构建和实现单点登录解决方案

将一个开放源码的基于 Java 的身份验证组件集成进 Web 门户中

在现有的应用程序中实现单点登录解决方案(single sign-on, SSO, 即登录一次, 就可以向所有网络资源验证用户的身份)是非常困难的, 但是在构建复杂的门户时, 每个开发人员都要面对这个问题。因为门户需要与后端资源集成, 而每个后端资源都有自己的身份验证需求, 所以门户常常必须向用户提供单点登录特性。在本文中, Chris Dunne 一步步地描述了他为一个 Web 门户构建单点登录解决方案的经历。他将讲解如何设置一个开放源码解决方案(来自 Yale University 的 Central Authentication Service), 以及如何将它扩展为根据 Microsoft Active Directory 基础设施进行身份验证。

Chris Dunne, 技术主管, Big Picture Software

2007 年 9 月 19 日

内容

0 评论:

我在自己的工作中发现, 对各种门户应用程序的需求正在增长。门户的技术和功能性需求变得越来越复杂了。尽管出现了可以构建简单门户的工具, 但是门户与远程或遗留数据源的集成问题仍然不容易解决。其中一个问题就是身份验证。

身份验证是个复杂的问题。门户需要向后端数据源和应用程序验证用户的身份, 但是这些应用程序可能具有互不相同的底层安全基础设施。理想的最高效的身份验证解决方案是*单点登录(single sign-on, SSO)*解决方案;在这种解决方案中, 用户只需要登录一次, 就可以向所有网络资源验证他的身份。

最近, 在构建一个需要 SSO 的教育门户时, 我研究了许多商业的和开放源码的 SSO 解决方案。在本文中, 我将一步步地介绍使用免费的 SSO 实现(来自 Yale University 的 CAS)构建简单 SSO 系统的过程。

为什么要选择单点登录?

有多少人实现过自己的身份验证机制(常常是某种简单的数据库查询)?您是否常常考虑创建和管理用户帐号所需的工作流?在任何开发项目中, 身份验证都是很常见的任务。如果幸运的话, 公司已经有一些通用的身份验证类或库。但是, 这个任务常常被忽视, 被当作只在后台发生的微不足道的事情。

一般来说, 公司往往没有一致的身份验证策略或可靠的身份验证框架。随着时间的推移, 这会导致大量应用程序具有自己的身份验证需求和用户存储库。每个人都需要记住多个用户名和密码, 才能访问网络上的不同应用程序。这给管理和支持部门带来很大的负担 —— 必须在每个应用程序中为每个职员设置帐号, 当用户忘记密码时还要帮助他们解决问题, 等等。

身份验证是多种应用程序、平台和基础设施共有的需求。一般来说, 一个用户应该不需要多个用户名。理想情况下, 他应该只需要证明自己的身份一次, 然后就能够访问所有已经得到授权的网络资源。

SSO 的目标是, 让用户能够通过一次登录访问所有应用程序。它提供一个统一的机制来管理用户的身份验证, 并实现业务规则来决定用户对应用程序和数据的访问。

在讨论单点登录的技术细节之前, 先谈谈单点登录的一些好处和风险。好处包括:

- *提高用户的效率*。用户不再被多次登录困扰, 也不需要记住多个 ID 和密码。另外, 用户忘记密码并求助于支持人员的情况也会减少。
- *提高开发人员的效率*。SSO 为开发人员提供了一个通用的身份验证框架。实际上, 如果 SSO 机制是独立的, 那么开发人员就完全不需要为身份验证操心。他们可以假设, 只要对应用程序的请求附带一个用户名, 身份验证就已经完成了。
- *简化管理*。如果应用程序加入了单点登录协议, 管理用户帐号的负担就会减轻。简化的程度取决于应用程序, 因为 SSO 只处理身份验证。所以, 应用程序可能仍然需要设置用户的属性(比如访问特权)。

对于单点登录, 经常提到的一些问题包括:

