超级狗

身份认证使用指南

gemalto
security to be free

目录

前言

本文档描述了如何使用超级狗用于身份认证及其示例等相关信息。

产品简介

超级狗双因素身份认证可以使网络访问者经过双重认证才能获得网络访问授权。授权的访问者必须持有超级狗并且知道其口令。访问者需要把超级狗插入 USB 接口中，并且输入正确的口令，以证实自己的身份。如果身份认证服务器不能识别访问者的身份，网络访问将被拒绝。

超级狗双因素身份认证向用户提供了比传统用户名口令方式更安全的网络身份认证机制。

工作原理

超级狗利用内置算法采用查询响应的方式对访问者身份进行验证和确认。

当身份认证服务器发给超级狗查询值时，超级狗使用内置算法计算出匹配的响应值，并通过网络发送给服务器。查询值是随机的，因此拦截超级狗与服务器的通讯数据并不能帮助破解。此方式与输入口令验证方式相比更安全，更难于破解，因为口令并没有在网络上传输。

超级狗采用单向不可逆哈希算法计算响应值。超级狗内置了智能卡芯片，能够有效地防止硬件复制或伪造。超强的硬件保护，使算法不可被非法读取，从而保障了安全性。

不同开发商购买的超级狗内置的算法因子不同，可以有效防止其他人购买超级狗伪装成合法用户。

测试环境

服务器端支持 JSP、ASP.NET 和 PHP，客户端支持主流浏览器。目前已测试的平台如下：

- Windows XP: IE8, Firefox, Google Chrome
- Windows 7: IE8, IE9, IE10, Firefox, Google Chrome
- Windows 8: IE10, Firefox, Google Chrome
- Windows 8.1: IE11, Firefox, Google Chrome
- Windows 10: IE11, Firefox, Google Chrome

第一章 软件包内容

此软件包包含如下文件：

文件名	版本	描述
Tools\AuthCodeGenerator.exe	2.3.1.52711	认证代码生成工具，用来为您的认证系统生成认证代码
Tools\AuthTool.exe	2.3.0.1	认证工具，管理员可以使用此工具： 1. 修改管理员口令 2. 设置认证因子 3. 解锁并设置新的用户口令 4. 设置用户名 5. 修改用户数据
Tools\Source\AuthTool	2.3.0.1	认证工具的源文件
WebServer\dog_auth_srv.dll *WebServer\dog_auth_srv_x64.dll*	2.3.1.52712	服务器端需要用到的认证用户的动态链接库
WebServer\dog_auth_srv_php.dll	2.3.0.1	PHP 工程的服务器端需要用到的认证用户的 PHP 扩展组件
WebServer\JSP\Authentication	2.3.0.2	超级狗 Java Web 认证示例程序
WebServer\ASP.NET\Authentication	2.3.0.4	超级狗 ASP.NET 认证示例程序
WebServer\PHP\Authentication	2.3.0.2	超级狗 PHP 认证示例程序
WebServer\PHP\Source\dog_auth_srv_php	2.3.0.1	超级狗服务器端 PHP 扩展 DLL 的源文件
BrowserPlugin\DogAuthPlugin.msi	2.4.1	超级狗认证控件安装包。用于安装客户端需要用到的支持 IE 和 Firefox 浏览器的控件
BrowserPlugin\DogAuthExtension.crx	1.0	支持超级狗认证的 Chrome 浏览器扩展程序
SuperDog Authentication Sample Readme.*pdf*	无	本文档

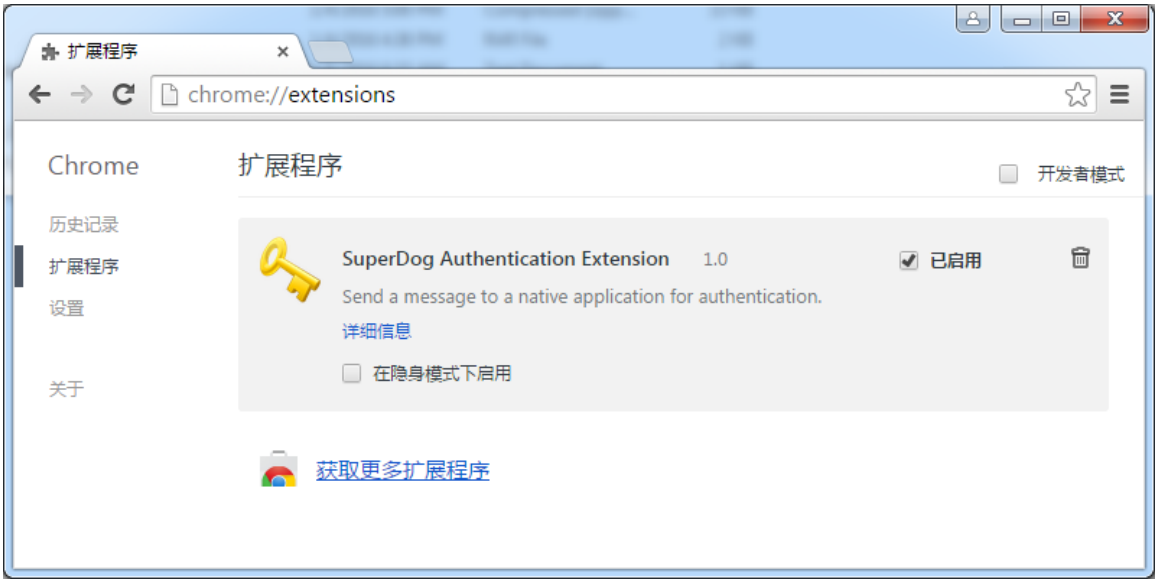
第二章 使用示例工程

安装客户端浏览器控件及组件

客户端需安装 32-bit IE 8.0 或者以上版本，Chrome 或者 Firefox 浏览器。

如果客户端使用 IE 浏览器，客户端既可以通过网页提示下载并安装客户端控件，也可以运行 DogAuthPlugin.msi 安装控件；如果客户端使用 Firefox 浏览器，请运行 DogAuthPlugin.msi 安装控件；如果客户端使用 Chrome 浏览器，除了运行 DogAuthPlugin.msi 安装用于超级狗认证的 Native Messaging Host 应用程序，还需要按照如下步骤安装 Chrome 扩展程序：

1. 在 Chrome 浏览器的地址栏中，输入 chrome://extensions，打开扩展管理页面
2. 将文件 DogAuthExtension.crx 拖进去即可



Java Web 工程（Tomcat）

使用需求

Java Web 服务器端需安装 Apache Tomcat 6.0 和 JRE 1.6 或 JRE1.7。

由于 JRE1.8 与 JDK1.8 不再包含 JDBC-ODBC Bridge，示例工程的 access 数据库将无法正常使用，所以示例工程不能使用 JRE1.8 与 JDK1.8。

编译/使用 Java Web 示例程序

Java Web 示例工程使用超级狗试用件演示。请按照如下步骤使用此软件包：

1. 在服务器端，根据您安装的 JRE 的版本是否为 64 位，选择将文件 dog_auth_srv.dll 或 dog_auth_srv_x64.dll 放到 Tomcat 安装目录的 bin 目录下。如：C:\Program Files\Apache Software Foundation\Tomcat 6.0\bin。
2. 在客户端打开 32-bit 浏览器，输入网址

   http://[服务器 IP]:8080/Authentication/Login.jsp 即可。

最终用户第一次使用请点击进入注册页面，输入用户名和用户口令进行注册。

用户名和用户口令将被写入超级狗硬件；同时，用户名和超级狗 ID 将被写入服务器端数据库中。

用户可以对用户口令进行更改。

部署您自己的 Java Web 工程

实际的工程将绑定您的开发商 ID 和认证代码。请参照如下步骤：

1. 请插入您的开发狗，使用认证代码生成工具生成认证代码，文件名默认为：auth_code.xml。文件内容包含您的开发商 ID 和认证代码。请将 auth_code.xml 拷贝至服务器端工程的目录 WEB-INF 中。
2. 使用超级狗认证初始化工具（AuthTool.exe）修改超级狗的管理员口令（SO PIN，默认为：“abcdefgh”）和认证因子（默认为：“00000000”），设置用户口令（USER PIN，默认为：“12345678”）与用户信息。也可通过 web 工程的用户注册页面让用户自行注册，设置用户名和用户口令。如果您修改了认证因子，请同时修改服务器端 WEB-INF 目录中的配置文件：auth_factor.xml。为了提高安全性，建议您修改默认的认证因子。
3. 在服务器端，将您的 web 工程按照示例工程修改后放到 Tomcat 安装目录的 webapps 文件夹下。
4. 在客户端打开 32-bit 浏览器，输入您的 web 工程的登录网址即可。

注意

当您使用 64 位的 JRE 部署工程，并使用 Microsoft Access 作为数据库时，您可能会遇到如下异常信息：“java.sql.SQLException: [Microsoft][ODBC Driver Manager] Data source name not found and no default driver specified”。这是由于您没有安装 Microsoft Access Database Engine（64-bit）的原因。您可以采取如下措施：

a. 采用其他数据库。本工程采取 Microsoft Access 作为演示数据库。在实际应用中，建议采用更常用的商业数据库。

b. 采用 32 位的 JRE 部署本工程。

c. 如果您继续使用 64 位的 JRE 与 Microsoft Access，您需要先确认卸载 32 位的 ODBC（可能需要卸载 Office），然后下载 64 位的 Microsoft Access Database Engine 2010 Redistributable（http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=13255）并安装。

ASP.NET 工程

使用需求

ASP.NET 服务器端需安装 Microsoft .NET Framework 4.0 和 IIS 7。

编译/使用 ASP.NET 示例程序

ASP.NET 示例工程使用超级狗试用件演示。请按照如下步骤使用此软件包：

1. 在服务器端，根据您的系统是否为 64 位，选择将文件 dog_auth_srv.dll 放到：（系统盘符:）\Windows\SysWOW64 或（系统盘符:）\Windows\System32 目录下。例如：C:\Windows\SysWOW64。（目前 64 位系统暂不支持调用 64 位动态库：dog_auth_srv_x64.dll）。
2. 使用 Visual Studio 2010 编译本工程并运行。

