Contents

以 .NET Framework 進行網路程式設計

2020/3/20 • • Edit Online

Microsoft .NET Framework 提供有層次、可擴充和網際網路服務的 Managed 實作，可以迅速而簡易地整合到您的應用程式。您的網路應用程式可以建置在可外掛式通訊協定上，以便自動利用新的網際網路通訊協定，或者也可以使用 Windows Socket 介面的 Managed 實作，以便搭配使用通訊端層級上的網路。

本節內容

可插式通訊協定簡介
說明如何存取網際網路資源而不用考慮需要的存取通訊協定。

要求資料
說明如何使用可外掛式通訊協定從網際網路資源上傳和下載資料。

可插式通訊協定程式設計
說明如何衍生通訊協定特定類別以實作可外掛式通訊協定。

使用應用程式通訊協定
說明如何設計使用網路通訊協定 (例如 TCP、UDP 和 HTTP) 的應用程式。

互聯網協定版本 6
說明在目前版本的網際網路通訊協定組合 (IPv4) 之上網際網路通訊協定第 6 版 (IPv6) 的優點，描述 IPv6 位址、路由和自動設定，以及如何啟用和停用 IPv6。

設定網際網路應用程式
說明如何使用 .NET Framework 組態檔以設定網際網路應用程式。

以 .NET Framework 進行網路追蹤
說明如何使用網路追蹤以取得有關方法叫用及 Managed 應用程式所產生的網路流量的資訊。

網路應用程式的快取管理
說明如何針對使用 System.Net.WebClient、System.Net.WebRequest和 System.Net.HttpWebRequest 類別的應用程式來使用快取。

網路程式設計的安全性
描述如何使用標準網際網路安全性和驗證技術。

System.Net 類別的最佳做法
提供可讓您善用網際網路應用程式的祕訣和訣竅。

通過代理訪問互聯網
描述如何設定 Proxy。

NetworkInformation
描述如何蒐集網路事件、變更、統計資料和屬性的相關資訊，以及說明如何判斷遠端主機是否可以使用 System.Net.NetworkInformation.Ping 類別取得聯繫。

版本 2.0 中對系統.Uri 命名空間的更改
說明在版本 2.0 中對 System.Uri 類別所做的幾個變更，以修正不正確的行為、提高可用性，以及增強安全性。

System.Uri 的國際資源識別項支援
說明在版本 3.5、3.0 SP1 和 2.0 SP1 中的 System.Uri 類別增強功能，以支援國際資源識別項 (IRI) 和國際化網域名稱 (IDN)。

3.5 版中的通訊端效能增強功能

說明在版本 3.5、3.0 SP1 和 2.0 SP1 中 System.Net.Sockets.Socket 類別的一組增強功能，其提供另一種非同步模式，可供專業化的高效能通訊端應用程式使用。

對等名稱解析通訊協定

說明在版本 3.5 中所新增的支援，以支援對等名稱解析通訊協定 (PNRP)、無伺服器和動態名稱登錄以及名稱解析通訊協定。這些新功能是由 System.Net.PeerToPeer 命名空間所支援。

對等共同作業

說明在版本 3.5 中所新增的支援，以支援建置在 PNRP 上的對等協同作業。這些新功能是由 System.Net.PeerToPeer.Collaboration 命名空間所支援。

3.5 SP1 版中 HttpWebRequest 之 NTLM 驗證的變更

說明在版本 3.5 SP1 中所做的安全性變更，這些變更會影響 System.Net.HttpWebRequest、System.Net.HttpListener、System.Net.Security.NegotiateStream以及 System.Net 命名空間中的相關類別處理整合式 Windows 驗證的方式。

Integrated Windows Authentication with Extended Protection

說明延伸保護的增強功能，這些增強功能會影響 System.Net.HttpWebRequest、System.Net.HttpListener、System.Net.Mail.SmtpClient、System.Net.Security.SslStream、System.Net.Security.NegotiateStream以及 System.Net 和相關命名空間中的相關類別處理整合式 Windows 驗證的方式。

使用 IPv6 和 Teredo 的 NAT 周遊

說明 System.Net、System.Net.NetworkInformation和 System.Net.Sockets 命名空間所新增的增強功能，以支援使用 IPv6 和 Teredo 進行 NAT 周遊。

Windows 市集應用程式的網路隔離

描述在 Windows 8.x 應用商店System.Net應用中System.Net.Http使用System.Net.Http.Headers中的 類和命名空間時網路隔離的影響。

網路程式設計範例

可供下載的網路程式設計範例的連結，這些範例會使用 System.Net、System.Net.Cache、System.Net.Configuration、System.Net.Mail、System.Net.Mime、System.Net.NetworkInformation、System.Net.PeerToPeer、System.Net.Security以及 System.Net.Sockets 命名空間中的類別。

參考

System.Net

提供一個簡單的程式設計介面，讓現今網路所用的許多通訊協定使用。此命名空間中的 System.Net.WebRequest 和 System.Net.WebResponse 類別是可外掛式通訊協定的基礎。

System.Net.Cache

可定義類型和列舉，這些類型和列舉是用來定義使用 System.Net.WebRequest 和 System.Net.HttpWebRequest 類別所取得之資源的快取原則。

System.Net.Configuration

應用程式用來以程式設計方式存取及更新 System.Net 命名空間組態設定的類別。

System.Net.Http

可為現代 HTTP 應用程式提供程式設計介面的類別。

System.Net.Http.Headers

為 System.Net.Http 命名空間所使用的 HTTP 標頭集合提供支援

System.Net.Mail

用於使用 SMTP 通訊協定撰寫及傳送郵件的類別。

System.Net.Mime

可定義類型，這些類型是用於表示 System.Net.Mail 命名空間中的類別所使用的多用途網際網路郵件交換 (MIME) 標頭。

System.Net.NetworkInformation
以程式設計方式收集網路事件、變更、統計資料和屬性的相關資訊的類別。

System.Net.PeerToPeer
為開發人員提供對等名稱解析通訊協定 (PNRP) 的 Managed 實作。

System.Net.PeerToPeer.Collaboration
為開發人員提供對等協同作業介面的 Managed 實作。

System.Net.Security
用於提供主機之間安全通訊的網路資料流的類別。

System.Net.Sockets
提供 Windows Sockets (Winsock) 介面的 Managed 實作，讓需要協助控制網路存取的開發人員使用。

System.Net.WebSockets
為開發人員提供 WebSocket 介面的 Managed 實作。

System.Uri
提供以物件表示屬性的統一資源識別碼 (URI)，而且可以輕鬆存取 URI 的部分。

System.Security.Authentication.ExtendedProtection
為應用程式提供使用延伸保護進行驗證的支援。

System.Security.Authentication.ExtendedProtection.Configuration
為應用程式提供使用延伸保護設定驗證組態的支援。

另請參閱

- .NET Framework 的傳輸層安全性 (TLS) 最佳做法
- 網路程式設計「如何」主題
- 網路程式設計範例
- HttpClient 範例

網路程式設計 HOW TO 主題

2020/3/20 • • Edit Online

下列清單包含網路程式設計概念文件中找到之「如何」主題的連結。

請求資料:

- 如何:要求網頁並擷取結果當作資料流
- 如何:使用 WebRequest 類別要求資料
- 如何:使用 WebRequest 類別傳送資料
- 如何:擷取符合 WebRequest 的通訊協定特定 WebResponse

