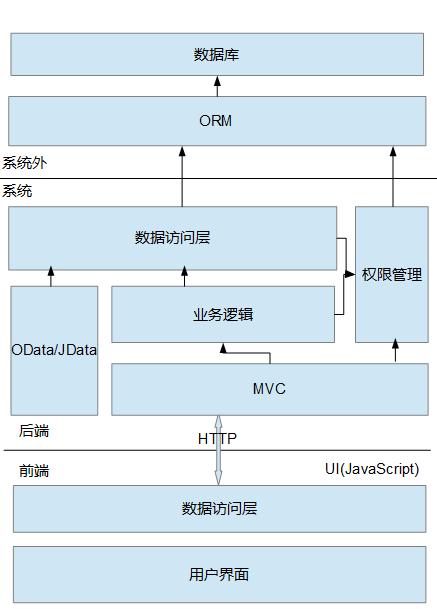








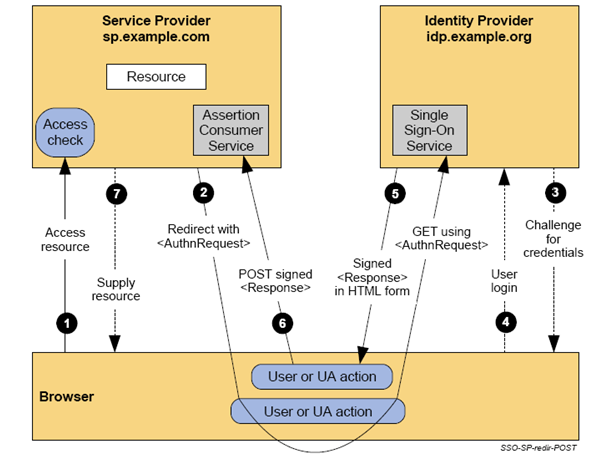




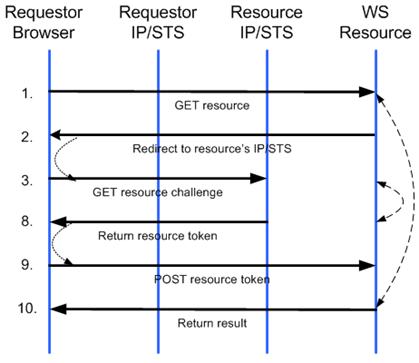
* + 1. Module II‘

租户管理

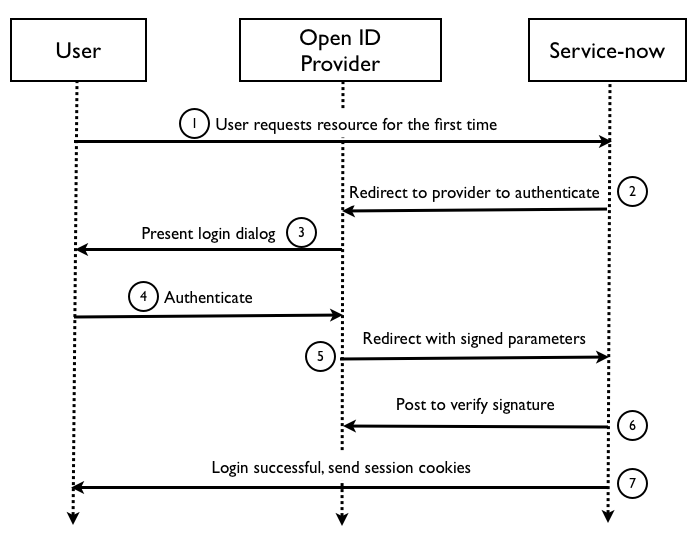




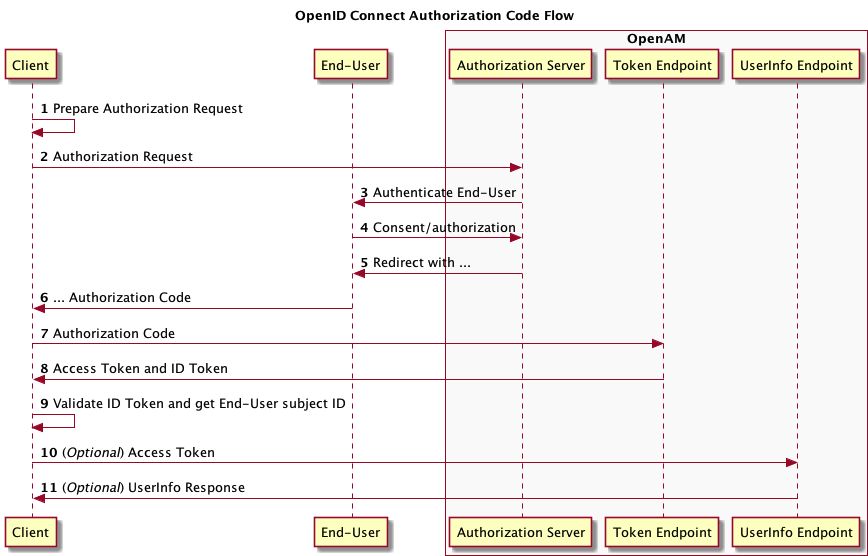
STS认证



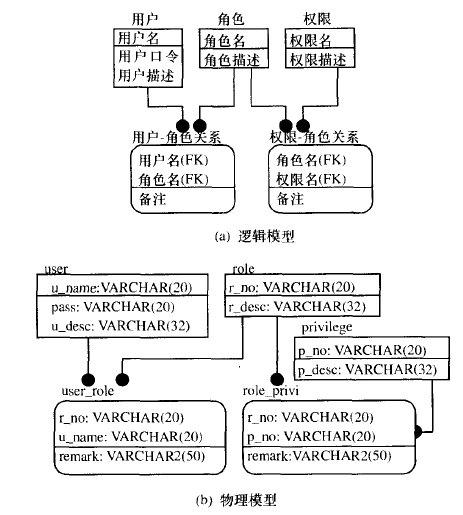
OpenID



OpenID Connect



* 1. Detail Design
     1. Data Structure Design



* + 1. Algorithms Design

1. Chapter 5 System Implementation

测试环境

本系统采用B/S架构组成，软硬件配置如下：

1. 软件配置：

Linux OS（Ubuntu 14.04 LTS）

Jdk 1.7 64bit