最终用户第一次使用请点击进入注册页面，输入用户名和用户口令进行注册。

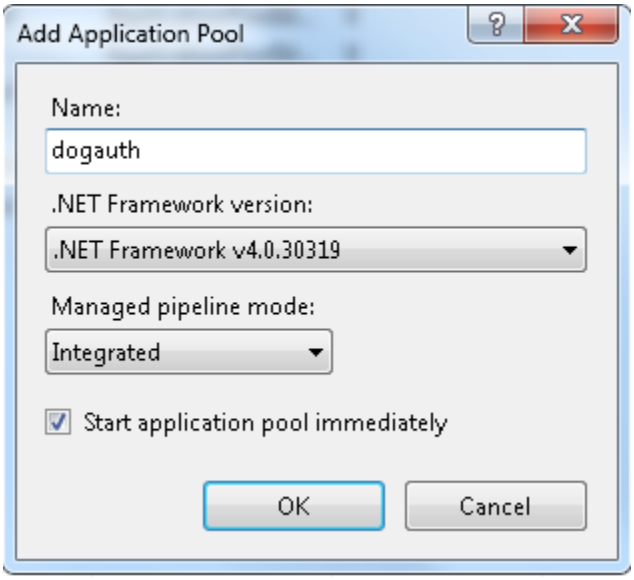
用户名和用户口令将被写入超级狗硬件；同时，用户名和超级狗 ID 将被写入服务器端数据库中。

用户可以对用户口令进行更改。

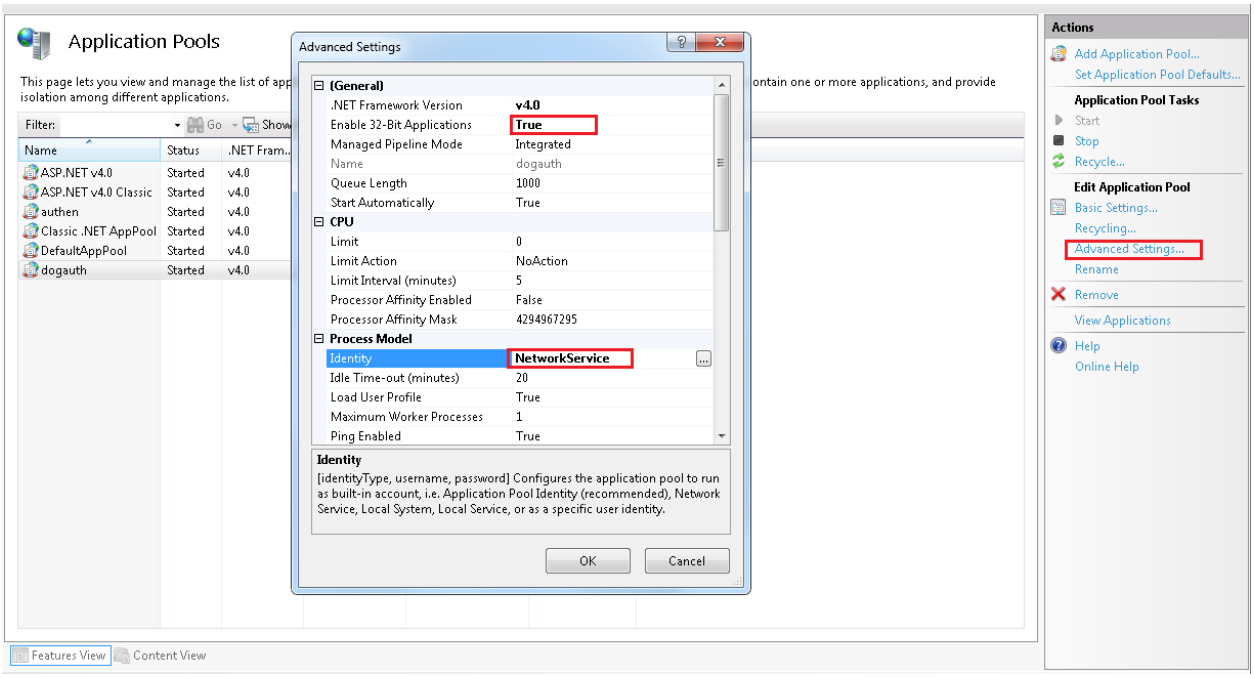
部署您自己的 ASP.NET 工程

实际的工程将绑定您的开发商 ID 和认证代码。此示例以 Windows 7 系统与 IIS 7 为例，步骤如下：

1. 请插入您的开发狗，使用认证代码生成工具生成认证代码，文件名默认为：auth_code.xml。文件内容包含您的开发商 ID 和认证代码。请将 auth_code.xml 拷贝至服务器端工程的目录 serverdata 中。
2. 使用超级狗认证初始化工具（AuthTool.exe）修改超级狗的管理员口令（SO PIN，默认为：“abcdefgh”）和认证因子（默认为：“00000000”），设置用户口令（USER PIN，默认为：“12345678”）与用户信息。也可通过 web 工程的用户注册页面让用户自行注册修改用户口令与用户名。如果您修改了认证因子，请同时修改服务器端 serverdata 目录中的配置文件：auth_factor.xml。为了提高安全性，建议您修改默认的认证因子。
3. 在服务器端，将您的 web 工程按照示例工程修改后放到一个目录中。例如：D:\Authentication。
4. 在服务器端安装 IIS 后，在控制面板中选择：Administrative Tools->Internet Information Services (IIS) Manager，双击打开。
5. 在左侧的 Application Pools 节点右键点击选择：“Add Application Pool…”，然后做出如图设置：

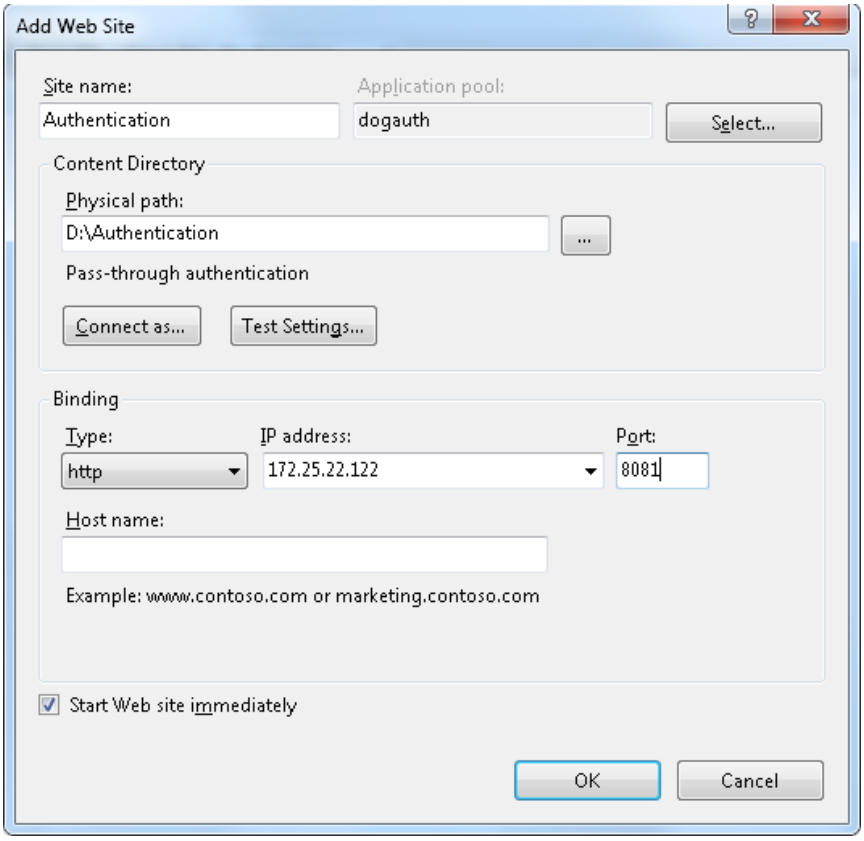


点击“OK”确认。然后点击右侧的“Advanced Settings…”，打开“Advanced Setting”对话框，做出如下图设置：



点击“OK”确认。

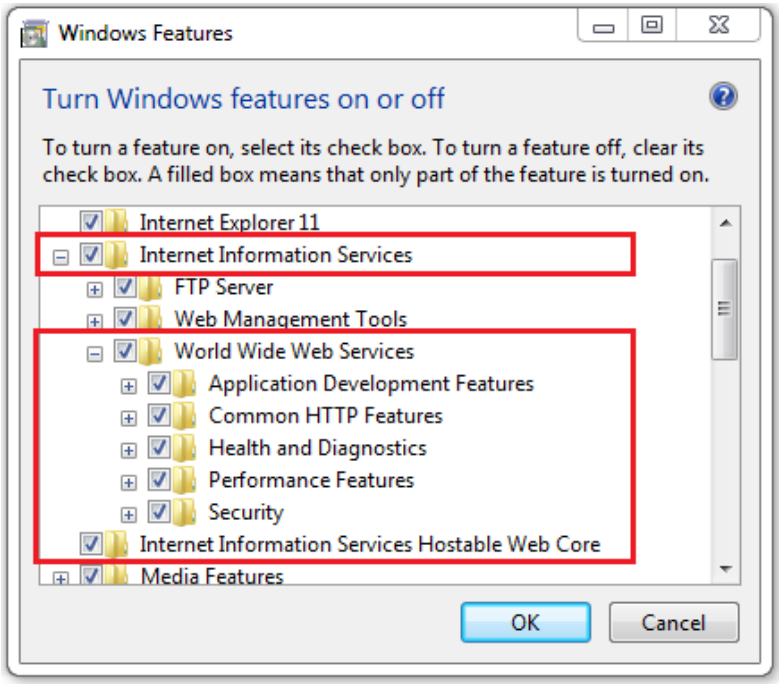
6. 在左侧的 Sites 节点右键点击选择：“Add Web Site…”，在弹出的对话框中，做出相关设置，例如：在“Application pool：”中选择上面设置的“dogauth”；在“Physical path”中设置为工程的存放路径；在“IP Address”中设置一个可用的 IP 地址，选择一个不会跟其他应用程序冲突的端口。如图：



点击“OK”确认。

注意

1．在服务器端安装 IIS 后，请在控制面板中选择：Programs->Programs and Features->Turn Windows features on or off，进行如下配置：



2．配置结束后，以管理员权限运行“命令提示符”，运行以下命令：

```
%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_regiis.exe -i
```