可插式和應用程式通訊協定:

- 如何:使用 WebRequest 註冊自訂通訊協定
- 如何:轉換 WebRequest 類型以存取通訊協定特定屬性
- 何:存取 HTTP 特定屬性
- 如何:將使用者資訊指派給群組連線
- 如何:建立通訊端
- 如何:透過 FTP 下載檔案
- 如何:使用 FTP 上傳檔
- 如何:使用 FTP 列出目錄內容

互聯網協定版本 6:

- 如何:修改電腦設定檔案以啟用 IPv6 支援

網路追蹤:

- 如何:設定網路追蹤

設定快取:

- 如何:為應用程式設定以位置為基礎的快取原則
- 如何:為應用程式設定以時間為基礎的預設快取原則
- 如何:自訂以時間為基礎的快取原則
- 如何:設定要求的快取原則

使用代理:

- 如何:啟用 WebRequest 以使用 Proxy 與網際網路通訊
- 如何:覆寫全域 Proxy 的選取範圍

網路資訊:

- 如何:偵測網路可用性和位址變更

- 如何：取得介面和通訊協定資訊
- 如何：Ping 主機

另請參閱

- .NET 框架中的網路程式設計
- 網路程式設計範例
- MSDN Code Gallery 上的 .NET 網路範例

可插式通訊協定簡介

2020/3/20 • • Edit Online

Microsoft .NET Framework 提供有層次、可擴充和網際網路服務的 Managed 實作，可以迅速而簡易地整合到您的應用程式。System.Net 和 System.Net.Sockets 命名空間中的網際網路存取類別，可用來實作 Web 架構和以網際網路為基礎的應用程式。

網際網路應用程式

網際網路應用程式可以概分為兩類：要求資訊的用戶端應用程式，和回應用戶端資訊要求的伺服器應用程式。傳統的網際網路用戶端/伺服器應用程式是全球資訊網，使用者使用瀏覽器在此存取儲存在全球網頁伺服器上的文件和其他資料。

應用程式並不限於其中一個角色。例如，熟悉的中介層應用程式伺服器回應用戶端要求的方法是向另一部伺服器要求資料，在此情況下，它同時擔任伺服器和用戶端。

用戶端應用程式透過識別要求的網際網路資源以及用於要求和回應的通訊協定，提出要求。必要時，用戶端也會提供完成要求所需的任何其他資料，例如 Proxy 位置或驗證資訊 (使用者名稱、密碼等等)。一旦形成要求，就可將要求傳送到伺服器。

識別資源

.NET Framework 使用統一資源識別碼 (URI)，識別所要求的網際網路資源與通訊協定。URI 包含至少三個片段，可能四個：配置識別碼，識別要求和回應的通訊協定；伺服器識別碼，由網域名稱系統 (DNS) 主機名稱，或在網際網路上可唯一識別伺服器的 TCP 位址；路徑識別碼，找出伺服器上要求的資訊；以及選擇性查詢字串，將資訊從用戶端傳遞至伺服器。例如，URI `http://www.contoso.com/whatsnew.aspx?date=today` 的構成為配置識別碼 `http`、伺服器識別碼 `www.contoso.com`、路徑 `/whatsnew.aspx` 和查詢字串 `?date=today`。

伺服器收到要求並處理回應之後，它會將回應傳回到用戶端應用程式。此回應包含補充資訊，例如內容類型 (例如，未經處理文字或 XML 資料)。

.NET Framework 中的要求和回應

.NET Framework 使用特定的類別提供透過要求/回應模型存取網際網路資源的三段必要資訊：Uri 類別，包含您所尋找之網際網路資源的 URI；WebRequest 類別，包含資源的要求；以及 WebResponse 類別，提供連入回應的容器。

用戶端應用程式藉由將網路資源的 URI 傳遞到 Create 方法，建立 `WebRequest` 執行個體。這個靜態方法會為特定的通訊協定建立 `WebRequest`，例如 HTTP。傳回的 `WebRequest` 可讓您存取屬性，同時控制伺服器的要求和提出要求時傳送的資料流存取。`WebRequest` 上的 GetResponse 方法會將要求從用戶端應用程式傳送至在 URI 中找到的伺服器。如果回應可能延遲，則可在 **WebRequest** 上使用 BeginGetResponse 方法以非同步方式提出要求，稍後使用 EndGetResponse 方法傳回回應。

GetResponse 和 **EndGetResponse** 方法傳回的 **WebResponse**，可讓您存取伺服器傳回的資料。因為此資料是 GetResponseStream 方法以資料流的形式提供給提出要求的應用程式，所以可用於應用程式中任何使用資料流的位置。

WebRequest 和 **WebResponse** 類別是可插式通訊協定的基礎：這是一項網路服務實作，可讓您開發應用程式使用網際網路資源，而不用擔心每項資源所使用的通訊協定特定詳細資料。**WebRequest** 的子代類別是向 **WebRequest** 類別註冊，以管理建立網際網路資源實際連線的詳細資料。

例如，HttpWebRequest 類別管理使用 HTTP 連線到網際網路資源的詳細資料。根據預設，當 **WebRequest.Create** 方法遇到開頭為 "http:" 或 "https:" (HTTP 與安全的 HTTP 通訊協定識別碼) 的 URI 時，傳回的 **WebRequest** 可依現

況使用，或將類型轉換成 HttpWebRequest 以存取通訊協定特有的屬性。在大部分情況下，WebRequest 會提供提出要求的所有必要資訊。

可以要求/回應交易表示的任何通訊協定都可用於 WebRequest。您也可以從 WebRequest 和 WebResponse 衍生通訊協定特定類別，然後註冊它們供應用程式搭配靜態 WebRequest.RegisterPrefix 方法使用。

當用於網際網路要求的用戶端驗證為必要時，WebRequest 的 Credentials 屬性會提供所需的認證。這些認證可以是基本 HTTP 或摘要式驗證的簡單名稱/密碼組，或者是 NTLM 或 Kerberos 驗證的名稱/密碼/網域集。一組認證可以儲存在 NetworkCredential 執行個體中，或多組認證同時儲存在 CredentialCache 執行個體中。CredentialCache 使用要求的 URI 和伺服器支援的驗證配置，以判斷要傳送到伺服器的認證。

使用 WebClient 的簡單要求

凡是需要提出網際網路資源簡單要求的應用程式，WebClient 類別都會提供常見方法，在網際網路伺服器上傳資料或下載資料。WebClient 依賴 WebRequest 類別提供對網際網路資源的存取；因此，WebClient 類別可以使用任何已註冊的可插式通訊協定。

對於無法使用請求/回應模型的應用程式，或者對於需要在網路上偵聽和發送請求的應用程式，System.Net.Sockets命名空間提供TcpClient和TcpListenerUdpClient類。這些類別會處理使用不同的傳輸通訊協定建立連線的詳細資料，並將應用程式的網路連線公開為資料流。

熟悉 Windows Sockets 介面的開發人員，或需要在通訊端層級由程式設計提供控制項的開發人員，會發現 System.Net.Sockets 類別符合他們的需要。System.Net.Sockets 類別是在 System.Net 類別內從 Managed 程式碼到原生程式碼的轉換點。在大部分情況下，System.Net.Sockets 類別會將資料封送處理到其 Windows 32 位元的對應項目，以及處理任何必要的安全性檢查。