- *难以重构*。对 SSO 解决方案进行重构来适应现有的应用程序很困难, 很耗费时间而且昂贵。
- *无人看守的桌面*。实现 SSO 会减少一些安全风险, 但是也增加了其他安全风险。例如, 如果用户登录之后离开了他的机器, 恶意用户就可以访问他的资源。尽管这是一个普遍存在的安全问题, 但是 SSO 会使这个问题更加严重, 因为恶意用户可以访问所有获得授权的资源。在采用多次登录方式时, 用户每次只能登录进一个系统, 所以只有一个资源被泄露。
- *单点攻击*。在使用单点登录时, 所有应用程序使用一个集中的身份验证服务。对于希望实施拒绝服务攻击的黑客, 这是一个有吸引力的目标。

所以, SSO 并非毫无缺点。但是我相信, 在用户、管理员和开发人员看来, 它的优点要超过缺点。

↑ 回页首

SSO 开放源码项目

正如前面提到的，我当前正在为一个教育机构构建 Web 门户。这个门户将为参与远程课程的学生提供一个在线学习环境。

这个门户的构造块已经就位了。站点已经建立了，课程内容已经开发出来了，虚拟学习环境也就位了，辅助应用程序（比如日记、日历、电子邮件和笔记本）已经构建好或已经取得。所有这些组件都正在由学生使用，而且每个应用程序都运行在自己的服务器上。

客户现在希望能够通过 Web 浏览器远程访问这些应用程序，所以要构建一个门户来提供访问应用程序的单一入口，门户应该向用户提供单点登录功能。用户登录进门户之后，对他的身份进行验证，然后他就能够访问门户中所有已授权的资源。

我决定搜索一下关于如何实现 SSO 方案的信息，寻找有帮助的白皮书、产品和开放源码项目。我使用以下条件来限制搜索的范围：

- Java 实现。这个门户应用程序基础设施是基于 Java 的，所以我希望 SSO 实现也是基于 Java 语言的。
- 容易实现。在这个场景中，容易实现是指不需要对基础设施或现有的应用程序做大量修改。
- 其效果已经得到证明。它应该已经被一些大型组织采用，而且仍然处于活跃的开发阶段。
- 与 LDAP 兼容。我们的客户使用 Microsoft Active Directory Server，所以我需要系统能够轻松地根据 ADS 进行身份验证。

我很高兴地发现了几个出色的开放源码项目。（参见 参考资料 中的列表。）

我开始使用来自 Yale University 的 CAS（Central Authentication Service）系统开发一个原型，因为它满足我的所有条件。它是基于 Java 的，源代码是开放的。只需使用 JSP 标记和 servlet 过滤器，就能够相当轻松地在 Java 应用程序环境中实现它。Yale University 正在使用它，这说明它的品质满足我的条件。它还允许轻松地改变或扩展实际的身份验证机制（无论是查询数据库还是 LDAP 服务器上的用户名和密码）。

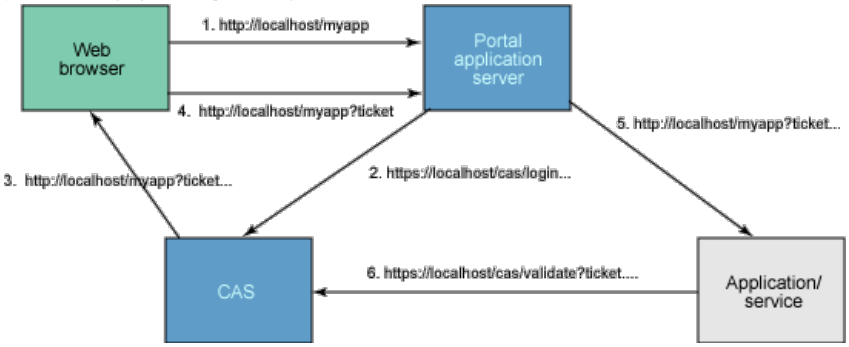
↑ 回页首

CAS 概述

注意，在采用 CAS 协议时，应用程序不会看到用户的密码。CAS 服务器执行身份验证，只有它能够看到用户的密码。这会增强安全性，因为用户名和密码并不通过网络传递给其他应用程序。

