# Windows Hello应用 动手实验手册

## 概述

Windows Hello是一个基于生物特征授权的智能身份认证系统。有了它，用户可以在不使用密码的情况下，通过脸部特征、虹膜或指纹等生物特征来解锁设备。Windows 10 SDK为开发者提供了Windows Hello的API。开发人员通过这些API，可以构建自己的安全策略，将自己的鉴权系统与Windows Hello相结合，开发更炫酷的应用。

## 通过示例您将学会

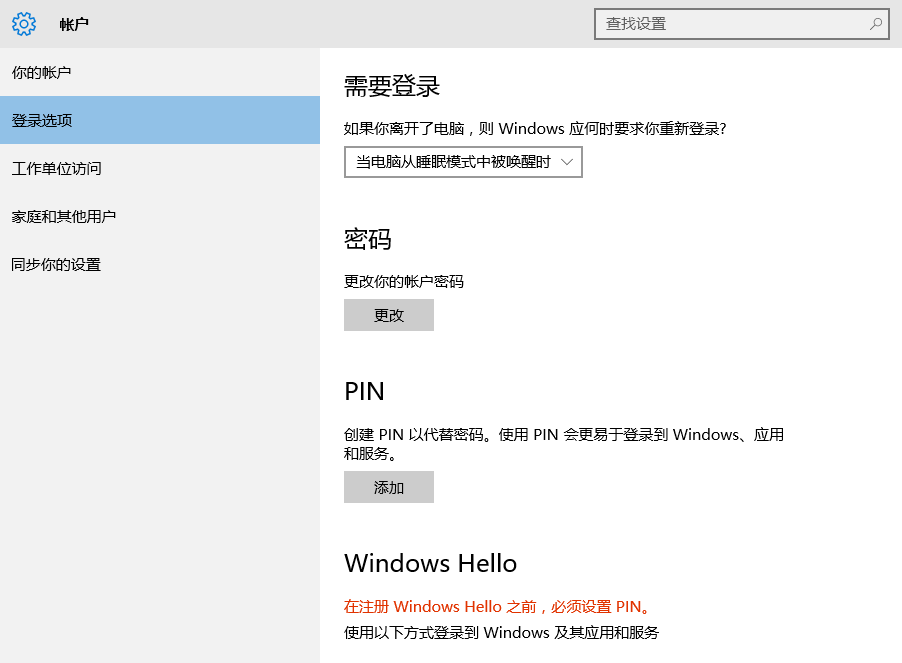
* 本地密码凭据管理。凭据创建、凭据获取。
* Windows Hello启用。PIN码使用。

## 准备工作

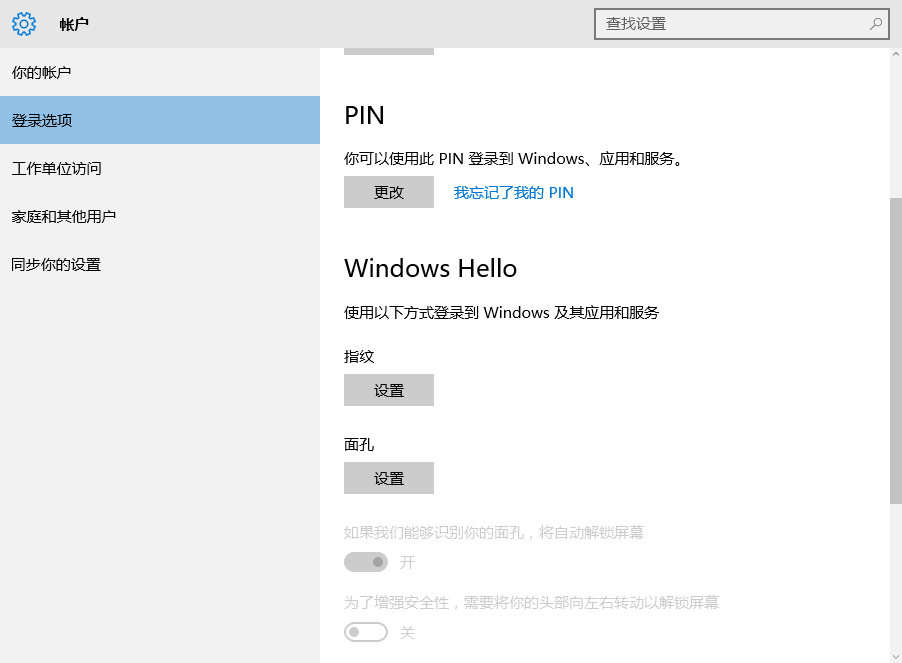
Windows 10 要求必须在包含生物特征传感器的设备上启用PIN码，才能使用Windows Hello来管理登录凭据。下面，我们来演示如何启用PIN码。（如果您的已启用PIN码，请忽略这个段落的内容）