PHP Web 工程

使用需求

PHP Web 服务器端需下载并配置 Apache 和 PHP。

本例中通过测试版本：

Apache 2.4.12 vc9 x86 和 php-5.4.36-Win32-VC9-x86

使用 PHP 示例程序

PHP 示例工程使用超级狗试用件演示。请按照如下步骤使用此软件包：

1．下载 Apache 与 PHP 压缩包并部署。

2．在服务器端，将超级狗认证服务器端 PHP 扩展组件 dog_auth_srv_php.dll 放到：（盘符:）\（PHP 安装文件夹）\ext 目录下。例如：D:\php\ext。在 PHP 的配置文件（php.ini）中添加“extension=dog_auth_srv_php.dll”。该组件对应 php 源码版本为：5.4.36 版本，请使用相应版本的 php 二进制包进行部署。

如果用户使用其他 php 版本部署工程，请使用超级狗认证服务器端 PHP 扩展组件源代码工程（目录：WebServer\PHP\Source\dog_auth_srv_php）链接相应的 php 源代码版本，编译出对应的 PHP 扩展组件。该组件提供了对客户端进行身份认证的相关功能。

3. 本工程无需编译，使用 Apache 部署本工程后，在客户端打开 32-bit 浏览器，输入网址：

   http://[服务器 IP]:（Apache 中部署的端口号）/Login.php 即可。

最终用户第一次使用请点击进入注册页面，输入用户名和用户口令进行注册。

用户名和用户口令将被写入超级狗硬件；同时，用户名和超级狗 ID 将被写入服务器端数据库中。

用户可以对用户口令进行更改。

部署您自己的 PHP 工程

实际的工程将绑定您的开发商 ID 和认证代码。工程部署示范采用 Windows 7 系统与 Apache 2.4.12 x86 版本，请参照如下步骤：

1. 请插入您的开发狗，使用认证代码生成工具生成认证代码，文件名默认为：auth_code.xml。文件内容包含您的开发商 ID 和认证代码。请将 auth_code.xml 拷贝至服务器端工程的目录 serverdata 中。
2. 使用超级狗认证初始化工具（AuthTool.exe）修改超级狗的管理员口令（SO PIN，默认为：“abcdefgh”）和认证因子（默认为：“00000000”），设置用户口令（USER PIN，默认为：“12345678”）与用户信息。也可通过 web 工程的用户注册页面让用户自行注册修改用户口令与用户名。如果您修改了认证因子，请同时修改 serverdata 目录中的配置文件：auth_factor.xml。为了提高安全性，建议您修改默认的认证因子。
3. 在服务器端，按照常规方法部署在 Apache 与 PHP 运行环境。在 Apache 配置文件（Apache 文件夹）\conf\httpd.conf 中，设置您的 web 工程路径，例如：

   ```
   DocumentRoot "E:/Authentication/php"
   <Directory "E:/Authentication/php">
   ```

注意

1. 在 Windows 系统部署 PHP 环境需要安装 Visual C++ Redistributable Package。请按照您选择的 PHP 版本中的说明文档安装对应版本的 Visual C++ Redistributable Package。
2. 示例工程中提供的 PHP 扩展组件 dog_auth_srv_php.dll 对应 php 源码版本为：5.4.36 版本，如果使用其他 php 版本部署工程，请使用超级狗认证服务器端 PHP 扩展组件源代码工程（目录：WebServer\PHP\Source\dog_auth_srv_php）链接相应的 php 源代码版本，编译出对应的 PHP 扩展组件。否则可能将导致 dog_auth_srv_php.dll 无法正常加载。
3. PHP 部署时需要指定一个会话缓存目录，并将该路径设置到 PHP 的配置文件中。例如：在 php.ini 文件中设置：

   ```
   session.save_path = "D:/tmp"
   ```

   没有正确设置将影响 PHP 工程的正常使用。

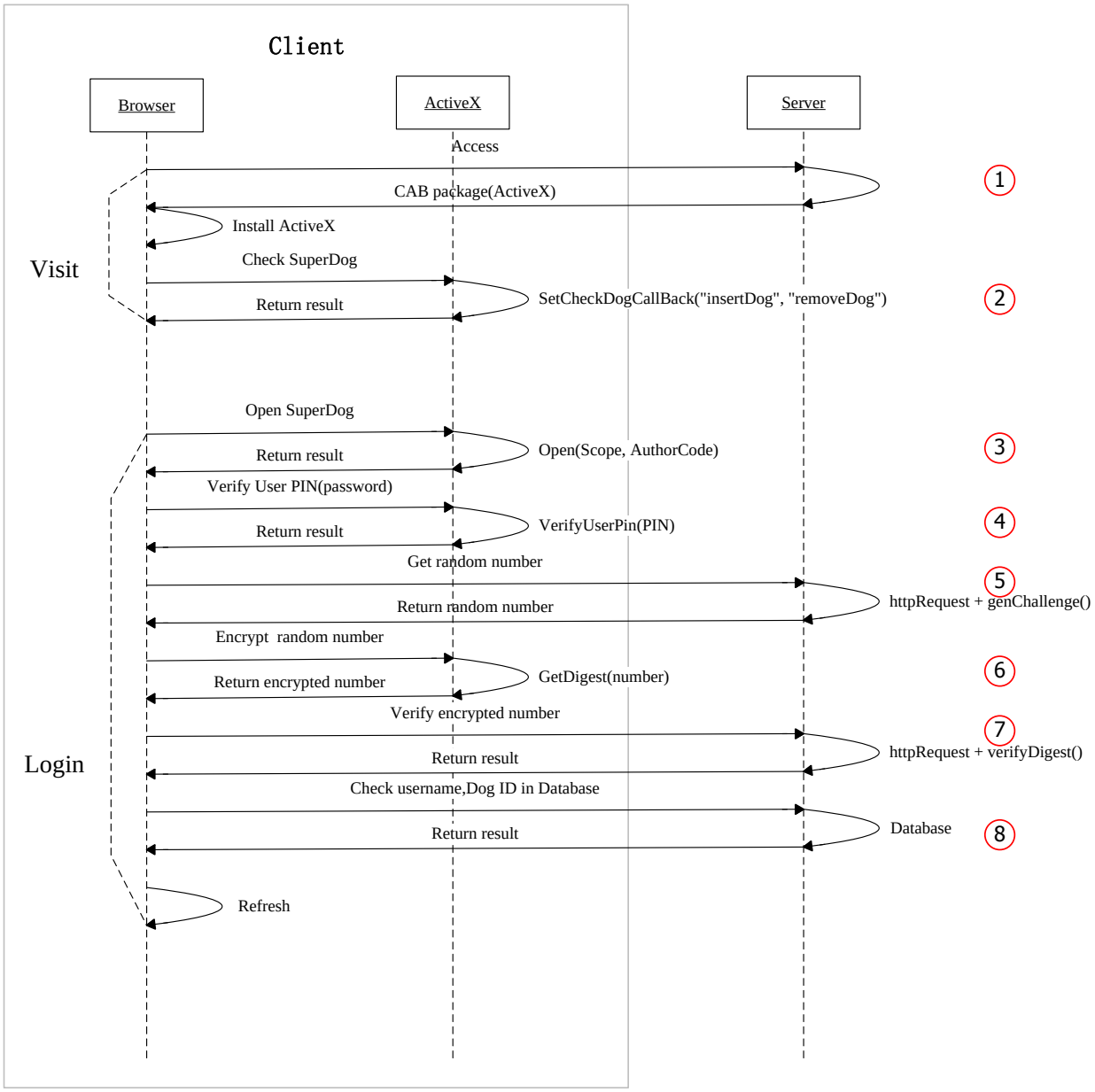
实际应用注意事项

1. 管理员可以使用认证工具 AuthTool.exe 修改默认的管理员口令（SO PIN），默认的管理员口令是：“abcdefgh”。为了提高安全性，强烈建议您修改默认的管理员口令。
2. 管理员可以使用超级狗认证初始化工具 AuthTool.exe 修改默认的认证因子（Authenticate Factor），默认的认证因子是“00000000”。如果您修改了认证因子，请同时修改服务器端配置文件：auth_factor.xml。为了提高安全性，建议您修改默认的认证因子。
3. 连续输入 15 次错误的管理员口令，管理员口令会被锁定。管理员口令被锁定后不影响用户使用超级狗进行认证操作。
4. 用户连续输入 15 次错误的用户口令，用户口令会被锁定。如果要解除锁定，用户需要将超级狗寄给管理员，管理员可以使用认证工具 AuthTool.exe 解锁并设置新的用户口令。
5. 如果用户使用 DogAuthPlugin.msi 安装控件，控件将被安装到如下目录：
    a. 如果是 64 位系统，C:\Program Files (x86)\Common Files\Gemalto Dog\SuperDog Auth
    b. 如果是 32 位系统，C:\Program Files\Common Files\Gemalto Dog\SuperDog Auth
6. 用户名、口令、用户数据等信息将被加密存储于超级狗默认的数据文件中（文件 ID 为 65524）。如果您还将超级狗用于软件保护，请不要调用 API 更改此文件。您可以使用授权管理工具设计并写入其它的数据文件。
7. 在测试的过程中，如果您想删除超级狗中的用户名、口令、用户数据等信息，请使用授权管理工具。步骤如下：
    a. 关闭打开着的浏览器，确保超级狗没有被使用
    b. 打开授权管理工具，点击右下角的“编程超级狗”按钮
    c. 插拔一次超级狗