Tomcat v7.0

MySQL Community Server 5.7

2. 硬件配置

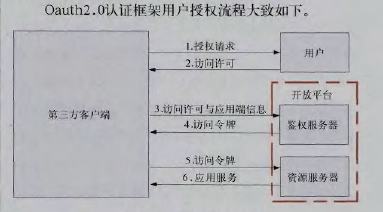
CPU：双核

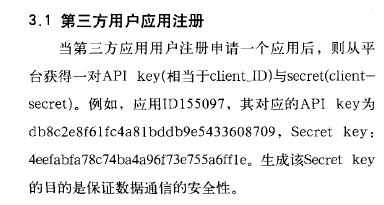
内存：2G

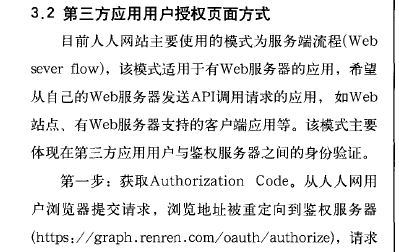
硬盘：70GB空间；

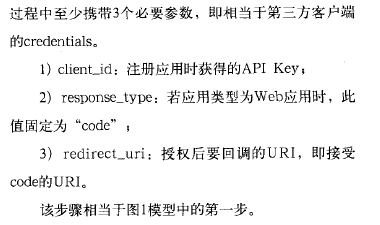
* 1. IDE Configuration

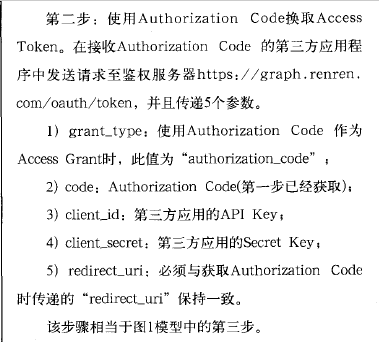
单点登录实例

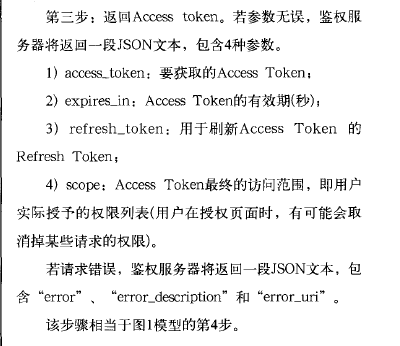












系统开发环境

1 硬件配置: Intel(R) Core(TM) i5-sss7U CPU@1.80GHz + 8.00GB

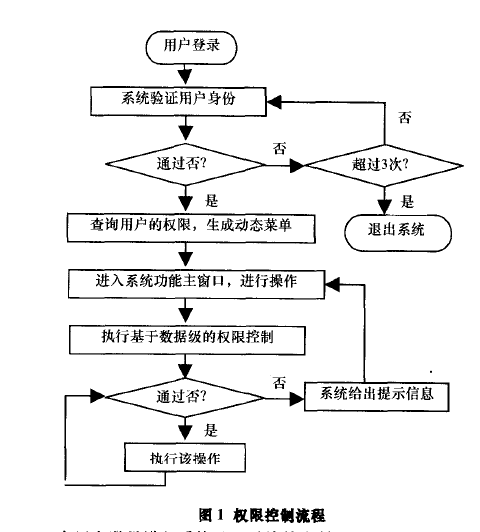
2 操作系统: Windows10 Home 64-bit Operating System