**设置PIN码**

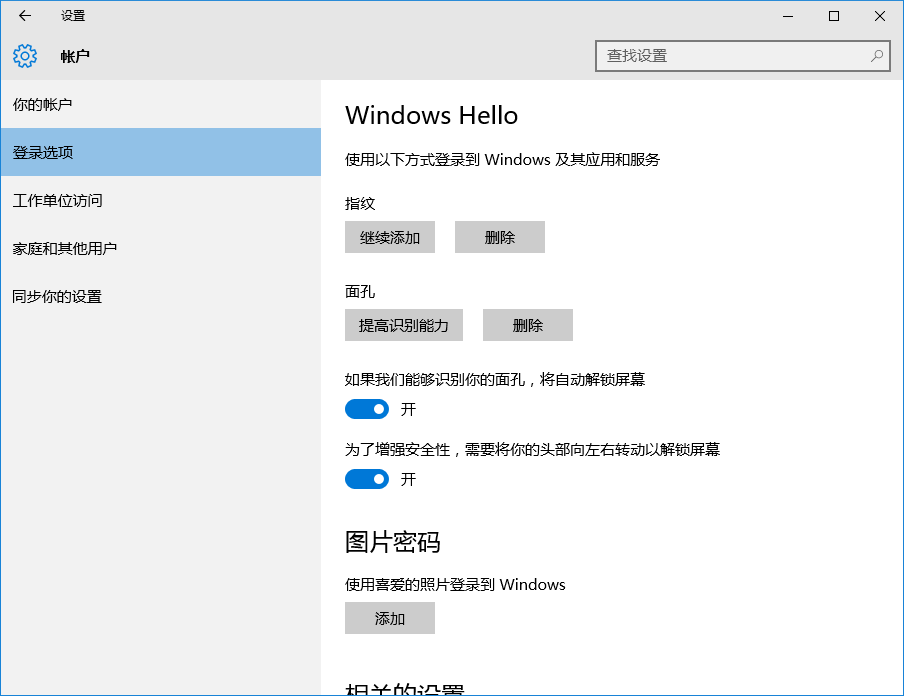
打开“开始”菜单，点击设置-登录选项。



在“登录选项”面板下，添加一个PIN码后，即可开启Windows Hello。



在Windows Hello 选项下面，我们可以设置指纹、面孔，通过设备的传感器来采集我们的生物特征。



## 挑战

我们将使用Visual Studio 2015 打开SimpleHello项目，来学习Windows Hello相关API。

— 打开SimpleHello项目。

## 创建场景

### 场景描述

在这个场景里面，主要演示了如何使用Windows Hello API来管理账户凭据，该场景演示了一个银行账户的登录过程。用户在首次登录时，应用程序会检测当前设备是否支持Windows Hello。如果设备支持，应用程序会使用将当前账户凭据，来创建本地的密码凭据。用户在下次登录时，可以通过传感器设备来验证生物特征，同时读取并验证本地凭据，当凭据验证成功后，用户可以看到银行账户的详细情况。

### 知识点

Windows Hello 密钥凭据管理是使用Windows.Security.Credentials命名空间下的KeyCredentialManager类。

**KeyCredentialManager类，该类用于本地秘钥凭据的管理。主要方法如下：**

**RequestCreateAsync(System.String name, KeyCredentialCreationOption option)**：用于创建本地凭据，该方法返回密码凭据的一个检索结果类KeyCredentialRetrievalResult。（该方法会调用Windows Hello，通过本地生物特征识别来返回生物特征验证结果）

**KeyCredentialCreationOption**参数：是一个创建操作枚举，主要有ReplaceExisting(密码凭据存在则替换)、FailIfExists(密码凭据存在，返回错误)两个选项。

**OpenAsync(System.String name)**：根据密码凭据用户名打开本地密码凭据，该方法返回密码凭据的一个检索结果类KeyCredentialRetrievalResult。（该方法会调用Windows Hello，通过本地生物特征识别来返回生物特征验证结果）