第三章 认证流程

用户登录流程

以使用 IE 浏览器为例，使用其它浏览器时略有区别。



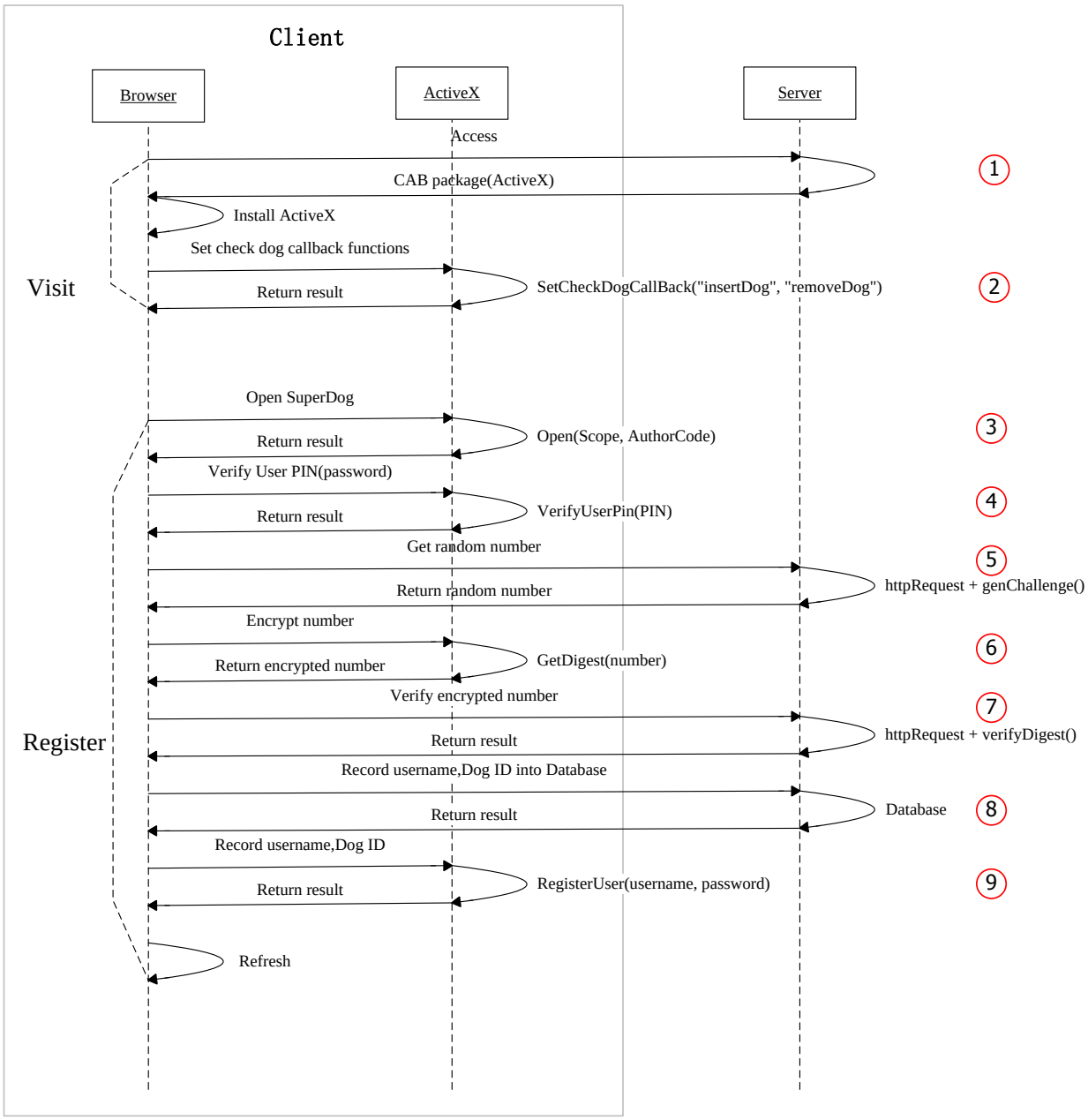
图（一）

流程说明

1. 浏览登录页面，如 http://localhost:8080/Authentication/Login.jsp 。客户端会自动提示加载安装 ActiveX 控件，点击安装，完成控件的安装，如果已经安装了控件则不会提示。
2. 页面的 javascript 代码调用控件的 SetCheckDogCallBack ("insertDog","removeDog") 方法设置 js 回调函数。"insertDog"插入超级狗的响应函数，"removeDog"拔出超级狗的响应函数。
3. 页面的 javascript 代码调用控件的 Open (Scope, AuthorCode) 方法，打开超级狗。只有打开狗以后才可访问后续的处理函数，使用结束后需调用 Close () 方法关闭超级狗。参数 Scope 表示在多个超级狗同时存在的情况下，可以打开特定的狗；参数 AuthorCode 是从服务端的 auth_code.xml 配置文件中读取的算法数据。
4. 页面的 javascript 代码调用控件的 VerifyUserPin(PIN) 方法，验证用户输入的口令与超级狗中的口令是否一致。
5. 页面的 javascript 代码调用 getChallenge() 方法发送 httpRequest 请求，获取服务端随机生成的挑战数据，服务端同时把数据记录到 session 中。
6. 页面的 javascript 代码调用控件的 GetDigest (PIN) 方法，对挑战数据进行加密处理。
7. 页面的 javascript 代码调用 doAuth () 方法发送 httpRequest 请求，把从 ActiveX 控件中获取的 DogID 和加密的挑战数据发送给服务端。服务端的 verifyDigest()方法对数据进行比对。
8. 对登录的用户进行数据库（Access Database）的查询匹配，如果匹配成功则可以访问主页。

用户注册流程

以使用 IE 浏览器为例，使用其它浏览器时略有区别。



图（二）

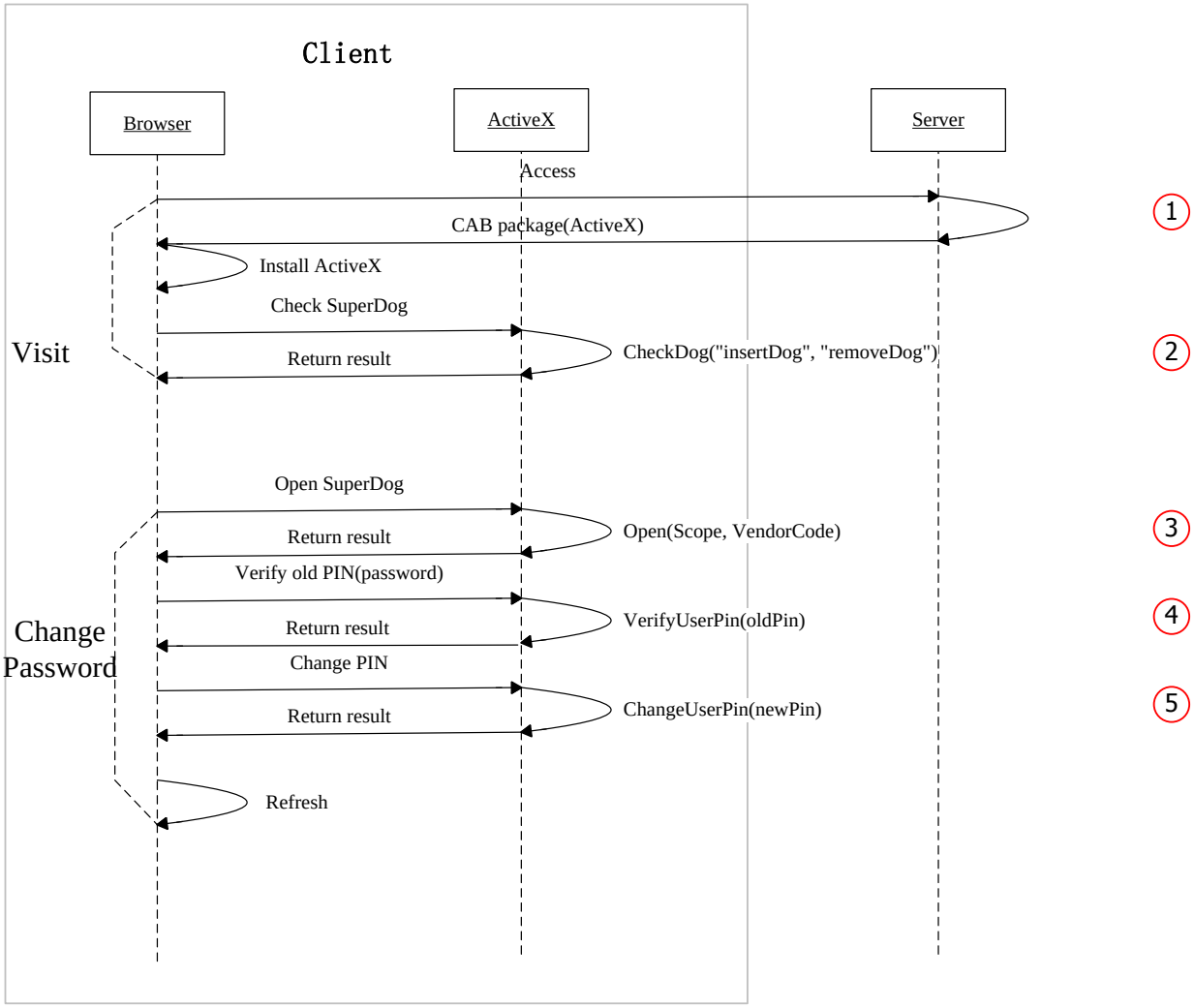
流程说明

1. 浏览登录页面，如：http://localhost:8080/Authentication/Register.jsp 。客户端会自动加载提示安装 ActiveX 控件，点击安装，完成控件的安装，如果已经安装了控件则不会提示。

2. 页面的 javascript 代码调用控件的 SetCheckDogCallBack ("insertDog","removeDog") 方法设置 js 回调函数。"insertDog"，插入超级狗的响应函数，"removeDog"拔出超级狗的响应函数。
3. 页面的 javascript 代码调用控件的 Open (Scope, AuthorCode) 方法，打开超级狗。只有打开狗以后才可访问后续的处理函数，使用结束后需调用 Close () 方法关闭超级狗。参数 Scope 表示在多个超级狗同时存在的情况下，可以打开特定的狗；参数 AuthorCode 是从服务端的 auth_code.xml 配置文件中读取的算法数据。
4. 页面的 javascript 代码调用控件的 VerifyUserPin(PIN) 方法，对于新注册的超级狗需要先验证，输入初始参数"12345678"。
5. 页面的 javascript 代码调用 getChallenge() 方法发送 httpRequest 请求，获取服务端随机生成的挑战数据，服务端同时把数据记录到 session 中。
6. 页面的 javascript 代码调用控件的 GetDigest (PIN) 方法，对挑战数据进行加密处理。
7. 页面的 javascript 代码调用 doAuth () 方法发送 httpRequest 请求，把从 ActiveX 控件中获取的 DogID 和加密的挑战数据发送给服务端。服务端的 verifyDigest()方法对数据进行比对。
8. 对新注册的用户记录用户信息，把信息写入数据库（Access Database）。
9. 数据库写入成功后，调用 ActiveX 控件的 RegisterUser(username, PIN)方法，把注册信息写入到超级狗里。