3 开发工具: Intellij IDEA Ultimate 2016.1

4 环境配置: JDK1.6,Ubuntu Server 14.04 LTS

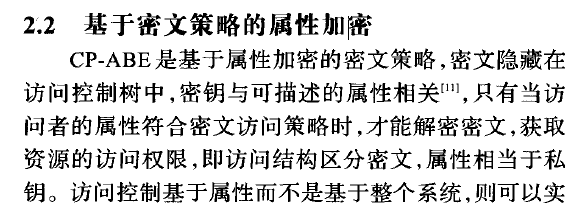
* 1. Core Code and Important Flow
     1. Flow I

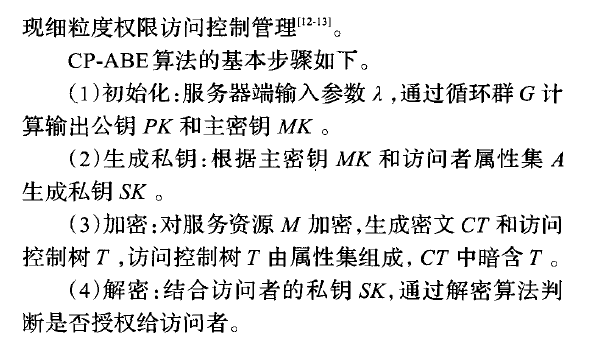
菜单级别的权限控制WEB UI上菜单级别的权限控制

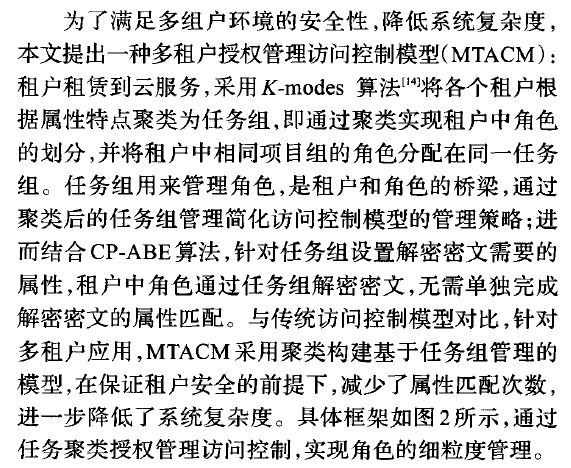


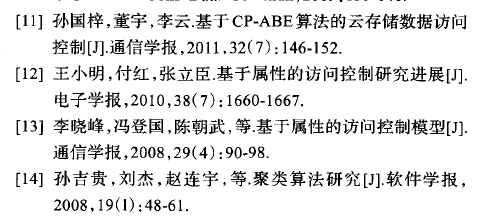
* + 1. Flow II

基于属性加密的算法

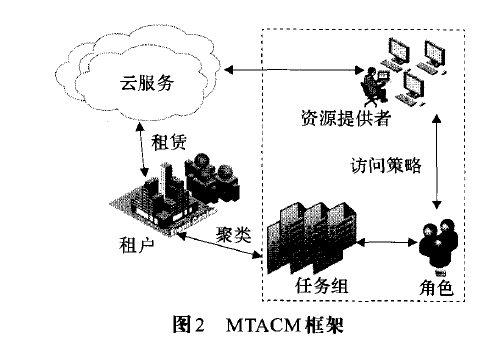


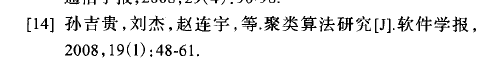


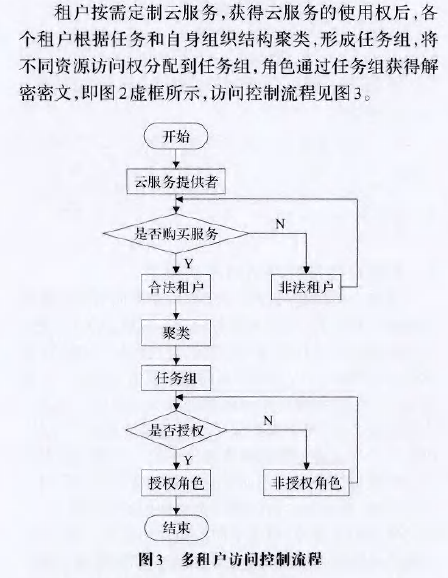




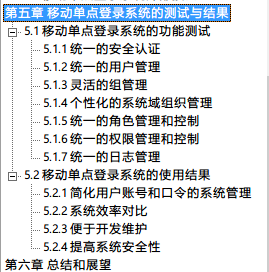
聚类算法







1. Chapter 6 Test and Performance Estimate
   1. Overall Analysis
   2. Test



1. 登录认证模块（以第三方应用的方式接入WSO2进行单点登录）：

直接使用WSO2单点登录界面；

1. 组织架构展示模块

部门和人员信息展示，如下图所示：



图6-5 欢迎界面



图6-6 查询界面



图6-7 添加界面



图6-8 编辑界面

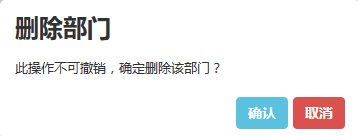


图6-9 删除界面



图6-10 查询界面



图6-11 添加界面



图6-12 编辑界面

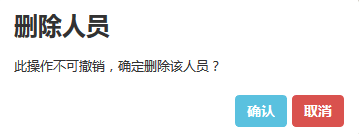


图6-13 删除界面

* 1. Modeling the Performance of System
     1. Concept and Definitions
     2. Model Assumptions
     3. Modeling the Performance

1. Chapter 7 Conclusion

Identity Must Integrate, Extend, and Abstract

Two core aspects of a cloud-scale identity fabric are a one-to-many integration model and the extension of ben­efits via the *network effect*. There is also a need to abstract identity through externalization and the ubiquitous adop­tion of open standards.

Identity integration

Integrating identity infrastructure and apps requires evolving to a hub-and-spoke model. Today, each app requires a different user account. However, building fed­erated identity connections on a one-to-one basis does not scale. The only way to scale is to move to a one-to-many model that includes the number of user identities in each application.

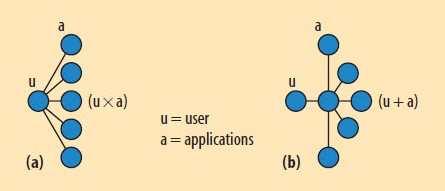
The problem can be broken down mathematically. As Figure 1a shows, the one-to-one model of a user having individual credentials in an app can be represented as

number of credentials = (number of users × number of apps).

An organization’s integration of apps can be represented as

number of connections = (company × number of integrated apps).