**IsSupportedAsync()**：该方法返回一个布尔类型，用于判断当前设备是否支持Windows Hello。

**KeyCredentialRetrievalResult**：该类用于保存密码凭据的检索结果。它包含KeyCredential、KeyCredentialStatus两个属性。

**KeyCredentialStatus**：是一个枚举类，定义了密钥凭据的检索结果。

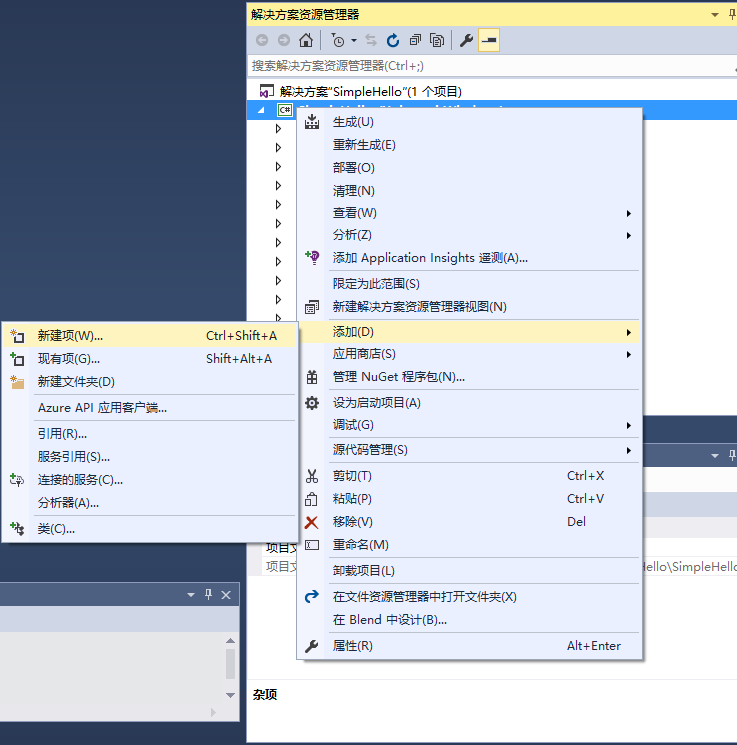
**KeyCredential**：该类用于保存秘钥凭据信息，主要方法如下：

**GetAttestationAsync()**：该方法返回一个鉴权认证结果。

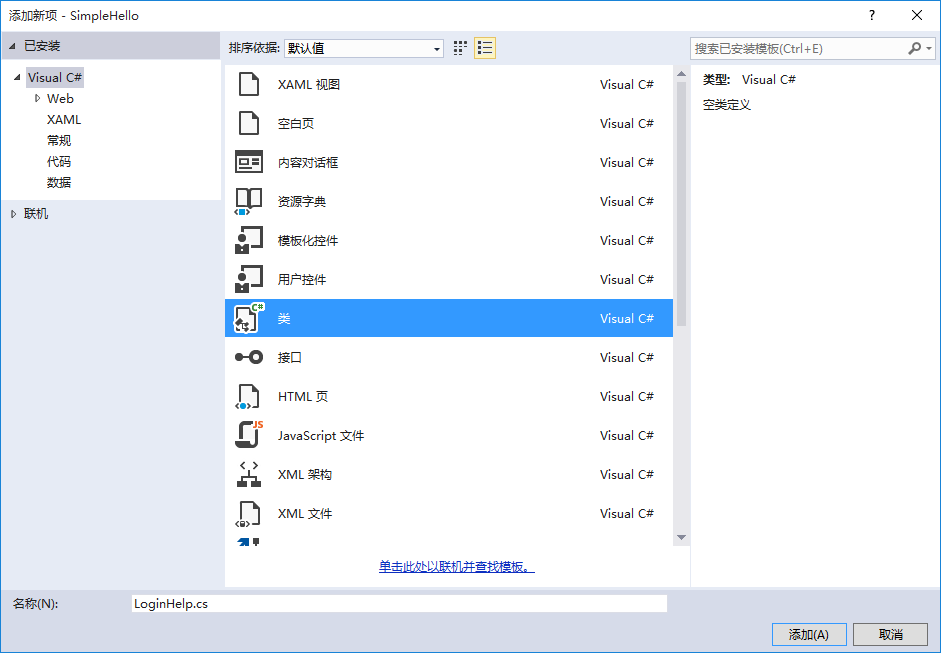
**RequestSignAsync(IBuffer data)**：该方法返回一个密码凭证验证结果。

### 创建登录辅助类

在“解决方案资源管理器”中，右键单击该项目SimpleHello。从“添加”菜单中，选择“新建项”