用户修改口令流程

以使用 IE 浏览器为例，使用其它浏览器时略有区别。



图（三）

流程说明

1．浏览登录页面，如：

http://localhost:8080/Authentication/ModifyPin.jsp 。客户端会自动加载提示安装 ActiveX 控件，点击安装，完成控件的安装，如果已经安装了控件则不会提示。

2．页面的 javascript 代码调用控件的 SetCheckDogCallBack（“insertDog”,”removeDog”） 方法设置 js 回调函数。“insertDog”，插入超级狗的响应函数，“removeDog”拔出超级狗的响应函数。

3. 页面的 javascript 代码调用控件的 Open (Scope, AuthorCode) 方法，打开超级狗。只有打开狗以后才可访问后续的处理函数，使用结束后需调用 Close () 方法关闭超级狗。参数 Scope 表示在多个超级狗同时存在的情况下，可以打开特定的狗；参数 AuthorCode 是从服务端的 auth_code.xml 配置文件中读取的算法数据。
4. 页面的 javascript 代码调用控件的 VerifyUserPin(PIN) 方法，验证用户输入的旧口令与超级狗中的口令是否一致。
5. 页面的 javascript 代码调用控件的 ChangeUserPin (newPIN) 方法，修改新口令为 newPIN。

第四章 客户端浏览器控件及组件

IE 浏览器控件

超级狗认证 IE 浏览器控件是一个经过签名的 ActiveX 形式的 COM 控件，它的文件名是 SuperDogAuth.ocx。浏览器控件与初始化工具相对应，向 web 前端开发者提供了一组可以利用超级狗内部的默认数据文件进行身份认证的函数接口。

超级狗认证 IE 浏览器控件的 CLSID 为“05C384B0-F45D-46DB-9055-C72DC76176E3”，并能够响应超级狗的硬件插拔消息。在网页中，可以这样对它定义：

```
<object id="AuthIE" name="AuthIE" width="0px" height="0px"
          codebase="DogAuth.CAB"
          classid="CLSID:05C384B0-F45D-46DB-9055-C72DC76176E3">
</object>
```

下面对超级狗认证 IE 浏览器控件的方法进行说明：

- **LONG Open([in]BSTR strScope, [in]BSTR strAuthCode)**

说明：打开目前系统中的一个超级狗。如果有多个超级狗连接到当前系统中，请在第一个输入参数中，指定要搜索超级狗 ID 的 XML 格式定义。更多信息，参阅超级狗帮助文档中 《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中 《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.Open(scope, auth_code);
```

注：这里的 AuthIE 是网页内定义的超级狗认证 IE 浏览器控件的名称（ID），开发中应根据实际命名进行改变。下同。

- **LONG Close()**

说明：从当前的一个会话环境中注销。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.Close();
```

- **LONG VerifyUserPin([in]BSTR strPin)**

说明：校验超级狗的用户口令。

输入数据：

strPin：用户口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.VerifyUserPin("12345678");
```

- **LONG ChangeUserPin([in]BSTR strNewPin)**

说明：修改超级狗的用户口令。需要先校验超级狗的用户口令成功。

输入数据：

strNewPin：新的用户口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.ChangeUserPin("111111");
```

- **LONG GetUserName()**

说明：获取超级狗的用户名字符串。需要在调用该方法之后再通过控件属性 UserNameStr 获取用户名。最大长度：32 字节

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.GetUserName();
var name = AuthIE.UserNameStr;
```

- **LONG GetDogID()**

说明：获取当前会话环境中的超级狗 ID。需要在调用该方法之后再通过控件属性 DogIdStr 获取超级狗 ID。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.GetDogID();
var dogid = AuthIE.DogIdStr;
```

- **LONG GetDigest([in]BSTR strChallenge)**

说明：根据输入的字符串生成加密的摘要字符串。需要在调用该方法之后再通过控件属性 DigestStr 获取加密的摘要字符串。需要先校验超级狗的口令成功。

输入数据：

strChallenge：用于生成加密摘要的字符串。即从服务器端获取的挑战数据。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.GetDigest("3702032AA96A457CB619C523E1126340");

var digest = AuthIE.DigestStr;
```

- **LONG SetCheckDogCallBack([in]BSTR strInstFunc, [in]BSTR strRmvFunc)**

说明：开启服务检测超级狗的插拔。通过参数传递响应超级狗插拔设备的 javascript 函数名称。

输入数据：

strInstFunc：用于响应超级狗连接到系统的 javascript 函数名称。

strRmvFunc：用于响应拔掉超级狗的 javascript 函数名称。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.SetCheckDogCallBack("insertfunc", "pullfunc");
```

- **LONG RegisterUser([in]BSTR strUser, [in]BSTR strPin)**

说明：注册用户信息。第一次使用超级狗时，将用户名和用户口令写入超级狗默认的数据文件中。

输入数据：

strUser：用户名字符串。最大长度：32 字节。

strPin：用户口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.RegisterUser("Edward", "12345678");
```

- **LONG SetUserData([in]LONG iDataType, [in]LONG iOffset, [in]BSTR strData)**

说明：设置自定义用户数据。需要管理员口令或用户口令验证通过。

输入数据：

iDataType：字符串类型为：16 进制格式或普通可显示字符串。

iOffset：存储到超级狗中的偏移量。

strData：自定义用户数据。

16 进制格式字符串，例如："EFD9AE38"，长度必须为偶数，函数内部会把该字符串转换为对应的 ASCII 码存储到超级狗中。字符串长度/2 + iOffset 必须小于或等于 50；

普通可显示字符串，例如："SuperDog"，字符串长度 + iOffset 必须小于或等于 50。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.SetUserData(1, 30, "SuperDog Auth");
```

- **LONG GetUserData([in]LONG iDataType, [in]LONG iOffset, [in] LONG iDataSize)**

说明：获取自定义用户数据。需要在调用该方法之后再通过控件属性 UserDataStr 获取用户信息。

输入数据：

iDataType：字符串类型为：16 进制格式或普通可显示字符串。

iOffset：从超级狗中的偏移量读取。

iDataSize：获取用户数据的大小。需要比欲取到的数据长度大一个字节。

16 进制格式字符串，例如："EFD9AE38"，(iDataSize - 1)/2 + iOffset 必须小于或等于 50；

普通可显示字符串，例如："SuperDog"，(iDataSize - 1) + ioffset 必须小于或等于 50。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
AuthIE.GetUserData(1, 30, 11);
var data = AuthIE.UserDataStr;
```

- **LONG EnumDog([in]BSTR strAuthCode)**

说明：获取当前连接到系统的超级狗的 ID。需要在调用该方法之后再通过控件属性 EnumResultStr 获取超级狗的 ID。

输入数据：

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

返回数据：

DOG_STATUS_OK： 请求已成功完成

返回的超级狗 ID 信息格式如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<dog_info>
    <dog id="591751055" />
    <dog id="591751069" />
</dog_info>
```

JavaScript 调用示例：

```
AuthIE.EnumDog(auth_code);
var dog_info = AuthIE.EnumResultStr;
```

Firefox 浏览器控件

超级狗认证 Firefox 浏览器控件是一个经过签名的基于 NPAPI 的控件，它的文件名是 npDogAuth.dll。浏览器控件与初始化工具相对应，向 web 前端开发者提供了一组可以利用超级狗内部的默认文件进行身份认证的函数接口。

超级狗认证 Firefox 浏览器控件能够响应超级狗的硬件插拔。在网页中，可以这样对它定义：

```
var temp = document.body.innerHTML;
var embed_tag = "<embed id=\"Authplg\" type=\"application/x-dogauth\" width=0 height=0 hidden=\"true\"></embed>";
document.body.innerHTML = embed_tag + temp;
```

下面对超级狗认证 Firefox 浏览器控件的所有的接口函数进行说明：

- **int Open([in]NPSTRING strScope, [in]NPSTRING strAuthCode)**

说明：打开目前系统中的一个超级狗。如果有多个超级狗连接到当前系统中，请在第一个输入参数中，指定要搜索超级狗 ID 的 XML 格式定义。更多信息，参阅超级狗帮助文档中 《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中 《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.Open(scope, auth_code);
```

注：这里的 Authplg 是网页内定义的超级狗认证 Firefox 浏览器控件的名称（ID），开发中应根据实际命名进行改变。下同。

- **int Close()**

说明：从当前的一个会话环境中注销。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.Close();
```

- **int VerifyUserPin([in]NPSTRING strPin)**

说明：校验超级狗的用户口令。

输入数据：

strPin：用户口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.VerifyUserPin("12345678");
```

- **int ChangeUserPin([in]NPSTRING strNewPin)**

说明：修改超级狗的用户口令。需要先校验超级狗的用户口令成功。

输入数据：

strNewPin：用于替换的用户口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.ChangeUserPin("111111");
```

- **int GetUserName()**

说明：获取超级狗的用户名字符串。需要在调用该函数之后再通过控件属性 UserNameStr 获取用户名。最大长度：32 字节

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.GetUserName();
var name = Authplg.UserNameStr;
```

- **int GetDogID()**

说明：获取当前会话环境中的超级狗 ID。需要在调用该函数之后再通过控件属性 DogIdStr 获取超级狗 ID。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.GetDogID();
var dogid = Authplg.DogIdStr;
```

- **int GetDigest([in]NPSTRING strChallenge)**

说明：根据输入的字符串生成加密的摘要字符串。需要在调用该函数之后再通过控件属性 DigestStr 获取加密的摘要字符串。需要先校验超级狗的口令成功。

输入数据：

strChallenge：用于生成加密摘要的字符串。即从服务器端获取的挑战数据。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.GetDigest("3702032AA96A457CB619C523E1126340");
var digest = Authplg.DigestStr;
```

- **int SetCheckDogCallBack([in]NPSTRING strInstFunc, [in]NPSTRING strRmvFunc)**

说明：开启服务检测超级狗的插拔。通过参数传递响应超级狗插拔设备的 javascript 函数名称。在调用该函数之后，需要通过控件属性 InsertFunc 与 RemoveFunc 分别传入响应超级狗插拔的 javascript 函数名称。

输入数据：

strInstFunc：用于响应超级狗连接到系统的 javascript 函数名称。

strRmvFunc：用于响应拔掉超级狗的 javascript 函数名称。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.SetCheckDogCallBack("insertfunc", "pullfunc");
Authplg.InsertFunc = insertfunc;
Authplg.RemoveFunc = pullfunc;
```