When graphed, these equations scale linearly, mean­ing that for each new connection made or app deployed there is a direct corresponding increase in the amount of credentials or integration work required. Increasing the number of users and the number of apps simultaneously results in exponential growth.



**Figure 1. Integrating identity infrastructure and apps requires evolving from (a) a one-to-one model, in which number of credentials = (number of users × number of apps), to (b) a one-to-many model, in which number of credentials = (number of users + number of apps).**

For example, consider an enterprise with 10,000 users (customers, employees, and partners) that access 15 apps. In a one-to-one model, this requires 150,000 credentials (passwords). Resetting each credential once a year via a $30 help desk call would entail $4.5 million annually in management expense. Assuming licensing, deployment, integration, and maintenance costs of $50,000 per connec­tion for the same 15 apps, the integration expense would be $750,000.

While the one-to-one model may work on a small scale, it is not sustainable. Most complex systems that start with a one-to-one architecture evolve to a more scalable one-to-many alternative. For example, stock markets have exchanges and clearing houses that connect millions of users and transactions using a hub-and-spoke model; likewise, travel reservation systems like Expedia provide a one-to-many model to connect travelers with multiple airlines.

To scale the number of connections and limit the prolif­eration of credentials in a federated identity, cloud service providers must likewise move to a one-to-many model. They should not integrate directly with apps in the cloud for SSO and access, creating redundant accounts and credentials, but instead federate with a fabric pre-integrated with multiple apps.

As Figure 1b shows, a one-to-many model yields

number of credentials = (number of users + number of apps)

and

number of connections = (company + number of integrated apps).

Applying these formulas to the previous example of 10,000 users would produce only 10,000 credentials, a 93 percent reduction compared to the one-to-one model. This would result in a management cost of only $31,500 per year and, with just one connection from the organiza­tion to the fabric, an annual integration expense of only $50,000.