另請參閱

- 可插式通訊協定程式設計
- .NET 框架中的網路程式設計
- 網路程式設計範例

要求資料

2020/3/20 • • Edit Online

開發在現今網際網路分散式作業環境中執行的應用程式，需要從所有類型的資源中擷取資料的有效且易用的方法。可插式通訊協定可讓您開發應用程式，而應用程式使用單一介面來擷取多個網際網路通訊協定中的資料。

從網際網路伺服器上傳和下載資料

針對簡單要求和回應交易，WebClient 類別提供最簡單的模型，將資料上傳至網際網路伺服器，或從中下載資料。WebClient 提供方法來上傳和下載檔案、傳送和接收資料流，並將資料緩衝區傳送至伺服器以及接收回應。WebClient 會使用 WebRequest 和 WebResponse 類別進行實際網際網路資源連線，讓任何註冊的插入式通訊協定可供使用。

需要使用 WebRequest 類別和其子系，從伺服器製作更複雜交易要求資料的用戶端應用程式。WebRequest 會封裝連線至伺服器、傳送要求以及接收回應的詳細資料。WebRequest 是定義一組屬性和方法的抽象類別，而屬性和方法可供使用可插式通訊協定的所有應用程式使用。WebRequest 子系 (例如 HttpWebRequest) 會使用與基礎通訊協定一致的方式，來實作 WebRequest 所定義的屬性和方法。

WebRequest 類別會使用傳遞至其 Create 方法的 URI 值來建立 WebRequest 子系的通訊協定特定執行個體，以判斷要建立的特定衍生類別執行個體。應用程式指出應該使用哪個 WebRequest 子系來處理要求，方法是使用 WebRequest.RegisterPrefix 方法來註冊子系的建構函式。

要求網際網路資源的方式是在 WebRequest 上呼叫 GetResponse 方法。GetResponse 方法會從 WebRequest 的屬性建構通訊協定特定要求，並建立與伺服器的 TCP 或 UDP 通訊端連線，然後傳送要求。針對將資料傳送至伺服器的要求 (例如 HTTP Post 或 FTP Put 要求)，WebRequest.GetRequestStream 方法提供在其中傳送資料的網路資料流。

GetResponse 方法會傳回符合 WebRequest 的通訊協定特定 WebResponse。

WebResponse 類別也是定義屬性和方法的抽象類別，而屬性和方法可供使用可插式通訊協定的所有應用程式使用。WebResponse 子系會針對基礎通訊協定實作這些屬性和方法。例如，HttpWebResponse 類別實作 WebResponse 類別來進行 HTTP。

向 WebResponse.GetResponseStream 方法所傳回資料流中的應用程式呈現伺服器所傳回的資料。您可以像使用任何其他資料流的方式一樣來使用此資料流，如下列範例所示。

```
StreamReader sr =
   new StreamReader(resp.GetResponseStream(), Encoding.ASCII);
```

```
Dim sr As StreamReader
sr = New StreamReader(resp.GetResponseStream(), Encoding.ASCII)
```

另請參閱

- .NET 框架中的網路程式設計
- 如何：要求網頁並擷取結果當作資料流
- 如何：擷取符合 WebRequest 的通訊協定特定 WebResponse

建立網際網路要求

2020/3/20 • • Edit Online

應用程式會透過 WebRequest.Create 方法來建立 WebRequest 執行個體。這是建立衍生自 **WebRequest** 之類別的靜態方法 (視傳遞給它的 URI 配置而定)。

Web、檔案和 FTP 要求

.NET Framework 提供衍生自 **WebRequest** 的 HttpWebRequest 類別，來處理 HTTP 和 HTTPS 要求。在大部分情況下，**WebRequest** 類別會提供您提出要求所需的所有屬性；不過，必要時，您可以將 **WebRequest.Create** 方法所建立的 **WebRequest** 物件轉換為 **HttpWebRequest** 類型，以存取要求的 HTTP 特定屬性。同樣地，**HttpWebResponse** 物件會處理來自 HTTP 和 HTTPS 要求的回應。若要存取 **HttpWebResponse** 物件的 HTTP 特定屬性，您需要將 **WebResponse** 物件轉換為 **HttpWebResponse** 類型。

.NET Framework 也會提供 FileWebRequest 和 FileWebResponse 類別，來處理使用 "file:" URI 配置之資源的要求。同樣地，提供 FtpWebRequest 和 FtpWebResponse 類別，以處理使用 "ftp:" 配置之資源的要求。如果您的要求適用於使用所有這些配置的資源，則可以使用 **WebRequest.Create** 方法，來取得用來提出要求的物件。

若要處理使用其他應用程式層級通訊協定的要求，您需要實作衍生自 **WebRequest** 和 **WebResponse** 的通訊協定特定類別。有關詳細資訊，請參閱**程式設計可插拔協定**。

另請參閱

- 如何：使用 WebRequest 類別要求資料
- 要求資料

如何：要求網頁並擷取結果當做資料流

2020/3/20 • • Edit Online

這個範例示範如何要求網頁並擷取資料流中的結果。

範例

```
var myClient = new WebClient();
Stream response = myClient.OpenRead("https://docs.microsoft.com/dotnet/");
// The stream data is used here.
response.Close();
```

```
Dim myClient As New WebClient()
Dim response As Stream = myClient.OpenRead("https://docs.microsoft.com/dotnet/")
' The stream data is used here.
response.Close()
```

編譯程式碼

這個範例需要：

- System.IO 和 System.Net 命名空間的參考。

另請參閱

- 要求資料

如何：使用 WebRequest 類請求資料

2020/3/20 • • Edit Online

下列程序描述向伺服器要求資源 (例如網頁或檔案) 的步驟。資源必須是以 URI 識別。

向主機伺服器要求資料

1. 使用資源的 URI 來呼叫 WebRequest.Create，以建立 WebRequest 執行個體。例如：

```
WebRequest request = WebRequest.Create("https://docs.microsoft.com");
```

```
Dim request as WebRequest = WebRequest.Create("https://docs.microsoft.com")
```

> **NOTE**
> .NET Framework 針對以 *http:*、*https:*、*ftp:* 和 *file:* 開頭的 URI，提供了衍生自 WebRequest 和 WebResponse 類別的通訊協定專用類別。

如果您需要設定或讀取通訊協定專用屬性，必須將 WebRequest 或 WebResponse 物件轉換為通訊協定專用物件類型。如需詳細資訊，請參閱可插式通訊協定程式設計。

2. 在 `WebRequest` 物件中設定任何需要的屬性值。例如，若要啟用驗證，請將 WebRequest.Credentials 屬性設定為 NetworkCredential 類別的執行個體：

```
request.Credentials = CredentialCache.DefaultCredentials;
```

```
request.Credentials = CredentialCache.DefaultCredentials
```

3. 藉由呼叫 WebRequest.GetResponse，將要求傳送到伺服器。這個方法會傳回包含伺服器回應的物件。傳回的 WebResponse 物件類型取決於要求 URI 的配置。例如：

```
WebResponse response = request.GetResponse();
```

```
Dim response As WebResponse = request.GetResponse()
```

4. 您可以存取 `WebResponse` 物件的屬性，或將其轉換為通訊協定專用執行個體，以讀取通訊協定專用屬性。

例如，若要存取 HttpWebResponse 的 HTTP 專用屬性，請將 `WebResponse` 物件轉換為 `HttpWebResponse` 參考。下列程式碼範例示範如何顯示與回應一起傳送的 HTTP 專用 HttpWebResponse.StatusDescription 屬性：

```
Console.WriteLine (((HttpWebResponse)response).StatusDescription);
```

```
Console.WriteLine(CType(response,HttpWebResponse).StatusDescription)
```