- **int RegisterUser([in]NPSTRING strUser, [in]NPSTRING strPin)**

说明：注册用户信息。第一次使用超级狗时，将用户名和用户口令写入超级狗默认的数据文件中。

输入数据：

strUser：用户名字符串。最大长度：32 字节。

strPin：用户口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.RegisterUser("Edward", "12345678");
```

- **int SetUserData([in]int iDataType, [in]int iOffset, [in]NPSTRING strData)**

说明：设置自定义用户数据。需要管理员口令或用户口令验证通过。

输入数据：

iDataType：字符串类型为：16 进制格式或普通可显示字符串。

iOffset：存储到超级狗中的偏移量。

strData：自定义用户数据。

16 进制格式字符串，例如："EFD9AE38"，长度必须为偶数，函数内部会把该字符串转换为对应的 ASCII 码存储到超级狗中。字符串长度/2 + iOffset 必须小于或等于 50；

普通可显示字符串，例如："SuperDog"，字符串长度 + iOffset 必须小于或等于 50。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.SetUserData(1, 30, "SuperDog Auth");
```

- **int GetUserData([in]int iDataType, [in]int iOffset, [in]int iDataSize)**

说明：获取自定义用户数据。需要在调用该函数之后再通过控件属性 UserDataStr 获取用户信息。

输入数据：

iDataType：字符串类型为：16 进制格式或普通可显示字符串。

iOffset：存储到超级狗中的偏移量。

iDataSize：获取用户数据的大小。需要比欲取到的数据长度大一个字节。

16 进制格式字符串，例如："EFD9AE38"，(iDataSize – 1)/2 + iOffset 必须小于或等于 50；

普通可显示字符串，例如："SuperDog"，(iDataSize – 1) + iOffset 必须小于或等于 50。

返回数据：

DOG_STATUS_OK： 请求已成功完成

JavaScript 调用示例：

```
Authplg.GetUserData(1, 30, 11);
var data = Authplg.UserDataStr;
```

- **int EnumDog([in]NPSTRING strAuthCode)**

说明：获取当前连接到系统的超级狗的 ID。需要在调用该函数之后再通过控件属性 EnumResultStr 获取超级狗的 ID。

输入数据：

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

返回数据：

DOG_STATUS_OK： 请求已成功完成

返回的超级狗 ID 信息格式如下：

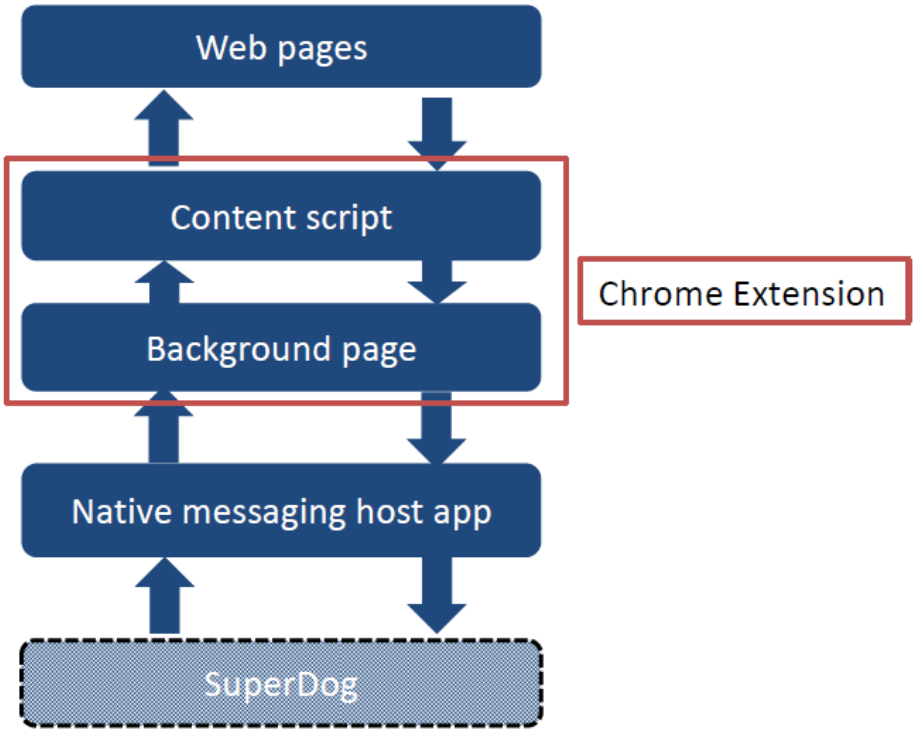
```
<?xml version="1.0" encoding="UTF-8" ?>
<dog_info>
    <dog id="591751055" />
    <dog id="591751069" />
</dog_info>
```

JavaScript 调用示例：

```
Authplg.EnumDog(auth_code);
var dog_info = Authplg.EnumResultStr;
```

Chrome 扩展程序及 Native Messaging Host 应用程序

超级狗认证 Chrome 扩展程序与 Native Messaging Host 应用程序协同工作，支持使用超级狗进行认证。具体原理如下图所示：



此示例将网页与 Chrome 扩展程序的消息传递封装为以下的方法：

- **GetUserNameEx([in]String strScope, [in]String strAuthCode)**

说明：获取存储于超级狗中的用户名字符串。需要在调用该方法之后再通过返回消息的 UserNameStr 字段获取用户名。最大长度：32 字节。如果有多个超级狗连接到当前系统中，请在第一个输入参数中，指定要搜索超级狗 ID 的 XML 格式定义。更多信息，参阅超级狗帮助文档中《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

返回数据：

返回消息的 Status 字段为 0 表示请求成功完成，其它值为错误码；

如果请求成功完成，返回消息包含用户名（UserNameStr 字段）。

JavaScript 调用示例：

```
AuthObject.GetUserNameEx(scope, auth_code);
```

处理返回消息的示例：

```
var ReturnText = event.data.text;
if ("GetUserNameEx" == ReturnText.InvokeMethod) {
```

```
    if (0 == ReturnText.Status) {
        var name = ReturnText.UserNameStr;
    }
}
```

注：这里的 AuthObject 是网页内封装的与超级狗认证 Chrome 扩展程序通信的对象的名称，开发中应根据实际命名进行改变，下同。

- **GetDigestEx([in]String strScope, [in]String strAuthCode, [in]String strPassword, [in]String strChallenge)**

说明：根据输入的字符串生成加密的摘要字符串。需要在调用该方法之后再通过返回消息的 DigestStr 字段获取加密的摘要字符串，通过返回消息的 DogIdStr 字段获取超级狗 ID。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

strPassword：用户口令字符串。范围：6-16 字节。

strChallenge：用于生成加密摘要的字符串。即从服务器端获取的挑战数据。

返回数据：

返回消息的 Status 字段为 0 表示请求成功完成，其它值为错误码；

如果请求成功完成，返回消息包含超级狗 ID（DogIdStr 字段）和加密的摘要字符串（DigestStr 字段）。

JavaScript 调用示例：

```
AuthObject.GetDigestEx(scope, auth_code, "12345678",
"3702032AA96A457CB619C523E1126340");
```

处理返回消息的示例：

```
var ReturnText = event.data.text;
if ("GetDigestEx" == ReturnText.InvokeMethod) {
    if (0 == ReturnText.Status) {
        var dogid = ReturnText.DogIdStr;
        var digest = ReturnText.DigestStr;
    }
}
```

- **RegisterUserEx([in]String strScope, [in]String strAuthCode, [in]String strUserName, [in]String strPassword)**

说明：注册用户信息。第一次使用超级狗时，将用户名和用户口令写入超级狗默认的数据文件中。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

strUserName：用户名字符串。最大长度：32 字节。

strPassword：用户口令字符串。范围：6-16 字节。

返回数据：

返回消息的 Status 字段为 0 表示请求成功完成，其它值为错误码。

JavaScript 调用示例：

```
AuthObject.RegisterUserEx(scope, auth_code, "Edward", "12345678");
```

- **ChangeUserPinEx([in]String strScope, [in]String strAuthCode, [in]String strOldPassword, [in]String strNewPassword)**

说明：修改超级狗的用户口令。需要先校验超级狗的用户口令成功。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

strOldPassword：当前的用户口令字符串。范围：6-16 字节。

strNewPassword：新的用户口令字符串。范围：6-16 字节。

返回数据：

返回消息的 Status 字段为 0 表示请求成功完成，其它值为错误码。

JavaScript 调用示例：

```
AuthObject.ChangeUserPinEx(scope, auth_code, "12345678", "111111");
```

- **SetUserDataEx([in]String strScope, [in]String strAuthCode, [in]String strPassword, [in]int iDataType, [in]int iOffset, [in]String strData)**

说明：设置自定义用户数据。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

strPassword：用户口令字符串。范围：6-16 字节。

iDataType：字符串类型为：16 进制格式或普通可显示字符串。

iOffset：存储到超级狗中的偏移量。

strData：自定义用户数据。

16 进制格式字符串，例如："EFD9AE38"，长度必须为偶数，方法内部会把该字符串转换为对应的 ASCII 码存储到超级狗中。字符串长度/2 + iOffset 必须小于或等于 50；

普通可显示字符串，例如："SuperDog"，字符串长度 + iOffset 必须小于或等于 50。

返回数据：

返回消息的 Status 字段为 0 表示请求成功完成，其它值为错误码。

JavaScript 调用示例：

```
AuthObject.SetUserDataEx(scope, auth_code, "12345678", 1, 30,
"SuperDog Auth");
```

- **GetUserDataEx([in]String strScope, [in]String strAuthCode, [in]int iDataType, [in]int iOffset, [in]int iDataSize)**

说明：获取自定义用户数据。需要在调用该方法之后再通过返回消息的 UserDataStr 字段获取用户数据。

输入数据：

strScope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

iDataType：字符串类型为：16 进制格式或普通可显示字符串。

iOffset：存储到超级狗中的偏移量。

iDataSize：获取用户数据的大小。需要比欲取到的数据长度大一个字节。

16 进制格式字符串，例如："EFD9AE38"，(iDataSize – 1)/2 + iOffset 必须小于或等于 50；

普通可显示字符串，例如："SuperDog"，(iDataSize – 1) + iOffset 必须小于或等于 50。

返回数据：

返回消息的 Status 字段为 0 表示请求成功完成，其它值为错误码；

如果请求成功完成，返回消息包含用户数据（UserDataStr 字段）。

JavaScript 调用示例：

```
Authplg.GetUserDataEx(scope, auth_code, 1, 30, 11);
```

处理返回消息的示例：

```
var ReturnText = event.data.text;
if ("GetUserDataEx" == ReturnText.InvokeMethod) {
    if (0 == ReturnText.Status) {
        var data = ReturnText.UserDataStr;
    }
}
```