Beyond the cost savings, cutting credentials by more than 90 percent brings material security and risk ben­efits. In addition, moving to a hub-and-spoke architecture reduces the number of components that could fail, result­ing in greater reliability.

Identity network effect

A network’s value increases as more people use it and it expands in reach. The classic example of this kind of posi­tive network effect is the telephone: the more people who owned one, the more valuable it became to each owner.

As more users and apps are integrated in the identity network, these benefits extend to other network members simply by virtue of their being connected. This is possible because the identity fabric serves as an identity integration broker, securely shared across the network. As the fabric is upgraded, the upgrade benefit is realized across the cloud.

Examples of the network effect can be found through­out the cloud. Consider Apple’s iTunes. When the company introduced its App Store on iTunes, millions of users already accustomed to buying music through iTunes could quickly acquire apps for any iPhone, iPod, or iPad on the network. Since its debut in July 2008, customers have downloaded more than 7 billion apps through the App Store ([see this link](http://en.wikipedia.org/wiki/App_Store)), providing Apple with significant revenue. Moreover, the vast major­ity of purchases occur through self-services and without IT intervention, and developers contribute new apps rather than Apple building them.

Similarly, the network effect dramatically benefits cloud identity. As providers add new SaaS apps to the fabric through federated SSO and provisioning, any network member can leverage that integration without extra effort due to the one-to-many approach. By distributing integra­tion through self-service capabilities, the identity fabric becomes exponentially more valuable as more connections are made between the fabric and apps.

Another example of the network effect’s positive impact on identity management is Google’s implementation of strong two-factor authentication for Google Apps. In September 2010, Google let enterprises protect their accounts by delivering to users a one-time code on mobile phones (via SMS) as an additional authentication factor along with their password.

This was a great improvement in security intended to reduce phishing attacks and other password weaknesses, but on its own, it benefits only Google Apps users. When combined with the identity fabric, however, the benefit of strong authentication can expand across the network. Specifically, if users’ authentication to the identity fabric originates at Google with two-factor authentication, then this trusted session can be federated to other apps on the network, even those without a need for strong authentication. By integrating and leveraging the strong authentication capabilities of Google through federation with the identity fabric, cloud service providers save them­selves the cost and effort of deploying and managing a strong authentication infrastructure.

Abstraction

Realizing a cloud-scale identity fabric requires abstract­ing identity into identity services. App developers have historically built identity into the app itself, maintain­ing a local user store for authentication. This results in redundant and often stale data, password proliferation, and greater help desk costs.

During the past 10 years, apps have begun to external­ize identity starting with leveraging external directories that use the Lightweight Directory Access Protocol (LDAP) to authenticate users centrally. This has been a great step forward for scaling identity management, but it must go further - LDAP password authentication is not enough. Enterprises must be able to use more than one type of authentication depending on the level of risk associated with an app.

**Externalizing identity.**By externalizing all key identity functions for Web apps in either public or private clouds, developers can focus on improving their apps, and enter­prises can manage identity across multiple apps more efficiently:

* Access control can be externalized to a network proxy instead of being performed within the local app.
* Federation and authentication can be externalized from the Web app to the webserver or proxy, where the app trusts the authentication service to hand it an authenticated user ID, usually through HTTP headers.
* Auditing can be externalized using a proxy to cen­trally log activity and aggregation tools to report activity.
* User management can be externalized by leverag­ing an external user directory like LDAP instead of an internal user database. This can also be done by exposing user management APIs to external man­agement systems, ideally using a standard interface like SPML. Also, using commonly defined directory/user schemas would streamline external identity management.
* With new cloud-designed apps, authorization and entitlements can be externalized using the emerg­ing claims-based model, wherein federated partners provide user and transaction attributes in HTTP head­ers or tokens for authorization.

**Standards.**The challenge in externalizing identity is the degree to which the existing app must be modified and whether changes would change its behavior. Change is expensive, and many developers have difficulty justifying “plumbing” improvements over features and enhance­ments customers are willing to buy. This underscores the need for standards that streamline how developers exter­nalize identity so that they need only do it once to work with many identity management technologies.

To see that open standards are the proven way to achieve ubiquity, we need only look at where HTML, IP, and SSL have taken the Internet. Identity is no stranger to standards, as evidenced by LDAP, x509, and HTTP au­thentication. Existing cloud identity standards are well designed and reference reliable implementations, including

* SAML for federated SSO,
* SPML for federated account management and provi­sioning, and
* XACML (Extensible Access Control Markup Language) for federated authorization and access control.

The problem with these and other cloud-scale identity standards relates to prioritization and adoption. Consider buying a computer with an electrical plug that you can only use in 5-10 percent of outlets - this is the rate of SAML adoption today. SPML and XACML are even further behind. To be relied upon, adoption of standards must be ubiquitous.