5. 若要取得含有伺服器所傳送之回應資料的資料流，請呼叫 WebResponse.GetResponseStream 方法。例如：

```
Stream dataStream = response.GetResponseStream();
```

```
Dim dataStream As Stream = response.GetResponseStream()
```

6. 從回應物件讀取資料之後，請使用 WebResponse.Close 方法關閉它，或使用 Stream.Close 方法關閉回應資料流。如果不關閉回應物件或資料流，應用程式可能會耗盡伺服器連線，而變得無法處理其他要求。`WebResponse.Close` 方法在關閉回應時會呼叫 `Stream.Close`，因此不需要對回應和資料流物件呼叫 `Close`，但是這麼做也不會造成損害。例如：

```
response.Close();
```

```
response.Close()
```

範例

下列程式碼範例示範如何對網頁伺服器建立要求，並讀取其回應中的資料：

```
using System;
using System.IO;
using System.Net;

namespace Examples.System.Net
{
    public class WebRequestGetExample
    {
        public static void Main()
        {
            // Create a request for the URL.
            WebRequest request = WebRequest.Create(
              "https://docs.microsoft.com");
            // If required by the server, set the credentials.
            request.Credentials = CredentialCache.DefaultCredentials;

            // Get the response.
            WebResponse response = request.GetResponse();
            // Display the status.
            Console.WriteLine(((HttpWebResponse)response).StatusDescription);

            // Get the stream containing content returned by the server.
            // The using block ensures the stream is automatically closed.
            using (Stream dataStream = response.GetResponseStream())
            {
                // Open the stream using a StreamReader for easy access.
                StreamReader reader = new StreamReader(dataStream);
                // Read the content.
                string responseFromServer = reader.ReadToEnd();
                // Display the content.
                Console.WriteLine(responseFromServer);
            }

            // Close the response.
            response.Close();
        }
    }
}
```

```
Imports System.IO
Imports System.Net

Namespace Examples.System.Net
    Public Class WebRequestGetExample
        Public Shared Sub Main()
            ' Create a request for the URL.
            Dim request As WebRequest =
              WebRequest.Create("https://docs.microsoft.com")
            ' If required by the server, set the credentials.
            request.Credentials = CredentialCache.DefaultCredentials

            ' Get the response.
            Dim response As WebResponse = request.GetResponse()
            ' Display the status.
            Console.WriteLine(CType(response, HttpWebResponse).StatusDescription)

            ' Get the stream containing content returned by the server.
            ' The using block ensures the stream is automatically closed.
            Using dataStream As Stream = response.GetResponseStream()
                ' Open the stream using a StreamReader for easy access.
                Dim reader As New StreamReader(dataStream)
                ' Read the content.
                Dim responseFromServer As String = reader.ReadToEnd()
                ' Display the content.
                Console.WriteLine(responseFromServer)
            End Using

            ' Clean up the response.
            response.Close()
        End Sub
    End Class
End Namespace
```

另請參閱

- 創建互聯網請求
- 在網路上使用流
- 透過 Proxy 存取網際網路
- 請求資料
- 如何：使用 WebRequest 類發送資料

如何：使用 WebRequest 類發送資料

2020/3/20 • • Edit Online

下列程序描述將資料傳送到伺服器的步驟。本程序通常用於在網頁上張貼資料。

將資料傳送到主機伺服器

1. 使用接受資料之資源 (例如，指令碼或 ASP.NET 網頁) 的 URI 呼叫 WebRequest.Create，以建立 WebRequest 執行個體。例如：

```
WebRequest request = WebRequest.Create("http://www.contoso.com/PostAccepter.aspx");
```

```
Dim request as WebRequest = WebRequest.Create("http://www.contoso.com/PostAccepter.aspx")
```

> **NOTE**
> .NET Framework 針對以 *http:*、*https:*、*ftp:* 和 *file:* 開頭的 URI，提供了衍生自 WebRequest 和 WebResponse 類別的通訊協定專用類別。

如果您需要設定或讀取通訊協定專用屬性，必須將 WebRequest 或 WebResponse 物件轉換為通訊協定專用物件類型。如需詳細資訊，請參閱可插式通訊協定程式設計。

2. 在 `WebRequest` 物件中設定任何需要的屬性值。例如，若要啟用驗證，請將 WebRequest.Credentials 屬性設定為 NetworkCredential 類別的執行個體：

```
request.Credentials = CredentialCache.DefaultCredentials;
```

```
request.Credentials = CredentialCache.DefaultCredentials
```

3. 指定允許資料與要求一起傳送的通訊協定方法，例如 HTTP `POST` 方法：

```
request.Method = "POST";
```

```
request.Method = "POST"
```

4. 將 ContentLength 屬性設定為您要求中包含的位元組數目。例如：

```
request.ContentLength = byteArray.Length;
```

```
request.ContentLength = byteArray.Length
```

5. 將 ContentType 屬性設定為適當值。例如：

```
request.ContentType = "application/x-www-form-urlencoded";
```

```
request.ContentType = "application/x-www-form-urlencoded"
```

6. 藉由呼叫 GetRequestStream 方法，取得保留要求資料的資料流。例如：

```
Stream dataStream = request.GetRequestStream();
```

```
Dim dataStream As Stream = request.GetRequestStream()
```

7. 將資料寫入 `GetRequestStream` 方法所傳回的 Stream 物件。例如：

```
dataStream.Write(byteArray, 0, byteArray.Length);
```

```
dataStream.Write(byteArray, 0, byteArray.Length)
```

8. 呼叫 Stream.Close 方法來關閉要求資料流。例如：

```
dataStream.Close();
```

```
dataStream.Close()
```

9. 藉由呼叫 WebRequest.GetResponse，將要求傳送到伺服器。這個方法會傳回包含伺服器回應的物件。傳回的 `WebResponse` 物件類型取決於要求 URI 的配置。例如：

```
WebResponse response = request.GetResponse();
```

```
Dim response As WebResponse = request.GetResponse()
```

10. 您可以存取 `WebResponse` 物件的屬性，或將其轉換為通訊協定專用執行個體，以讀取通訊協定專用屬性。

    例如，若要存取 HttpWebResponse 的 HTTP 專用屬性，請將 `WebResponse` 物件轉換為 HttpWebResponse 參考。下列程式碼範例示範如何顯示與回應一起傳送的 HTTP 專用 HttpWebResponse.StatusDescription 屬性：

```
Console.WriteLine(((HttpWebResponse)response).StatusDescription);
```

```
Console.WriteLine(CType(response, HttpWebResponse).StatusDescription)
```

11. 若要取得含有伺服器所傳送之回應資料的資料流，請呼叫 `WebResponse` 物件的 WebResponse.GetResponseStream 方法。例如：

```
Stream dataStream = response.GetResponseStream();
```

```
Dim dataStream As Stream = response.GetResponseStream()
```

12. 從回應物件讀取資料之後，請使用 WebResponse.Close 方法關閉它，或使用 Stream.Close 方法關閉回應資料流。如果不關閉回應或資料流，應用程式可能會耗盡伺服器連線，而變得無法處理其他要求。`WebResponse.Close` 方法在關閉回應時會呼叫 `Stream.Close`，因此不需要對回應和資料流物件呼叫 `Close`，但是這麼做也不會造成損害。例如：

```
response.Close();
```

```
response.Close()
```