- **EnumDog([in]String strAuthCode)**

说明：获取当前连接到系统的超级狗的 ID。需要在调用该方法之后再通过返回消息的 EnumResultStr 字段获取超级狗的 ID。

输入数据：

strAuthCode：认证代码字符串。该内容从认证代码生成工具生成的 xml 文件中取出。

返回数据：

返回消息的 Status 字段为 0 表示请求成功完成，其它值为错误码；

如果请求成功完成，返回消息包含超级狗的 ID（EnumResultStr 字段）。返回的超级狗 ID 信息格式如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<dog_info>
    <dog id="591751055" />
    <dog id="591751069" />
</dog_info>
```

JavaScript 调用示例：

```
AuthObject.EnumDog(auth_code);
```

处理返回消息的示例：

```
var ReturnText = event.data.text;
if ("EnumDog" == ReturnText.InvokeMethod) {
    if (0 == ReturnText.Status) {
        var dog_info = ReturnText.EnumResultStr;
    }
}
```

第五章 服务器端组件

动态链接库

超级狗认证服务器端动态链接库分为 32 位版本（dog_auth_srv.dll）和 64 位版本（dog_auth_srv_x64.dll）。该动态库向服务器端开发者提供了一组可以对客户端进行身份认证的函数接口。

下面对超级狗认证服务器端动态链接库的接口函数进行说明：

- **public native static String getChallenge(int status[])**

说明：生成一个随机的挑战字符串。

输出数据：

status：用于获取函数执行结果。如果执行成功，status[0]返回值为：DOG_STATUS_OK。

返回数据：

生成的随机挑战字符串。

调用示例：

```
String challenge = getChallenge(status);
```

- **public native static int verifyDigest(**
  int vendor_id,
  int dog_id,
  String challenge,
  String digest,
  String factor)

说明：验证输入的挑战字符串与生成的加密摘要字符串是否能够正确匹配。

输入数据：

vendor_id：开发商 ID。

dog_id：超级狗 ID。

challenge：随机生成的挑战字符串。

digest：由客户端生成的加密的摘要字符串。

factor：认证因子。

返回数据：

DOG_STATUS_OK： 验证成功

调用示例：

```
int ires = verifyDigest(37515, 591751299,
"4B2E2D1266241012460276532052663 8",
"288692145D2C611861A26093318F86437F0151A7", "00000000");
```

PHP 扩展组件

超级狗认证服务器端 PHP 扩展组件为 32 位版本（dog_auth_srv_php.dll）。该组件向 PHP 工程的服务器端开发者提供了一组可以对客户端进行身份认证的函数接口。该组件提供源代码工程（目录：WebServer\PHP\Source\dog_auth_srv_php），用于开发者链接相应的 php 源代码版本，编译出对应版本的 PHP 扩展组件。

下面对超级狗认证服务器端 PHP 扩展组件的接口函数进行说明：

- **char* dog_auth_get_challenge()**

说明：生成一个随机的挑战字符串。

返回数据：

生成的随机挑战字符串。如果执行成功，将返回包含 32 个 Base16 字符的字符串。

PHP 调用示例：

```
$strchallenge = dog_auth_get_challenge();
```

- **int dog_auth_verify_digest(**
  int vendor_id,
  int dog_id,
  char challenge,
  char digest,
  char factor)

说明：验证输入的挑战字符串与生成的加密摘要字符串是否能够正确匹配。

输入数据：

vendor_id：开发商 ID。

dog_id：超级狗 ID。

challenge：随机生成的挑战字符串。

digest：由客户端生成的加密的摘要字符串。

factor：认证因子。

返回数据：

DOG_STATUS_OK： 验证成功。

PHP 调用示例：

```
$ires = dog_auth_verify_digest(37515, 591751299,
"4B2E2D126624101246027653205266 38",
"288692145D2C611861A26093318F86437F0151A7", "00000000");
```

第六章 超级狗认证动态链接库

超级狗认证动态链接库的文件名是 dogauthdsp.dll。该动态库由初始化工具调用，向开发者提供了一组可以利用超级狗内部的默认文件进行身份认证的函数接口。

下面对超级狗认证动态链接库的接口函数进行说明：

- **dog_status_t DOG_CALLCONV dog_auth_open(**
  const char *scope,
  const char *auth_code,
  auth_handle_t *handle)

说明：打开目前系统中的一个超级狗。如果有多个超级狗连接到当前系统中，请在第一个输入参数中，指定要搜索超级狗 ID 的 XML 格式定义。更多信息，参阅超级狗帮助文档中 《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

输入数据：

scope：要搜索的数据的 XML 格式定义。更多信息，参阅超级狗帮助文档中 《超级狗 Licensing API 帮助》的“范围输入 XML 标记”一节。

auth_code：指向认证代码的指针。该内容从**认证代码生成工具**生成的 xml 文件中取出。

handle：指向生成的会话句柄的指针。

返回数据：

DOG_STATUS_OK：请求已成功完成

调用示例：

```
dog_auth_open(scope, auth_code, &handle);
```

- **dog_status_t DOG_CALLCONV dog_auth_close(**
  auth_handle_t handle)

说明：关闭与当前会话关联的超级狗。

输入数据：

handle：指向会话句柄的指针。

返回数据：

DOG_STATUS_OK：请求已成功完成

调用示例：

```
dog_auth_close(handle);
```

- **dog_status_t DOG_CALLCONV dog_auth_verify_pin(**
  auth_handle_t handle,
  dogauth_pin_t pin_type,

const char *current_pin)

说明：校验超级狗的管理员口令或用户口令。

输入数据：

handle：会话句柄。

pin_type：口令类型，可以是管理员口令或者用户口令。

current_pin：欲验证的口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK：请求已成功完成

调用示例：

校验管理员口令

```
dog_auth_verify_pin(handle, PINTYPE_SO_PIN, "abcdefgh");
```

校验用户口令

```
dog_auth_verify_pin(handle, PINTYPE_USER_PIN, "12345678");
```

- **dog_status_t DOG_CALLCONV dog_auth_change_pin(**
  auth_handle_t handle,
  dogauth_pin_t pin_type,
  const char *new_pin)

说明：修改超级狗的口令。需要先校验超级狗的口令成功。校验超级狗管理员口令成功后，可以修改管理员口令与用户口令。校验超级狗用户口令成功后，只能修改用户口令。

输入数据：

handle：会话句柄。

pin_type：管理员口令或者用户口令。

new_pin：用于替换的口令字符串。范围：6-16 字节。

返回数据：

DOG_STATUS_OK：请求已成功完成

调用示例：

```
dog_auth_change_pin(handle, PINTYPE_SO_PIN, "111111");
```

- **dog_status_t DOG_CALLCONV dog_auth_get_digest(**
  auth_handle_t handle,
  const char *challenge,
  char *digest,
  dog_size_t *digest_size)

说明：根据输入的挑战字符串生成加密的摘要字符串。调用此函数之前需要先成功校验超级狗的口令。

输入数据：

handle：会话句柄。

challenge：从服务器端获取的挑战字符串，用于生成加密的摘要字符串。挑战字符串长度为 32 字节。

digest_size：指定存储获取的摘要字符串缓冲区的大小。长度至少为 41 字节。

输出数据：

digest：指向获取的加密摘要字符串的指针。生成的摘要字符串的长度为 40 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

调用示例：

```
dog_auth_get_digest(handle, challenge, digest, 41);
```

- **dog_status_t DOG_CALLCONV dog_auth_set_factor(**
  auth_handle_t handle,
  const char *factor)

说明：设置认证因子。认证因子将在生成加密的摘要字符串中使用。需要先校验超级狗的管理员口令成功。

输入数据：

handle：会话句柄。

factor：认证因子字符串。长度为 8 字节。

返回数据：

DOG_STATUS_OK： 请求已成功完成

调用示例：

```
dog_auth_set_factor(handle, "cv6ejrbj");
```

- **dog_status_t DOG_CALLCONV dog_auth_get_dogid(**
  auth_handle_t handle,
  char *dog_id,
  dog_size_t *id_size)

说明：获取当前会话的超级狗 ID。

输入数据：

handle：会话句柄。

id_size：指定存储获取的超级狗 ID 字符串的缓冲区的大小。

输出数据：

dog_id：指向存储获取的超级狗 ID 字符串的缓冲区的指针。

返回数据：

DOG_STATUS_OK： 请求已成功完成

调用示例：

```
dog_auth_get_dogid(handle, dogid, id_size);
```

- **dog_status_t DOG_CALLCONV dog_auth_set_username(**
  auth_handle_t handle,
  const char *user_name)

说明：设置用户名。最大长度：32 字节。需要管理员口令或用户口令验证通过。

输入数据：

handle：会话句柄。

user_name：要设置的用户名字符串。

返回数据：

DOG_STATUS_OK：请求已成功完成

调用示例：

```
dog_auth_set_username(handle, "demouser");
```

- **dog_status_t DOG_CALLCONV dog_auth_get_username(**
  auth_handle_t handle,
  char *user_name,
  dog_size_t *name_size)

说明：获取用户名。最大长度：32 字节。

输入数据：

handle：会话句柄。

name_size：用于存储获取的用户名的缓冲区的大小。

输出数据：

user_name：指向存储获取的用户名的缓冲区的指针。

返回数据：

DOG_STATUS_OK：请求已成功完成

调用示例：

```
dog_auth_get_username(handle, tmpbuf, buf_size);
```

- **dog_status_t DOG_CALLCONV dog_auth_set_userdata(**
  auth_handle_t handle,
  dog_data_t data_type,
  dog_size_t offset,

const char *user_data)

说明：设置自定义用户信息。需要管理员口令或用户口令验证通过。

输入数据：

handle：会话句柄。

data_type：字符串类型为：16 进制格式或普通可显示字符串。

offset：存储到超级狗中的偏移量。

user_data：自定义用户数据。

16 进制格式字符串，例如："EFD9AE38"，长度必须为偶数，函数内部会把该字符串转换为对应的 ASCII 码存储到超级狗中。字符串长度/2 + offset 必须小于或等于 50；

普通可显示字符串，例如："SuperDog"，字符串长度 + offset 必须小于或等于 50。

返回数据：

DOG_STATUS_OK： 请求已成功完成

调用示例：