下图说明了在集成了 CAS 服务器的系统中身份验证是如何执行的。

图 1. CAS 协议如何执行身份验证



下面是这个身份验证协议中的主要步骤。

1. 用户尝试使用应用程序的 URL 访问应用程序。用户被重定向到 CAS 登录 URL，采用的是 HTTPS 连接，他请求的服务的名称作为参数传递。这时向用户显示一个用户名/密码对话框。
2. 用户输入 ID 和密码，CAS 对他进行身份验证。如果身份验证失败，目标应用程序根本不会知道这个用户曾经试图访问它 —— 用户在 CAS 服务器上就被拦住了。
3. 如果身份验证成功，CAS 就将用户重定向回目标应用程序，并在 URL 中附加一个称为 *ticket* 的参数。然后，CAS 尝试创建一个称为 *ticket-granting cookie* 的内存 cookie。这是为了以后进行自动的重新验证；如果存在这个 cookie，就表示这个用户已经成功地登录了，用户就不需要再次输入他的用户名和密码。
4. 然后，应用程序要检查这个 ticket 是否正确，以及是否代表一个有效用户；检查的方法是，打开一个 HTTPS 连接来调用 CAS `serviceValidate` URL，并作为参数传递 ticket 和服务名称。CAS 检查这个 ticket 是否有效，以及是否与请求的服务相关联。如果检查成功，CAS 就将用户名返回给应用程序。

如果采用 Servlet 2.3 规范进行开发，那么甚至不需要为这些步骤操心。一个 servlet 过滤器会处理整个协议。您要做的只是在 web.xml 文件中配置过滤器参数。我就采用这种方式 —— 它意味着对门户中应用程序代码的修改非常少。

对 CAS 的深入讨论超出了本文的范围，我建议您阅读 参考资料 中列出的来自 Yale University 的文章，从而判断这种身份验证方案是否满足您的需要。

↑ 回页首

CAS 入门

设置 CAS 软件是非常简单的，但是在开始设置之前，您应该了解我用的一些软件。我只用 Tomcat 4.1、Java Development Kit 1.4 和 Ant 1.5 测试

了 CAS。(可以从 参考资料 下载这里提到的文件和客户机库。)

首先，下载 CAS 服务器和客户机库。已经针对许多语言和环境开发了客户机库，包括 Java、ASP、Perl、PHP 和 PL/SQL。

CAS 使用 HTTPS，所以必须在 Tomcat 中启用这个功能。我发现这需要点儿技巧，但是如果按照我提供的说明(readme_tomcat_ssl.txt 文件)去做，应该不困难。

对 CAS 服务器 ZIP 文件进行解压，并使用 Ant 构建脚本构建 CAS 服务器软件。将 WAR 文件(Web Archives)部署到 Tomcat 的 */webapps* 目录中。在启动 Tomcat 时，用 WAR 文件在 *Tomcat/webapps* 中创建一个 CAS 目录。

下载 CAS 客户机库。对 ZIP 文件进行解压，就会看到许多目录。我要使用的是 Java 客户机库。同样，也提供了 Ant 构建脚本。运行这个构建脚本。这会生成一个称为 casclient.jar 的 JAR 文件。将这个文件复制到 Tomcat 根目录下的 *common/lib* 目录中。

现在，需要配置应用程序来使用 CAS。本文中用来进行演示的应用程序是 Tomcat 提供的可靠的 “HelloWorld” servlet 示例。这个应用程序应该在 Tomcat 系统中的 */webapps/examples* 目录下面。修改 web.xml 文件来配置 servlet 过滤器。

HelloWorld JSP 的 web.xml 文件包含下面的 servlet 过滤器配置。它对 HTTPS 使用本地主机和端口 8443。根据自己的配置修改这些设置。我提供的 zip 文件中包含一个 web.xml 文件示例。