在左侧Visual C# 模板中，可以选择文件类型，我们添加一个“类”用来实现密钥的管理。基于我们的场景，我们建议的命名是LoginHelp.cs。它主要用于示例的登录辅助功能。



打开LoginHelp.cs 文件，添加密钥管理及登录相关方法，代码如下：

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using Windows.Security.Credentials;  using Windows.Security.Cryptography;  using Windows.Storage.Streams;  namespace SimpleHello  {  public class LoginHelp  {  private Account activeAccount;  public LoginHelp(Account account)  {  this.activeAccount = account;  }  /// <summary>  /// The function called when the user wants to sign in with Passport but from  /// the username/password dialog.  ///  /// First we check if they already use Passport as their primary login method  ///  /// Otherwise we authenticate their username/password and then begin the Passport  /// enrollment process which consists of creating a Passport key and then sending that to  /// the authentication server.  /// </summary>  public async Task<bool> SignInPassport()  {  if (await AuthenticatePassport() == true)  {  SuccessfulSignIn(false);  return true;  }  else  {  return false;  }  }  /// <summary>  /// Username and password authentication  /// </summary>  /// <param name="isnewAccount">true:new user,false:old user</param>  /// <returns></returns>  public async Task<bool> SignInPassword(bool isnewAccount)  {  try  {  bool signedIn = await AuthenticatePasswordCredentials();  if (signedIn == false)  {  return false;  }  else  {  if(isnewAccount)  SuccessfulSignIn(true);  return true;  }  }  catch (Exception e)  {  return false;  }  }  /// <summary>  /// Authenticate user credentials with server and return result.  /// </summary>  /// <returns>Boolean representing if authenticating the user credentials with the server succeeded</returns>  private async Task<bool> AuthenticatePasswordCredentials()  {  // TODO: Authenticate with server once that part is done for the sample.  return true;  }  /// <summary>  /// Function to be called when we need to register our public key with  /// the server for Microsoft Passport  /// </summary>  /// <returns>Boolean representing if adding the Passport login method to this account on the server succeeded</returns>  public async Task<bool> AddPassportToAccountOnServer()  {  // TODO: Add Passport signing info to server when that part is done for the sample  return true;  }  /// <summary>  /// Attempts to get the authentication message from the Passport key for this account.  ///  /// This will bring up the Passport PIN dialog and prompt the user for their PIN.  ///  /// The authentication message will be null if signing fails.  /// </summary>  /// <returns>Boolean representing if authenticating Passport succeeded</returns>  private async Task<bool> AuthenticatePassport()  {  IBuffer message = CryptographicBuffer.ConvertStringToBinary("LoginAuth", BinaryStringEncoding.Utf8);  IBuffer authMessage = await GetPassportAuthenticationMessage(message, this.activeAccount.Email);  if (authMessage != null)  {  return true;  }  return false;  }  /// <summary>  /// Handles user saving for our list of users if this is a new user  /// </summary>  private void SuccessfulSignIn(bool isAdd)  {  // If this is an already existing account, replace the old  // version of this account in the account list.  if (isAdd == false)  {  foreach (Account a in UserSelect.accountList)  {  if (a.Email == this.activeAccount.Email)  {  UserSelect.accountList.Remove(a);  break;  }  }  }  UserSelect.accountList.Add(this.activeAccount);  AccountsHelper.SaveAccountList(UserSelect.accountList);  }  /// <summary>  /// Attempts to sign a message using the Passport key on the system for the accountId passed.  /// </summary>  /// <param name="message">The message to be signed</param>  /// <param name="accountId">The account id for the Passport key we are using to sign</param>  /// <returns>Boolean representing if creating the Passport authentication message succeeded</returns>  private async Task<IBuffer> GetPassportAuthenticationMessage(IBuffer message, string accountId)  {  KeyCredentialRetrievalResult openKeyResult = await KeyCredentialManager.OpenAsync(accountId);  if (openKeyResult.Status == KeyCredentialStatus.Success)  {  KeyCredential userKey = openKeyResult.Credential;  IBuffer publicKey = userKey.RetrievePublicKey();  KeyCredentialOperationResult signResult = await userKey.RequestSignAsync(message);  if (signResult.Status == KeyCredentialStatus.Success)  {  return signResult.Result;  }  else if (signResult.Status == KeyCredentialStatus.UserCanceled)  {  // User cancelled the Passport PIN entry.  //  // We will return null below this and the username/password  // sign in form will show.  }  else if (signResult.Status == KeyCredentialStatus.NotFound)  {  // Must recreate Passport key  }  else if (signResult.Status == KeyCredentialStatus.SecurityDeviceLocked)  {  // Can't use Passport right now, remember that hardware failed and suggest restart  }  else if (signResult.Status == KeyCredentialStatus.UnknownError)  {  // Can't use Passport right now, try again later  }  return null;  }  else if (openKeyResult.Status == KeyCredentialStatus.NotFound)  {  // Passport key lost, need to recreate it  }  else  {  // Can't use Passport right now, try again later  }  return null;  }  /// <summary>  /// Creates a Passport key on the machine using the account id passed.  /// Then returns a boolean based on whether we were able to create a Passport key or not.  ///  /// Will also attempt to create an attestation that this key is backed by hardware on the device, but is not a requirement  /// for a working key in this scenario. It is possible to not accept a key that is software-based only.  /// </summary>  /// <param name="accountId">The account id associated with the account that we are enrolling into Passport</param>  /// <returns>Boolean representing if creating the Passport key succeeded</returns>  public async Task<bool> CreatePassportKey(string accountId)  {  KeyCredentialRetrievalResult keyCreationResult = await KeyCredentialManager.RequestCreateAsync(accountId, KeyCredentialCreationOption.ReplaceExisting);  if (keyCreationResult.Status == KeyCredentialStatus.Success)  {  KeyCredential userKey = keyCreationResult.Credential;  IBuffer publicKey = userKey.RetrievePublicKey();  KeyCredentialAttestationResult keyAttestationResult = await userKey.GetAttestationAsync();  if (keyAttestationResult.Status == KeyCredentialAttestationStatus.Success)  {  //keyAttestation Included.  //TODO:read keyAttestationResult.AttestationBuffer and keyAttestationResult.CertificateChainBuffer  }  else if (keyAttestationResult.Status == KeyCredentialAttestationStatus.TemporaryFailure)  {  //keyAttestation CanBeRetrievedLater  }  else if (keyAttestationResult.Status == KeyCredentialAttestationStatus.NotSupported)  {  //keyAttestation is not supported  }  // Package public key, keyAttesation if available,  // certificate chain for attestation endorsement key if available,  // status code of key attestation result: keyAttestationIncluded or  // keyAttestationCanBeRetrievedLater and keyAttestationRetryType  // and send it to application server to register the user.  bool serverAddedPassportToAccount = await AddPassportToAccountOnServer();  if (serverAddedPassportToAccount == true)  {  return true;  }  }  else if (keyCreationResult.Status == KeyCredentialStatus.UserCanceled)  {  // User cancelled the Passport enrollment process  }  else if (keyCreationResult.Status == KeyCredentialStatus.NotFound)  {  // User needs to create PIN  return false;  }  return false;  }  }  } |