範例

下列範例會示範如何將資料傳送到網頁伺服器，並讀取其回應中的資料：

```
using System;
using System.IO;
using System.Net;
using System.Text;

namespace Examples.System.Net
{
    public class WebRequestPostExample
    {
        public static void Main()
        {
            // Create a request using a URL that can receive a post.
            WebRequest request = WebRequest.Create("http://www.contoso.com/PostAccepter.aspx ");
            // Set the Method property of the request to POST.
            request.Method = "POST";

            // Create POST data and convert it to a byte array.
            string postData = "This is a test that posts this string to a Web server.";
            byte[] byteArray = Encoding.UTF8.GetBytes(postData);

            // Set the ContentType property of the WebRequest.
            request.ContentType = "application/x-www-form-urlencoded";
            // Set the ContentLength property of the WebRequest.
            request.ContentLength = byteArray.Length;

            // Get the request stream.
            Stream dataStream = request.GetRequestStream();
            // Write the data to the request stream.
            dataStream.Write(byteArray, 0, byteArray.Length);
            // Close the Stream object.
            dataStream.Close();

            // Get the response.
            WebResponse response = request.GetResponse();
            // Display the status.
            Console.WriteLine(((HttpWebResponse)response).StatusDescription);

            // Get the stream containing content returned by the server.
            // The using block ensures the stream is automatically closed.
            using (dataStream = response.GetResponseStream())
            {
                // Open the stream using a StreamReader for easy access.
                StreamReader reader = new StreamReader(dataStream);
                // Read the content.
                string responseFromServer = reader.ReadToEnd();
                // Display the content.
                Console.WriteLine(responseFromServer);
            }

            // Close the response.
            response.Close();
        }
    }
}
```

```
Imports System.IO
Imports System.Net
Imports System.Text

Namespace Examples.System.Net
    Public Class WebRequestPostExample
        Public Shared Sub Main()
            ' Create a request using a URL that can receive a post.
            Dim request As WebRequest = WebRequest.Create("http://www.contoso.com/PostAccepter.aspx ")
            ' Set the Method property of the request to POST.
            request.Method = "POST"

            ' Create POST data and convert it to a byte array.
            Dim postData As String = "This is a test that posts this string to a Web server."
            Dim byteArray As Byte() = Encoding.UTF8.GetBytes(postData)

            ' Set the ContentType property of the WebRequest.
            request.ContentType = "application/x-www-form-urlencoded"
            ' Set the ContentLength property of the WebRequest.
            request.ContentLength = byteArray.Length

            ' Get the request stream.
            Dim dataStream As Stream = request.GetRequestStream()
            ' Write the data to the request stream.
            dataStream.Write(byteArray, 0, byteArray.Length)
            ' Close the Stream object.
            dataStream.Close()

            ' Get the response.
            Dim response As WebResponse = request.GetResponse()
            ' Display the status.
            Console.WriteLine(CType(response, HttpWebResponse).StatusDescription)

            ' Get the stream containing content returned by the server.
            ' The using block ensures the stream is automatically closed.
            Using dataStream1 As Stream = response.GetResponseStream()
                ' Open the stream using a StreamReader for easy access.
                Dim reader As New StreamReader(dataStream1)
                ' Read the content.
                Dim responseFromServer As String = reader.ReadToEnd()
                ' Display the content.
                Console.WriteLine(responseFromServer)
            End Using

            ' Clean up the response.
            response.Close()
        End Sub
    End Class
End Namespace
```

另請參閱

- 創建互聯網請求
- 在網路上使用流
- 透過 Proxy 存取網際網路
- 請求資料
- 如何：使用 WebRequest 類請求資料

如何：擷取符合 WebRequest 的通訊協定特定 WebResponse

2020/3/20 • • Edit Online

這個範例示範如何擷取符合 WebRequest 的通訊協定特定 WebResponse。

範例

```
WebRequest req = WebRequest.Create("http://www.contoso.com/");
WebResponse resp = req.GetResponse();
```

```
Dim req As WebRequest = WebRequest.Create("http://www.contoso.com")
Dim resp As WebResponse = req.GetResponse()
```

編譯程式碼

這個範例需要：

- 對 System.Net 命名空間的參考。

另請參閱

- 要求資料

在網路上使用資料流

2020/3/20 • • Edit Online

網路資源在 .NET Framework 中是以資料流的形式呈現。因為對資料流沒有任何特殊待遇，.NET Framework 提供下列功能：

- 以一般方式傳送和接收 Web 資料。不論實際的檔案內容為何 (HTML、XML 或其他格式)，應用程式都會使用 Stream.Write 和 Stream.Read 來傳送和接收資料。
- 與整個 .NET Framework 資料流的相容性。.NET Framework 中無處不使用資料流，其處理資料流的基礎結構健全。例如，您可將讀取 FileStream 中 XML 資料的應用程式，修改為讀取 NetworkStream 中的資料，而非僅變更將資料流初始化的幾行程式碼。**NetworkStream** 類別與其他資料流最大的不同處，在於 **NetworkStream** 是不可搜尋的，CanSeek 屬性一律傳回 **false**而 Seek 和 Position 方法則擲回 NotSupportedException。
- 資料送達時的處理。來自網路的資料以資料流方式到達時，應用程式可立即存取該資料，而不必等到整個資料集下載完成後才存取。

System.Net.Sockets 命名空間包含 **NetworkStream** 類別，其可特別實作用於網路資源的 Stream 類別。System.Net.Sockets 命名空間中的類別會使用 **NetworkStream** 類別來代表資料流。

若要使用傳回的資料流將資料傳送到網路上，請在 WebRequest 上呼叫 GetRequestStream。**Web 請求**將向伺服器發送請求標頭;然後，您可以通過調用 返回的流上的BeginWrite、EndWrite或Write方法將資料發送到網路資源。有些通訊協定 (例如 HTTP) 可能會要求您先設定特定通訊協定屬性，再傳送資料。下列程式碼範例示範如何設定用於傳送資料的特定 HTTP 屬性。它假設變數 `sendData` 包含要傳送的資料，而變數 `sendLength` 則是要傳送資料的位元組數。

```
HttpWebRequest request =
   (HttpWebRequest) WebRequest.Create("http://www.contoso.com/");
request.Method = "POST";
request.ContentLength = sendLength;
try
{
   Stream sendStream = request.GetRequestStream();
   sendStream.Write(sendData,0,sendLength);
   sendStream.Close();
}
catch
{
   // Handle errors . . .
}
```

```
Dim request As HttpWebRequest = _
   CType(WebRequest.Create("http://www.contoso.com/"), HttpWebRequest)
request.Method = "POST"
request.ContentLength = sendLength
Try
   Dim sendStream As Stream = request.GetRequestStream()
   sendStream.Write(sendData, 0, sendLength)
   sendStream.Close()
Catch
   ' Handle errors . . .
End Try
```