Fewer users will ask for federated SAML SSO than for a new feature or a mobile version of a particular app. In addi­tion, recovering the cost of implementing SAML federation software, which can exceed $100,000, is difficult. It is thus not surprising that improvements in identity infrastructure lag. The adoption solution is twofold: first, leverage open source tools like Fedlet to lower the cost of implementing a standard; and second, leverage the identity fabric to pro­vide these identity services in an easily consumable way, minimizing effort and cost.

Identity Infrastructure as a Service

According to Nicholas Carr’s analysis of IT’s evolution into a utility[[4]](https://www.infoq.com/articles/architecting-cloud-scale-identity" \l "_ftn4_6484), infrastructure-like power generation began with a dedicated model: only organizations with large amounts of capital could afford to build and operate their own waterwheels and, later, electrical generators. This infrastructure gradually became centralized at a utility - a power plant or telephone company, for instance - due to economic forces and democratization. Finally, the estab­lishment of distribution standards and a delivery network led to wide adoption of a utility service, resulting in econo­mies of scale, cost reductions, and new capabilities.

Identity management for the cloud must also evolve to the point of being standardized and accessible by multiple applications and users. Why must it be built differently than conventional single-tenant, behind-the-firewall identity management software? Nobody has success­fully retrofitted cloud constructs and models onto legacy software, which was created with a different set of assump­tions. To be successful in the cloud era, organizations and vendors must fundamentally rethink how they manage, deliver, and consume identity.

The concept of *identity infrastructure as a service*fol­lows two major trends: the evolution of IT from capital infrastructure to a service, and the consumerization of IT.

Identity as a service

Rather than investing heavily in identity when developing or using an app, it makes more sense for an organization to utilize a service for its identity needs. Iden­tity as an infrastructure presents an on-demand model that delivers the right amount of capability at the right time. Companies need to think less about identity technology and focus instead on service-level agreements and ser­vice management, not infrastructure. This means moving from a company-owned to a service-provider-owned and -operated identity management approach.

An identity management solution that requires making a capital investment to use a cloud service does not reflect this shift. If each person had to buy a cell phone tower just to use a mobile phone, how many people would have mobile phones? Mobile phones are ubiquitous because users do not have to buy any infrastructure to use the net­work. They simply pay a subscription for the service and get access to the entire shared cellular infrastructure.

Consider the Internet’s Domain Name System (DNS) backbone. Major DNS name services provide these services transparently and reliably. How the provider actually runs the name servers is irrelevant. All that matters is that name services are readily available and perform as expected. Identity services must be as transparent and reliable as DNS services.

Consumerization

Each of us uses sophisticated, consumer-based Web apps every day in our personal lives. For example, millions of users access Amazon to search for and purchase items; hundreds of millions of people connect via Facebook; and iTunes offers millions of songs and thousands of TV shows, movies, podcasts, and audiobooks for downloading. Our experience with such sites leads us to expect enterprise systems to be just as simple, effortless, and reliable. More­over, we all prefer a “freemium” model that lets us try out a service without a commitment or investment.

We also like to do things ourselves. We generally want to use apps without IT’s help and with a minimum of administration, and we buy technology like smartphones and laptops at the grassroots level, not centrally from a company. Anyone can set up a rented development envi­ronment with a credit card and the same Amazon account they use to buy books and other goods. The same goes for identity—the line between professional and personal per­sonas has blurred. When you use Twitter or Facebook for work, for example, are you an employee or a consumer? It is often very difficult to distinguish between the two.

Cloud service providers will deliver better identity solu­tions to their customers by utilizing a specialist’s expertise rather than building it themselves. They must build scal­ability into every design decision from the beginning. Too often, scalability is an afterthought or a reaction to an overloaded system. Developers set performance ex­pectations when they build an app, and if it meets those expectations, it is deemed scalable. But problems arise when expectations change, as is the case with an unex­pectedly successful viral adoption or when porting an app server to the cloud. Using a hypervisor to abstract compute power from the app is a simple way to scale up, but most apps are not architected to readily leverage the *N*+1 scale that clouds offer.

By consolidating identity infrastructure, service pro­viders can achieve economies of scale. As with solid-state drives, the number of moving parts is reduced, resulting in greater uptime and reliability. Each identity integration point becomes a stress point, and each credential creates a broader attack surface and potential help desk expense.