清单 1. HelloWorld JSP 的默认 servlet 过滤器配置

```
<filter>
    <filter-name>CAS Filter</filter-name>
    <filter-class>edu.yale.its.tp.cas.client.filter.CASFilter</filter-class>
    <init-param>
      <param-name>edu.yale.its.tp.cas.client.filter.loginUrl</param-name>
      <param-value>https://localhost:8443/cas/login</param-value>
    </init-param>
    <init-param>
      <param-name>edu.yale.its.tp.cas.client.filter.validateUrl</param-name>
      <param-value>https://localhost:8443/cas/proxyValidate</param-value>
    </init-param>
    <init-param>
      <param-name>edu.yale.its.tp.cas.client.filter.serverName</param-name>
      <param-value>localhost</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>CAS Filter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

启动 Tomcat。然后输入 URL *http://localhost/examples/servlet/HelloWorldExample*。这时会重定向到 CAS 登录屏幕。默认的身份验证器只要求为用户名和密码提供相同的字符串，例如在两个域中都输入 *demo*。然后，就会重定向到 HelloWorld 页面。

这只是个简单的 CAS 演示，但是它说明，通过使用这个强大的 servlet 过滤器，让现有的 Java servlet 应用程序使用 CAS 是多么容易。还可以使用 JSP 标记集替代 servlet 过滤器，这种方法适合那些无法使用 servlet 过滤器的应用程序或应用服务器。

↑ 回页首

Active Directory Server 身份验证

在我的教育门户项目中，可以很轻松地修改应用程序，因为我能够得到源代码。我在每个应用程序中使用了 servlet 过滤器。在这种情况下，当用户首次访问门户并请求一个受限制的应用程序或数据源时，CAS 将执行身份验证。身份验证成功之后，CAS servlet 过滤器可以检查 ticket，并将用户名作为参数传递给请求的应用程序。我修改了现有的应用程序登录屏幕，让它们在成功的身份验证之后从 CAS 接收用户名参数。然后，应用程序可以使用这个参数对用户进行授权，并对用户的活动进行审计记录。

但是，CAS 本身没有附带任何真正有用的身份验证器。默认的身份验证器仅仅检查用户名和密码是否相同。在教育门户项目中，我使用 Microsoft 的 Active Directory Server 存储用户的个人信息。所以我需要扩展 CAS 服务器，让它根据 Active Directory Server 检查用户名和密码。

Yale University 的 CAS 设计人员开发了一个可插入的身份验证器架构。通过扩展 `PasswordHandler` 接口，开发人员可以创建一个类来实现密码检查逻辑。

清单 2. CAS 的可插入身份验证器架构

```
/** Interface for password-based authentication handlers. */
public interface PasswordHandler extends AuthHandler {

  /**
   * Authenticates the given username/password pair, returning true
   * on success and false on failure.
   */
  boolean authenticate(javax.servlet.ServletRequest request,
                       String username,
                       String password);

}
```

我要做的只是了解如何使用 Active Directory Server 检查用户名和密码，创建一个类来扩展这个接口，并添加执行身份验证的代码。

ADS 是一种与 LDAP3 兼容的目录服务器。它支持许多不同的身份验证方法，主要包括*匿名(anonymous)*、*简单(simple)* 和 *SASL*(Simple Authentication and Security Layer)。匿名身份验证不适合我的项目；简单身份验证用明文发送密码，这也不满足需要。所以我使用 SASL。

SASL 支持可插入的身份验证。这意味着可以对 LDAP 客户机和服务器进行配置，让它们相互协商并使用多种身份验证机制之一。下面是当前定义的一部分 SASL 机制：

- Anonymous
- CRAM-MD5
- Digest-MD5
- External
- Kerberos V4
- Kerberos V5
- SecurID
- Secure Remote Password
- S/Key
- X.509

在这些机制中，流行的 LDAP 服务器（比如 Sun、OpenLDAP 和 Microsoft 提供的服务器）都支持 External、Digest-MD5 和 Kerberos V5。

CAS 本身与 Kerberos 相似，它们有许多相同的概念，比如 ticket 和 ticket-granting ticket（在 CAS 中实际上称为 ticket-granting cookie），它们的协议也是相似的。所以，选择 Kerberos 机制似乎是很自然的。另外，添加对 Kerberos 身份验证的支持让 CAS 在未来的开发中能够作为基于 Web 的 Kerberos 代理代表用户，这使 CAS 能够替用户管理 Kerberos ticket 的所有方面。这意味着远程用户也可以使用相同的 Kerberos 身份验证机制访问本地和网络资源。

Kerberos 协议已经内置在 ADS 和 Windows 2000/XP 中。Java Authentication and Authorization Service（JAAS）提供了 Kerberos Login Module 的一个实现（参考资料 中列出的一个教程详细描述了如何建立一个示例应用程序）。精确地说，它是 GSS-API SASL 机制，但是它只为 LDAP3 服务器提供 Kerberos v5 身份验证。

Kerberos 身份验证是一个很简单的过程（如果仔细地按照以下说明进行操作的话）：

1. 在 JAAS 配置文件中为应用程序类配置 Login Module。