若要從網路接收資料，請在 WebResponse 上呼叫 GetResponseStream。然後，您即可在傳回的資料流上呼叫

BeginRead、EndRead 或 Read 方法，以讀取網路資源的資料。

使用網路資源的資料流時，請牢記下列要點：

- 因為 NetworkStream 類別無法在資料流中變更位置，所以 CanSeek 屬性一律傳回 false。Seek 和 Position 方法會擲回 NotSupportedException。
- 使用 WebRequest 和 WebResponse 時，呼叫 GetResponseStream 所建立的資料流執行個體為唯讀，而呼叫 GetRequestStream 所建立的資料流執行個體則為唯寫。
- 使用 StreamReader 類別可讓編碼工作變得更輕鬆。下列程式碼範例會使用 StreamReader 從 WebResponse 讀取以 ASCII 編碼的資料流 (範例中未示範如何建立要求)。
- 如果沒有可用的網路資源，即可能會封鎖對 GetResponse 的呼叫。您應考慮透過 BeginGetResponse 和 EndGetResponse 方法使用非同步要求。
- 完成建立伺服器連線時，可能會封鎖對 GetRequestStream 的呼叫。您應考慮透過 BeginGetRequestStream 和 EndGetRequestStream 方法，對資料流使用非同步要求。

```
// Create a response object.
WebResponse response = request.GetResponse();
// Get a readable stream from the server.
StreamReader sr =
   new StreamReader(response.GetResponseStream(), Encoding.ASCII);
// Use the stream. Remember when you are through with the stream to close it.
sr.Close();
```

```
' Create a response object.
Dim response As WebResponse = request.GetResponse()
' Get a readable stream from the server.
Dim sr As _
   New StreamReader(response.GetResponseStream(), Encoding.ASCII)
' Use the stream. Remember when you are through with the stream to close it.
sr.Close()
```

另請參閱

- 如何：使用 WebRequest 類別要求資料
- 要求資料

進行非同步要求

2020/3/20 • • Edit Online

System.Net 類別會使用 .NET Framework 的標準非同步程式設計模型，非同步存取網際網路資源。WebRequest 類別的 BeginGetResponse 和 EndGetResponse 方法會啟動和完成網際網路資源的非同步要求。

> **NOTE**
> 在非同步回呼方法中使用同步呼叫可能會導致嚴重效能降低。使用 WebRequest 和其子系所進行的內部要求，必須使用 Stream.BeginRead 來讀取 WebResponse.GetResponseStream 方法所傳回的資料流。

下列範例程式碼示範如何搭配使用非同步呼叫與 WebRequest 類別。此範例是一種主控台程式，可從命令列接受 URI，並要求 URI 上的資源，然後將從網際網路收到的資料列印至主控台。

此程式定義兩個類別供它自己使用：RequestState 類別可跨非同步呼叫傳遞資料，以及 ClientGetAsync 類別可實作網際網路資源非同步要求。

RequestState 類別會跨提供要求的非同步方法呼叫來保留要求的狀態。它所含的 WebRequest 和 Stream 執行個體包含目前資源要求以及基於回應而收到的資料流、所含的緩衝區包含目前從網際網路資源收到的資料，而且所含的 StringBuilder 包含完整回應。向 WebRequest.BeginGetResponse 註冊 AsyncCallback 方法時，會將 RequestState 傳遞為 *state* 參數。

ClientGetAsync 類別實作網際網路資源非同步要求，並將產生的回應寫入主控台中。它包含下列清單所述的方法和屬性。

- `allDone` 屬性包含發出要求完成訊號的 ManualResetEvent 類別執行個體。
- `Main()` 方法會讀取命令列，並開始所指定網際網路資源的要求。它會建立 WebRequest `wreq` 和 RequestState `rs`，並呼叫 BeginGetResponse 開始處理要求，然後呼叫 `allDone.WaitOne()` 方法，讓應用程式不會在回呼完成之前結束。讀取來自網際網路資源的回應之後，`Main()` 會將它寫入至主控台，而且應用程式會結束。
- `showusage()` 方法會寫入主控台上的範例命令列。命令列上未提供任何 URI 時，它是由 `Main()` 所呼叫。
- `RespCallBack()` 方法會實作網際網路要求的非同步回呼方法。它會建立包含網際網路資源回應的 WebResponse 執行個體，並取得回應資料流，然後開始非同步地讀取資料流中的資料。
- `ReadCallBack()` 方法會實作非同步回呼方法來讀取回應資料流。它會將接收自網際網路資源的資料傳送至 RequestState 執行個體的 ResponseData 屬性，接著對回應資料流啟動另一個非同步讀取，直到不再傳回其他資料為止。已讀取所有資料之後，`ReadCallBack()` 會關閉回應資料流，並呼叫 `allDone.Set()` 方法，指出整個回應存在於 ResponseData 中。