Looking Ahead

Many large-scale identity ecosystems have the disrup­tive potential to accelerate change in cloud identity. Google Apps, which launched in March 2010, acquired more than 27 million users in its first month[[5]](https://www.infoq.com/articles/architecting-cloud-scale-identity" \l "_ftn5_6484); obviously, that number is much higher today. Other examples of such systems in­clude Twitter, which has 106 million registered users and is adding more than 300,000 per day[[6]](https://www.infoq.com/articles/architecting-cloud-scale-identity" \l "_ftn6_6484), and Google Gmail, which boasts more than 170 million monthly users[[7]](https://www.infoq.com/articles/architecting-cloud-scale-identity" \l "_ftn7_6484).

Facebook has exploded in popularity, with more than 550 million users—more identities than the combined populations of the US, Canada, the UK, and Italy[[8]](https://www.infoq.com/articles/architecting-cloud-scale-identity" \l "_ftn8_6484). This massive number of identities can be integrated with the identity fabric en masse due to the walled-garden aspect of Facebook’s platform and user community. When Face­book introduced support for identity sharing via OpenID in 2009, hundreds of millions of people suddenly had OpenID credentials.

When Google decided to support OpenID a year earlier, it brought more than 100 million users into the market overnight. The implication for cloud identity is that con­sumer authentication models that have been proven to scale to hundreds of millions will be part of the identity fabric and give it critical mass and scale. These examples clearly indicate that an identity access fabric linking enter­prises to the cloud is not only relevant but also necessary.

Cloud growth can be compared to a city that builds homes at an explosive pace without building the infrastructure of roadways needed to support the influx of traffic. In the case of identity management, the cloud has not kept pace with the enormous volume of user identities that network administrators must manage and secure.

To realize the cloud’s benefits, enterprises must have an identity infrastructure in place that can overcome the limitations of precloud identity architectures. This means using an identity fabric that links many apps to a single identity. The proliferation of identities demands a better identity management solution, not only to ease the burden on IT administrators’ time to manage them but also to address security and privacy concerns.

An identity fabric that provides a secure linkage be­tween the enterprise and the cloud, while reducing the number of identities, is the clear answer to enabling full-scale cloud adoption. Cloud-based identity management delivered as an infrastructure service with on-demand dial-tone quality benefits users, network administrators, application vendors, and service providers in dramatic ways. The cloud, with all its ubiquitous technology, must also make the identity management fabric ubiquitous, as its growth and acceptance depend on it.