```
edu.yale.its.tp.cas.auth.provider.KerberosAuthHandler
{
com.sun.security.auth.module.Krb5LoginModule required client=TRUE;
};
```

2. 创建一个 `LoginContext`，传递执行身份验证的类的名称和 `CallBackHandler` 对象。

```
LoginContext lc = new LoginContext(CASApp.class.getName(),
                    new CASCallbackHandler());
```

3. 调用 `login()` 方法来执行身份验证。如果执行过程没有异常，身份验证就成功了。如果抛出了异常，那么这个异常会指出失败的原因。

要想深入了解 Kerberos 身份验证，建议您利用本文末尾的参考资料。（我还提供了自己的实现和配置文件，`KerberosAuthHandler` 和 `CASCallBackHandler`。）

需要创建一个新的 `PasswordHandler` 实现，`KerberosAuthHandler`，它按照上面的方法根据使用 Kerberos v5 的 Active Directory Server 进行身份验证。

清单 3. 创建新的 PasswordHandler 实现

```
public class KerberosAuthHandler implements PasswordHandler {

  public boolean authenticate(javax.servlet.ServletRequest request,
                              String username,
                              String password)
  {

    LoginContext lc = null;

    try
    {
 /* Set up the Callback handler, and initialise */
 /* the userid and password fields */

    CASCallbackHandler ch = new CASCallbackHandler();
    ch.setUserId(username);
       ch.setPassword(password);

    /* Initialise the login context - LoginModule configured */
    /* in cas_jaas.conf and */
             /* set to use Krb5LoginModule. */
    lc = new LoginContext(KerberosAuthHandler.class.getName(), ch);

    /* Perform the authentication */
    lc.login();

    }
    catch (LoginException le)
    {
     System.err.println("Authentication attempt failed" + le);
     return false;
    }
    return true;

  }
}
```

在创建 LoginContext 时，传递这个类名和 CASCallbackHandler 对象。JAAS 配置文件指定这个类使用的登录模块。

在调用 login() 方法时，登录模块知道为了进行身份验证它需要从用户/应用程序获得哪些信息。在使用 Kerberos 时，它需要用户名和密码，所以它构造一个数组，其中包含两个回调对象（NameCallback 和 PasswordCallback），然后调用 CallbackHandler 对象，这个对象决定如何执行这些回调并获得用户名和密码。

我采用最简单最有利的方式，使用 CallbackHandler 上的 setter 方法显式地设置用户 ID 和密码。然后，CallbackHandler 将它们传递给 NameCallback 和 PasswordCallback 对象。

清单 4. 用 setName (CASUserId) 和 setPassword (CASPassword) 设置 ID 和密码

```
public class CASCallbackHandler implements CallbackHandler
{

    private String CASUserId;
    private char [] CASPassword;

    public void handle(Callback[] callbacks)
 throws java.io.IOException, UnsupportedCallbackException {
     for (int i = 0; i < callbacks.length; i++) {
  if (callbacks[i] instanceof NameCallback) {
      NameCallback cb = (NameCallback)callbacks[i];
      cb.setName(CASUserId);

  } else if (callbacks[i] instanceof PasswordCallback) {
      PasswordCallback cb = (PasswordCallback)callbacks[i];
      cb.setPassword(CASPassword);

  } else {
      throw new UnsupportedCallbackException(callbacks[i]);
  }
     }
    }

    public void setUserId(String userid)
    {
 CASUserId = userid;
    }

    public void setPassword(String password)
    {
 CASPassword = new char[password.length()];
 password.getChars(0, CASPassword.length, CASPassword, 0);
    }
```

下面要做的是让 CAS 使用新的身份验证处理器。方法是在 *webapps/cas* 中 CAS 服务器的 web.xml 文件中进行以下设置：

清单 5. 让 CAS 使用新的身份验证处理器