  > **NOTE**
  > 請務必關閉所有網路資料流。如果您未關閉每個要求和回應資料流，應用程式就會耗盡與該伺服器的連線，而無法處理其他要求。

```
using System;
using System.Net;
using System.Threading;
using System.Text;
using System.IO;
```

```
// The RequestState class passes data across async calls.
public class RequestState
{
   const int BufferSize = 1024;
   public StringBuilder RequestData;
   public byte[] BufferRead;
   public WebRequest Request;
   public Stream ResponseStream;
   // Create Decoder for appropriate enconding type.
   public Decoder StreamDecode = Encoding.UTF8.GetDecoder();

   public RequestState()
   {
      BufferRead = new byte[BufferSize];
      RequestData = new StringBuilder(String.Empty);
      Request = null;
      ResponseStream = null;
   }
}

// ClientGetAsync issues the async request.
class ClientGetAsync
{
   public static ManualResetEvent allDone = new ManualResetEvent(false);
   const int BUFFER_SIZE = 1024;

   public static void Main(string[] args)
   {
      if (args.Length < 1)
      {
         showusage();
         return;
      }

      // Get the URI from the command line.
      Uri httpSite = new Uri(args[0]);

      // Create the request object.
      WebRequest wreq = WebRequest.Create(httpSite);

      // Create the state object.
      RequestState rs = new RequestState();

      // Put the request into the state object so it can be passed around.
      rs.Request = wreq;

      // Issue the async request.
      IAsyncResult r = (IAsyncResult) wreq.BeginGetResponse(
         new AsyncCallback(RespCallback), rs);

      // Wait until the ManualResetEvent is set so that the application
      // does not exit until after the callback is called.
      allDone.WaitOne();

      Console.WriteLine(rs.RequestData.ToString());
   }

   public static void showusage() {
      Console.WriteLine("Attempts to GET a URL");
      Console.WriteLine("\r\nUsage:");
      Console.WriteLine("   ClientGetAsync URL");
      Console.WriteLine("   Example:");
      Console.WriteLine("      ClientGetAsync http://www.contoso.com/");
   }

   private static void RespCallback(IAsyncResult ar)
   {
      // Get the RequestState object from the async result.
```

```
      RequestState rs = (RequestState) ar.AsyncState;

      // Get the WebRequest from RequestState.
      WebRequest req = rs.Request;

      // Call EndGetResponse, which produces the WebResponse object
      //  that came from the request issued above.
      WebResponse resp = req.EndGetResponse(ar);

      //  Start reading data from the response stream.
      Stream ResponseStream = resp.GetResponseStream();

      // Store the response stream in RequestState to read
      // the stream asynchronously.
      rs.ResponseStream = ResponseStream;

      //  Pass rs.BufferRead to BeginRead. Read data into rs.BufferRead
      IAsyncResult iarRead = ResponseStream.BeginRead(rs.BufferRead, 0,
         BUFFER_SIZE, new AsyncCallback(ReadCallBack), rs);
   }

   private static void ReadCallBack(IAsyncResult asyncResult)
   {
      // Get the RequestState object from AsyncResult.
      RequestState rs = (RequestState)asyncResult.AsyncState;

      // Retrieve the ResponseStream that was set in RespCallback.
      Stream responseStream = rs.ResponseStream;

      // Read rs.BufferRead to verify that it contains data.
      int read = responseStream.EndRead( asyncResult );
      if (read > 0)
      {
         // Prepare a Char array buffer for converting to Unicode.
         Char[] charBuffer = new Char[BUFFER_SIZE];

         // Convert byte stream to Char array and then to String.
         // len contains the number of characters converted to Unicode.
      int len =
         rs.StreamDecode.GetChars(rs.BufferRead, 0, read, charBuffer, 0);

         String str = new String(charBuffer, 0, len);

         // Append the recently read data to the RequestData stringbuilder
         // object contained in RequestState.
         rs.RequestData.Append(
            Encoding.ASCII.GetString(rs.BufferRead, 0, read));

         // Continue reading data until
         // responseStream.EndRead returns –1.
         IAsyncResult ar = responseStream.BeginRead(
            rs.BufferRead, 0, BUFFER_SIZE,
            new AsyncCallback(ReadCallBack), rs);
      }
      else
      {
         if(rs.RequestData.Length>0)
         {
            //  Display data to the console.
            string strContent;
            strContent = rs.RequestData.ToString();
         }
         // Close down the response stream.
         responseStream.Close();
         // Set the ManualResetEvent so the main thread can exit.
         allDone.Set();
      }
      return;
   }
```

```
}
```

```
Imports System
Imports System.Net
Imports System.Threading
Imports System.Text
Imports System.IO

' The RequestState class passes data across async calls.
Public Class RequestState

      Public RequestData As New StringBuilder("")
      Public BufferRead(1024) As Byte
      Public Request As HttpWebRequest
      Public ResponseStream As Stream
      ' Create Decoder for appropriate encoding type.
      Public StreamDecode As Decoder = Encoding.UTF8.GetDecoder()

      Public Sub New
         Request = Nothing
         ResponseStream = Nothing
      End Sub
End Class

' ClientGetAsync issues the async request.
Class ClientGetAsync
   Shared allDone As New ManualResetEvent(False)
      Const BUFFER_SIZE As Integer = 1024

    Shared Sub Main()
        Dim Args As String() = Environment.GetCommandLineArgs()

        If Args.Length < 2 Then
            ShowUsage()
            Return
        End If

      ' Get the URI from the command line.
        Dim HttpSite As Uri = New Uri(Args(1))

      ' Create the request object.
        Dim wreq As HttpWebRequest = _
           CType(WebRequest.Create(HttpSite), HttpWebRequest)

      ' Create the state object.
      Dim rs As RequestState = New RequestState()

      ' Put the request into the state so it can be passed around.
      rs.Request = wreq

      ' Issue the async request.
        Dim r As IAsyncResult = _
           CType(wreq.BeginGetResponse( _
           New AsyncCallback(AddressOf RespCallback), rs), IAsyncResult)

      ' Wait until the ManualResetEvent is set so that the application
      ' does not exit until after the callback is called.
        allDone.WaitOne()
    End Sub

    Shared Sub ShowUsage()
        Console.WriteLine("Attempts to GET a URI")
        Console.WriteLine()
        Console.WriteLine("Usage:")
        Console.WriteLine("ClientGetAsync URI")
        Console.WriteLine("Examples:")
        Console.WriteLine("ClientGetAsync http://www.contoso.com/")
```

```
    End Sub

    Shared Sub RespCallback(ar As IAsyncResult)
       ' Get the RequestState object from the async result.
       Dim rs As RequestState = CType(ar.AsyncState, RequestState)

       ' Get the HttpWebRequest from RequestState.
       Dim req As HttpWebRequest= rs.Request

       ' Call EndGetResponse, which returns the HttpWebResponse object
       ' that came from the request issued above.
       Dim resp As HttpWebResponse = _
           CType(req.EndGetResponse(ar), HttpWebResponse)

       ' Start reading data from the response stream.
       Dim ResponseStream As Stream = resp.GetResponseStream()

       ' Store the reponse stream in RequestState to read
       ' the stream asynchronously.
       rs.ResponseStream = ResponseStream

       ' Pass rs.BufferRead to BeginRead. Read data into rs.BufferRead.
       Dim iarRead As IAsyncResult = _
          ResponseStream.BeginRead(rs.BufferRead, 0, BUFFER_SIZE, _
          New AsyncCallback(AddressOf ReadCallBack), rs)
    End Sub

   Shared Sub ReadCallBack(asyncResult As IAsyncResult)
      ' Get the RequestState object from the AsyncResult.
      Dim rs As RequestState = CType(asyncResult.AsyncState, RequestState)

      ' Retrieve the ResponseStream that was set in RespCallback.
      Dim responseStream As Stream = rs.ResponseStream

      ' Read rs.BufferRead to verify that it contains data.
      Dim read As Integer = responseStream.EndRead( asyncResult )
      If read > 0 Then
         ' Prepare a Char array buffer for converting to Unicode.
         Dim charBuffer(1024) As Char

         ' Convert byte stream to Char array and then String.
         ' len contains the number of characters converted to Unicode.
         Dim len As Integer = _
           rs.StreamDecode.GetChars(rs.BufferRead, 0, read, charBuffer, 0)
         Dim str As String = new String(charBuffer, 0, len)

         ' Append the recently read data to the RequestData stringbuilder
         ' object contained in RequestState.
         rs.RequestData.Append( _
            Encoding.ASCII.GetString(rs.BufferRead, 0, read))

         ' Continue reading data until responseStream.EndRead
         ' returns –1.
         Dim ar As IAsyncResult = _
            responseStream.BeginRead(rs.BufferRead, 0, BUFFER_SIZE, _
            New AsyncCallback(AddressOf ReadCallBack), rs)
      Else
          If rs.RequestData.Length > 1 Then
            ' Display data to the console.
            Dim strContent As String
            strContent = rs.RequestData.ToString()
            Console.WriteLine(strContent)
         End If

         ' Close down the response stream.
         responseStream.Close()

         ' Set the ManualResetEvent so the main thread can exit.
         allDone.Set()
```

```
        End If

        Return
    End Sub
End Class
```

另請參閱

- 要求資料

處理錯誤

2020/3/20 • • Edit Online

WebRequest 和 WebResponse 類別會擲回兩個系統例外狀況 (例如 ArgumentException) 和 Web 特定例外狀況 (這些是 GetResponse 方法所擲回的 WebException)。

每個 WebException 包含一個 Status 屬性，其中包含來自 WebExceptionStatus 列舉的值。您可以檢查 Status 屬性來判斷發生的錯誤，並採取適當的步驟來解決這個錯誤。