单点登录系统于 2006 年 12 月投入使用。通过实施建立企业级的单点登录系统

和安全防护系统，为公司用户提供统一的信息资源认证访问入口，建立统一的、基

于角色的和个性化的信息访问、集成平台；通过实施单点登录功能，使用户只需一

次登录就可以根据相关的规则去访问不同的应用系统，提高信息系统的易用性、安

全性和稳定性。单点登录 SSO 实现了“一人一个身份标识”登录所有后台系统的管

理模式。通过稳步、高效地整合和安全快捷的身份认证与授权机制，使得原本疲于

处理纷繁冗杂的系统登录与信息安全问题的过程变得安全且简单。

整个单点登录系统的建设是一个庞大的工程。除了要设计和开发单点登录系统

之外，现有的各运营支撑系统也必须与之配合进行改造，特别是要改变现有系统客

户端的运行平台，工作量非常巨大。

就单点登录平台本身，主要的难点在于各模块的具体设计，如何保证用户管理

模块与权限管理模块之间的协调工作，如何保证用户票据的安全性，如何在统一登

录平台实现对各服务系统地址的有效切换都是本系统实施过程中的难点问题。

在整个单点登录系统的实施过程中，我们着重考虑了系统的稳定性，安全性，

可扩展性和平台无关性，确保了此系统上线后运行的稳定。

总之，现有或新的 Web 化支撑系统若需要实现单点登录，那么它们只需要集成

一个验证客户端的应用模块，完成后，对于这些系统的登录都可以被单点登录方式

取代了。

只要单点登录系统中的访问权限列表实现了对于客服 CCS 等支撑系统的权限控

制，那么集成单点登录模块对于各系统本身的运行没有任何影响，而对于用户使用

来说，由于只需要记忆一个用户名和密码就可以通行于所有被授权使用的系统，使

用系统将会更加方便和高效。

显得非常脆弱，任何一个入侵者可以轻易的通过口令侦查就可以获得对资源的访问

权限。

因此，信息社会的到来使人们在享受信息资源所带来的巨大利益的同时，也面

临着信息安全的严峻考验。统一身份认证是整个网络安全体系的基础层，否则安全

问题将成为隐患，随时威胁着系统正常运行。只有建立了基于整个网络的、全面的

统一身份认证机制，才能彻底解决这一隐患，同时也为信息化的各项应用系统提供

安全可靠的保证。采用单点登录(SSO)就是很好的解决方案，因为有统一的信息访问

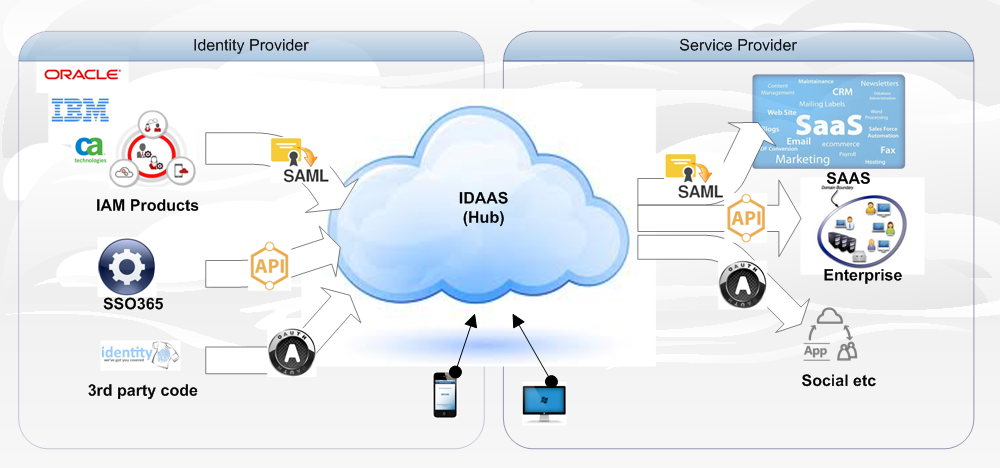
渠道，通过将各种相对分散独立的信息系统组成一个统一的整体，使用户能够从统

一的渠道访问其所需的系统与信息。

本系统目前只是对 BOSS,CCS，B-BOSS 等常用系统实施了单点登录，今后投诉保

障系统，办公自动化系统及各应用子系统将随 WEB 化的进程陆续集成到 SSO 中。

IDAAS展望



Reference

1.王电钢. 基于LDAP目录服务的单点登录系统的研究与实现[D].四川大学,2004.

2.高焕芝. 单点登录技术的研究[D].北京邮电大学,2006.

3.罗坤. 基于RBAC的Java通用权限控制框架研究与设计[D].电子科技大学,2010.

4.程小飞. 基于REST架构的Web Services的研究与设计[D].武汉理工大学,2010.

5.吴敏. Web Services访问控制机制及其整合研究[D].东华大学,2006.

6.刘铁钢. 基于XACML的多租户访问控制的研究与应用[D].内蒙古大学,2014.

7.师绍秋. 面向云计算的RESTful信息资源池的设计与实现[D].上海交通大学,2014.

8.Laferriere et al., "Authentication and Authorization Techniques in Distributed Systems", Carnhan Conf. on Security Technlogy, pp. 164-170, 1993.

9.Bryant, "Designing an Authentication System: a Dialog in Four Scenes", pp. 1-18, http://web.mit.edu/kerberos/www/dialogue.html., Dec. 1988.

10. Ravi S. Sandhu, Edward J. Coyne Hal, L. Feinstein Charles. Role-Based Access Control Models[J]. IEEE, 1996, Issue No.02, 38-47

一定要出现

1. Vincent C. Hu David Ferraiolo, Guide to Attribute Based Access Control (ABAC) Definition and Considerations [J], NIST Special Publication 800-162, January 2014.

基于云服务的云南省电子政务生态系统研究\_周晓琦

策略冲突解决 访问控制中授权一致性问题的研究\_李赤松

云计算访问控制技术研究综述

Acknowledgements

First and foremost, I want to express my heartfelt thankfulness and appreciation to Ms. Chunyan Han who is my supervisor in Northeastern University. It is her guidance that makes my poor paper a beautiful and normative one. I really can’t imagine how many difficulties I would have encountered without her help. Besides, I’m also impressed by her religious and earnest style of study. I think I will be influenced by her for a very long time.

Second, I sincerely thank Professor Chunping Li who is my advisor in Tsinghua University. He helped me so much while I was in Beijing doing research in his library. He will be my advisor in the future for another two or three years when I study in Tsinghua University for my master degree. I’m looking for the days when I’ll be there study from him again.

Besides, I give my everlasting gratefulness to my parents, especially for my mother. She spares nothing to do everything for me. She is the person who loves me the most in the world!

At last, I give my appreciation to all my good friends and my girl friend, for their company and encouragement during these four years in Northeastern University. I want to let all of you know that I appreciate it and won’t forget it forever and never.