* **public async Task<bool> SignInPassport():**该方法用于Passport登录，如果成功设置PIN码，调用该方法会调用Windows Hello API来进行用户身份鉴权。
* **public async Task<bool> SignInPassword(bool isnewAccount)**：该方法用于用户输入用户名、密码鉴权登录，如果验证成功，我们会将用户名缓存在本地文件中。为用户的下一次登录提供用户列表选择。

**isnewAccount 参数：**当前用户是否新用户。（如果从用户列表选择用户登录为false，如果用户第一次登录为True）

* **private async Task<bool> AuthenticatePasswordCredentials()：**该方法用于服务器用户身份鉴权。（我们可以构建自己的鉴权服务，通过扩展该方法来完成APP与服务器的数据交互,用于用户的身份鉴权）。
* **public async Task<bool> AddPassportToAccountOnServer()：**该方法用于上传Passport信息到我们自己的服务器，用作服务器鉴权。（通过扩展该方法我们可以上传Passport信息）
* **private async Task<bool> AuthenticatePassport()：**该方法用于调用Windows Hello API来完成本地身份鉴权。
* **private void SuccessfulSignIn(bool isAdd)：**该方法用于用户登录信息的持久化。
* **private async Task<IBuffer> GetPassportAuthenticationMessage(IBuffer message, string accountId)：**该方法用于本地用户身份鉴权。

**message 参数：**本地数据二进制流。

**accountId参数：**登录用户唯一标识。（通常为用户的邮箱，在这里我们使用用户名）

* **public async Task<bool> CreatePassportKey(string accountId):**该方法通过调用Windows Hello API 来创建本地鉴权数据。

**accountId 参数：**用户唯一标识。（通常为用户的邮箱，在这里我们使用用户名）