```
<!-- Authentication handler -->
    <context-param>
        <param-name>edu.yale.its.tp.cas.authHandler</param-name>
        <param-value>
             edu.yale.its.tp.cas.auth.provider.KerberosAuthHandler
        </param-value>
    </context-param>
```

ZIP 文件（参考资料 中的 KerberosAuthSrc.zip）中包含一个 web.xml 示例。

必须重新启动 Tomcat，但是这一次还需要设置一些 Java 运行时属性。对 ZIP 文件（KerberosAuthSrc.zip）进行解压，并将文件 cas_jaas.conf、

krb5.conf 和 setkerberosjvmoptions.bat 复制到 *TOMCAT_HOME* 目录中。运行 setkerberosjvmoptions.bat，然后启动 Tomcat。

现在，可以再用 HelloWorld 应用程序做实验了。这一次，可以使用 Active Directory Server 中定义的有效的 Kerberos 用户名和密码对。

↑ 回页首

结束语

如果没有统一的策略，开发人员就要为每个网络应用程序重复实现定制的安全机制。这会导致各种可伸缩性和维护问题。单点登录解决方案为安全性和身份验证提供了统一的框架，这大大减轻了用户、管理员和开发人员的负担。

单点登录的概念、技术和蕴涵对用户和管理员而言很复杂，我在本文中只触及了这个领域的皮毛。但是，我解释了如何使用 Yale University 的 CAS 系统实现一个单点登录方案，还详细描述了如何扩展这种技术，从而对 LDAP 服务器用户（具体地说，是使用 Kerberos 协议的 Active Directory Server）进行身份验证。

参考资料

- 您可以参阅本文在 developerWorks 全球网站上的 英文原文。
- 下载本文中使用的 CAS 服务器和客户机库。
- 下载本文中使用的 Tomcat web.xml for CAS，包括设置 SSL 的说明。
- 下载本文中使用的 `KerberosAuthHandler` 和 `CASCallBackHandler` 代码。
- 阅读 “Using CAS at Yale: A Quick Introduction”。这篇文章详细描述 CAS 的组件，解释什么时候应该使用它，并演示使用它进行用户身份验证的最简单方法。
- 阅读 “CAS 2.0: Proxiable Credentials”，了解这次 CAS 升级的原因以及系统的新特性和改进。
- 阅读 “将 Web 服务用于电子交易的单点登录”（*developerWorks*，2002 年 1 月），了解如何将 electronic Customer Relationship Management（eCRM）应用程序集成进现有的电子商务市场应用程序中，从而向用户提供单点登录体验。
- 阅读文章 “Simplify enterprise Java authentication with single sign-on”（*developerWorks*，2003 年 9 月），了解如何使用 GSS-API 和 Kerberos 标准在 Java 平台上实现 SSO 来处理安全问题。
- 通过 “WebSphere and Domino single sign-on”（*developerWorks*，2001 年 1 月），了解 Lightweight Third Party Authentication（LTPA）的细节，这是服务器把单点登录身份验证任务委托给一个公用第三方服务的一种方法。
- 阅读 “Java 安全性，第二部分：认证与授权”（*developerWorks*，2002 年 7 月），这个教程介绍了身份验证和授权的基本概念，概述了 JAAS 的架构。
- 访问 ITS Central Authentication Service 站点，这里解释了 CAS 1.0 的设计和实现思想。
- 访问 Tips for LDAP Users 站点，了解如何使用不同的安全身份验证机制和 SSL（Secure Socket Layer）访问 LDAP 服务。
- 访问 Security Guide, Kerberos chapter，了解安全远程命令、使用 Kerberos 向 AIX 进行身份验证以及关于 Kerberos 故障检修的信息。

条评论

请 登录 或 注册 后发表评论。

添加评论：

注意：评论中不支持 HTML 语法

有新评论时提醒我　　剩余 1000 字符

发布

快来添加第一条评论

↑ 回页首