```
dog_auth_set_userdata(handle, DATA_TYPE_STRING, 3, "SuperDog Auth");
```

- **dog_status_t DOG_CALLCONV dog_auth_get_userdata(**
  auth_handle_t handle,
  dog_data_t data_type,
  dog_size_t offset,
  dog_size_t length,
  char *user_data)

说明：获取自定义用户信息。

输入数据：

handle：会话句柄。

data_type：字符串类型为：16 进制格式或普通可显示字符串。

offset：用户信息块的偏移量。

length：想要获取用户信息的长度。

注意：请预留足够的空间以存储获取的用户信息。根据数据类型的不同，需要的大小计算如下：

16 进制格式字符串，例如："EFD9AE38"，空间须大于 length * 2

普通可显示字符串，例如："SuperDog"，空间须大于 length

输出数据：

user_data：指向存储获取的用户信息的缓冲区的指针。

返回数据：

DOG_STATUS_OK： 请求已成功完成

调用示例：

```
dog_auth_get_userdata(handle, DATA_TYPE_STRING, 0, 7, buffer, 8);
```

第七章 超级狗认证状态代码

以下是运行超级狗认证示例可能返回的状态代码列表。

编号	状态代码	说明
0	DOG_STATUS_OK	请求已成功完成
1	DOG_MEM_RANGE	请求超出数据文件的范围
3	DOG_INSUF_MEM	系统内存不足
4	DOG_TMOF	打开的登录会话数目过多
5	DOG_ACCESS_DENIED	访问被拒绝
7	DOG_NOT_FOUND	未找到所需的超级狗
8	DOG_TOO_SHORT	加密/解密的数据长度太短
9	DOG_INV_HND	输入函数的句柄无效
10	DOG_INV_FILEID	无法识别文件标识符
15	DOG_INV_FORMAT	无效的 XML 格式
16	DOG_REQ_NOT_SUPP	当前会话环境不支持的功能
17	DOG_INV_UPDATE_OBJ	无效的升级内容
18	DOG_KEYID_NOT_FOUND	未找到待升级的超级狗
19	DOG_INV_UPDATE_DATA	未找到所需的 XML 标记，或者二进制数据的内容已丢失或无效。
20	DOG_INV_UPDATE_NOTSUPP	该超级狗不支持升级请求
21	DOG_INV_UPDATE_CNTR	升级计数器设置不正确
22	DOG_INV_VCODE	输入的开发商代码无效
24	DOG_INV_TIME	输入的时间值超出被支持的数值范围
26	DOG_NO_ACK_SPACE	升级要求回执数据，但输入参数 ack_data 为 NULL。
27	DOG_TS_DETECTED	程序在终端服务器上运行
29	DOG_UNKNOWN_ALG	V2C 文件中使用了未知算法
30	DOG_INV_SIG	签名验证失败
31	DOG_FEATURE_NOT_FOUND	特征不可用
33	DOG_LOCAL_COMM_ERR	API 和超级狗运行环境（License Manager）通讯错误
34	DOG_UNKNOWN_VCODE	API 不识别开发商代码
35	DOG_INV_SPEC	无效的 XML 格式
36	DOG_INV_SCOPE	无效的 XML 范围
37	DOG_TOO_MANY_KEYS	当前连接的超级狗数目过多
39	DOG_BROKEN_SESSION	会话被中断
41	DOG_FEATURE_EXPIRED	特征已失效
42	DOG_OLD_LM	超级狗的运行环境版本太旧
43	DOG_DEVICE_ERR	与超级狗通讯中出现 USB 通信错误
45	DOG_TIME_ERR	系统时钟已被篡改
46	DOG_SCHAN_ERR	安全通道中发生了通信错误
47	DOG_STORAGE_CORRUPT	超级狗软许可数据被破坏

50	DOG_SCOPE_RESULTS_EMPTY	不能找到与范围匹配的特征
52	DOG_HARDWARE_MODIFIED	超级狗软许可与设备不匹配
54	DOG_UPDATE_TOO_OLD	文件中的升级计数器的数值小于超级狗中的升级计数器的数值，不允许安装 V2C 文件。
55	DOG_UPDATE_TOO_NEW	文件中的升级计数器的数值大于超级狗中的升级计数器的数值，不允许安装 V2C 文件。
64	DOG_CLONE_DETECTED	发现克隆的超级狗软许可
65	DOG_UPDATE_ALREADY_ADDED	已经使用过当前的超级狗升级文件
78	DOG_SECURE_STORE_ID_MISMATCH	超级狗安全存储 ID 与当前应用程序不匹配
400	DOG_NO_API_DYLIB	未找到 API 的动态库
401	DOG_INVALID_API_DYLIB	API 的动态库无效
500	DOG_INVALID_OBJECT	对象的初始化不正确
501	DOG_INVALID_PARAMETER	无效的函数参数
502	DOG_ALREADY_LOGGED_IN	两次登录到同一对象
503	DOG_ALREADY_LOGGED_OUT	从同一对象注销两次
525	DOG_OPERATION_FAILED	系统或平台的使用不正确
698	DOG_NOT_IMPL	未实施所请求的功能
699	DOG_INT_ERR	API 中内部错误
802	DOGAUTH_ERR_PARAM_NULL	参数为空
803	DOGAUTH_ERR_AUTHCODE_LEN	认证代码长度不正确
804	DOGAUTH_ERR_NOT_LOGIN	请先登录
810	DOGAUTH_ERR_PIN_LEN	口令长度不正确
811	DOGAUTH_ERR_PARAM_LEN	参数长度不正确
812	DOGAUTH_ERR_USERDATA_LEN	用户数据长度不正确
813	DOGAUTH_ERR_USERNAME_LEN	用户名长度不正确
814	DOGAUTH_ERR_FACTOR_LEN	认证因子长度不正确
815	DOGAUTH_ERR_DOGID_LEN	超级狗 ID 长度不正确
821	DOGAUTH_SO_PIN_NOTVERIFY	请先验证管理员口令
822	DOGAUTH_PIN_NOTVERIFY	请先验证用户口令
823	DOGAUTH_LEN_TOO_SHORT	缓冲区长度不足
824	DOGAUTH_INIT_FAILED	认证动态库初始化失败
825	DOGAUTH_ERR_PIN_LOCKED	用户口令被锁定
831	DOGAUTH_VERIFY_PIN_FAILED_01	验证用户口令失败（累计：1 次）
832	DOGAUTH_VERIFY_PIN_FAILED_02	验证用户口令失败（累计：2 次）
833	DOGAUTH_VERIFY_PIN_FAILED_03	验证用户口令失败（累计：3 次）
834	DOGAUTH_VERIFY_PIN_FAILED_04	验证用户口令失败（累计：4 次）
835	DOGAUTH_VERIFY_PIN_FAILED_05	验证用户口令失败（累计：5 次）
836	DOGAUTH_VERIFY_PIN_FAILED_06	验证用户口令失败（累计：6 次）
837	DOGAUTH_VERIFY_PIN_FAILED_07	验证用户口令失败（累计：7 次）
838	DOGAUTH_VERIFY_PIN_FAILED_08	验证用户口令失败（累计：8 次）
839	DOGAUTH_VERIFY_PIN_FAILED_09	验证用户口令失败（累计：9 次）
840	DOGAUTH_VERIFY_PIN_FAILED_10	验证用户口令失败（累计：10 次）

841	DOGAUTH_VERIFY_PIN_FAILED_11	验证用户口令失败（累计：11 次）
842	DOGAUTH_VERIFY_PIN_FAILED_12	验证用户口令失败（累计：12 次）
843	DOGAUTH_VERIFY_PIN_FAILED_13	验证用户口令失败（累计：13 次）
844	DOGAUTH_VERIFY_PIN_FAILED_14	验证用户口令失败（累计：14 次）
845	DOGAUTH_VERIFY_PIN_FAILED_15	验证用户口令失败（累计：15 次），用户口令被锁定
871	DOGAUTH_LOAD_FILE_FAILED	打开或读取文件失败
872	DOGAUTH_STORE_FILE_FAILED	打开或写入文件失败
873	DOGAUTH_DOG_NOT_FOR_AUTH	C2V 文件代表的超级狗不是用于认证用途
901	DOGAUTH_AUTHENTICATE_FAILED	认证失败
902	DOGAUTH_GEN_CHALLENGE_STR_FAILED	生成挑战数据失败
903	DOGAUTH_ERR_CHARACTER_IN_NAME	用户名包含不支持的字符
904	DOGAUTH_ERR_EMPTY_PIN	请输入口令
905	DOGAUTH_ERR_PIN_LENGTH	口令长度须在 6-16 字节之间
906	DOGAUTH_ERR_CHARACTER_IN_PIN	口令包含不支持的字符
907	DOGAUTH_ERR_EMPTY_NAME	请输入用户名
908	DOGAUTH_ERR_EMPTY_CONFIRM_PIN	请再次输入口令
909	DOGAUTH_ERR_PIN_LENGTH_1	口令长度须在 6-16 字节之间
910	DOGAUTH_ERR_CHARACTER_IN_PIN_1	口令包含不支持的字符
911	DOGAUTH_PIN_AND_CONFIRM_PIN_MISMATCH	口令和确认口令不一致
912	DOGAUTH_ERR_EMPTY_OLD_PIN	请输入当前口令
913	DOGAUTH_ERR_EMPTY_NEW_PIN	请输入新口令
914	DOGAUTH_ERR_NAME_LENGTH	用户名长度须在 1-32 字节之间
915	DOGAUTH_ALREADY_REGISTERED	此超级狗已经注册过，不支持再次注册
916	DOGAUTH_LOAD_LIBRARY_FAILED	在 java.library.path 指定的文件夹找不到 dog_auth_srv 库文件
917	DOGAUTH_GET_CHALLENGE_STR_FAILED_1	获取挑战数据失败
918	DOGAUTH_GET_CHALLENGE_STR_FAILED_2	获取挑战数据失败
919	DOGAUTH_GET_SESSION_FILE_FAILED	找不到会话文件，请确认会话文件夹已经正确创建和设置。
920	DOGAUTH_LOAD_PHP_LIBRARY_FAILED	加载动态库失败：dog_auth_srv_php.dll，请确认配置文件已经正确设置。