下表描述 Status 屬性的可能值。

"	"
ConnectFailure	無法在傳輸層級連絡遠端服務。
ConnectionClosed	連線過早關閉。
KeepAliveFailure	伺服器已關閉使用保持連線標頭集合建立的連線。
NameResolutionFailure	名稱服務無法解析主機名稱。
ProtocolError	從伺服器收到的回應已完成，但是指出通訊協定層級發生錯誤。
ReceiveFailure	未從遠端伺服器收到完整的回應。
RequestCanceled	已取消要求。
SecureChannelFailure	安全通道連結發生錯誤。
SendFailure	無法將完整的要求傳送到遠端伺服器。
ServerProtocolViolation	伺服器回應不是有效的 HTTP 回應。
Success	沒有遇到任何錯誤。
逾時	在針對要求所設定的逾時內未收到任何回應。
TrustFailure	無法驗證伺服器憑證。
MessageLengthLimitExceeded	從伺服器傳送要求或接收回應時，收到超過指定限制的訊息。
Pending	內部非同步要求正在擱置中。
PipelineFailure	此值支援 .NET Framework 基礎結構，但不能直接用於您的程式碼中。
ProxyNameResolutionFailure	名稱解析服務無法解析 Proxy 主機名稱。
UnknownError	發生未知類型的例外狀況。

當 Status 屬性是 WebExceptionStatus.ProtocolError 時，即可使用包含伺服器回應的 WebResponse。您可以檢查這個回應，以判斷實際的通訊協定錯誤來源。

下列範例示範如何快取 WebException。

```
try
{
    // Create a request instance.
    WebRequest myRequest =
    WebRequest.Create("http://www.contoso.com");
    // Get the response.
    WebResponse myResponse = myRequest.GetResponse();
    //Get a readable stream from the server.
    Stream sr = myResponse.GetResponseStream();

    //Read from the stream and write any data to the console.
    bytesread = sr.Read( myBuffer, 0, length);
    while( bytesread > 0 )
    {
        for (int i=0; i<bytesread; i++) {
            Console.Write( "{0}", myBuffer[i]);
        }
        Console.WriteLine();
        bytesread = sr.Read( myBuffer, 0, length);
    }
    sr.Close();
    myResponse.Close();
}
catch (WebException webExcp)
{
    // If you reach this point, an exception has been caught.
    Console.WriteLine("A WebException has been caught.");
    // Write out the WebException message.
    Console.WriteLine(webExcp.ToString());
    // Get the WebException status code.
    WebExceptionStatus status =  webExcp.Status;
    // If status is WebExceptionStatus.ProtocolError,
    //   there has been a protocol error and a WebResponse
    //   should exist. Display the protocol error.
    if (status == WebExceptionStatus.ProtocolError) {
        Console.Write("The server returned protocol error ");
        // Get HttpWebResponse so that you can check the HTTP status code.
        HttpWebResponse httpResponse = (HttpWebResponse)webExcp.Response;
        Console.WriteLine((int)httpResponse.StatusCode + " - "
           + httpResponse.StatusCode);
    }
}
catch (Exception e)
{
    // Code to catch other exceptions goes here.
}
```

```
Try
    ' Create a request instance.
    Dim myRequest As WebRequest = WebRequest.Create("http://www.contoso.com")
    ' Get the response.
    Dim myResponse As WebResponse = myRequest.GetResponse()
    'Get a readable stream from the server.
    Dim sr As Stream = myResponse.GetResponseStream()

    Dim i As Integer
    'Read from the stream and write any data to the console.
    bytesread = sr.Read(myBuffer, 0, length)
    While bytesread > 0
        For i = 0 To bytesread - 1
            Console.Write("{0}", myBuffer(i))
        Next i
        Console.WriteLine()
        bytesread = sr.Read(myBuffer, 0, length)
    End While
    sr.Close()
    myResponse.Close()
Catch webExcp As WebException
    ' If you reach this point, an exception has been caught.
    Console.WriteLine("A WebException has been caught.")
    ' Write out the WebException message.
    Console.WriteLine(webExcp.ToString())
    ' Get the WebException status code.
    Dim status As WebExceptionStatus = webExcp.Status
    ' If status is WebExceptionStatus.ProtocolError,
    '   there has been a protocol error and a WebResponse
    '   should exist. Display the protocol error.
    If status = WebExceptionStatus.ProtocolError Then
        Console.Write("The server returned protocol error ")
        ' Get HttpWebResponse so that you can check the HTTP status code.
        Dim httpResponse As HttpWebResponse = _
           CType(webExcp.Response, HttpWebResponse)
        Console.WriteLine(CInt(httpResponse.StatusCode).ToString() & _
           " - " & httpResponse.StatusCode.ToString())
    End If
Catch e As Exception
    ' Code to catch other exceptions goes here.
End Try
```

當 Windows 通訊端發生錯誤時，使用 Socket 類別的應用程式會擲回 SocketException。TcpClient、TcpListener 和 UdpClient 類別都建置在 Socket 類別之上，因此也會擲回 SocketExceptions。

擲回 SocketException 時，SocketException 類別會將 ErrorCode 屬性設定為最後發生的作業系統通訊端錯誤。如需通訊端錯誤碼的詳細資訊，請參閱 MSDN 中的 Winsock 2.0 API 錯誤碼文件。

另請參閱

- 在 .NET 中處理和擲回例外狀況
- 要求資料

可插式通訊協定程式設計

2020/3/20 • • Edit Online

抽象 WebRequest 和 WebResponse 類別提供可插式通訊協定的基底。透過從 WebRequest 和 WebResponse 衍生通訊協定特定類別，應用程式可以要求來自網際網路資源的資料，並讀取回應，而不需要指定所要使用的通訊協定。

建立通訊協定特定 WebRequest 之前，您必須先註冊 Create 方法。使用 WebRequest 的靜態 RegisterPrefix(String, IWebRequestCreate) 方法來註冊 WebRequest 子系，以處理對特定網際網路配置、配置和伺服器或者配置、伺服器和路徑的一組要求。

在大部分情況下，您將可以使用 WebRequest 類別的方法和屬性來傳送和接收資料。不過，如果您需要存取通訊協定特定屬性，則可以將 WebRequest 類型轉換為特定衍生類別執行個體。

若要善用可插式通訊協定，WebRequest 子系必須提供不需要設定通訊協定特定屬性的預設要求和回應交易。例如，實作 WebRequest 類別以進行 HTTP 的 HttpWebRequest 類別預設會提供 `GET` 要求，並傳回包含從網頁伺服器傳回之資料流的 HttpWebResponse。

另請參閱

- 衍生自 WebRequest
- 衍生自 WebResponse
- .NET 框架中的網路程式設計
- 如何：轉換 WebRequest 類型以存取通訊協定特定屬性

如何：使用 WebRequest 註冊自訂通訊協定

2020/3/20 • • Edit Online

這個範例示範如何註冊在其他位置定義的通訊協定特定類別。在此範例中，`CustomWebRequestCreator` 是使用者實作的物件，其可實作 `CustomWebRequest` 物件傳回的 **Create**方法。此程式碼範例假設您已撰寫實作自訂通訊協定的 `CustomWebRequest` 程式碼。

範例

```
WebRequest.RegisterPrefix("custom", new CustomWebRequestCreator());
WebRequest req = WebRequest.Create("custom://customHost.contoso.com/");
```

```
WebRequest.RegisterPrefix("custom", New CustomWebRequestCreator())
Dim req As WebRequest = WebRequest.Create("custom://customHost.contoso.com/")
```

編譯程式碼

這個範例需要：

對 System.Net 命名空間的參考。

另請參閱

- 可插式通訊協定程式設計

如何：轉換 WebRequest 類型以存取通訊協定特定屬性

2020/3/20 • • Edit Online

這個範例示範如何轉換 WebRequest 類型以存取通訊協定特定屬性。

範例

```
HttpWebRequest httpreq =
   (HttpWebRequest) WebRequest.Create("http://www.contoso.com/");
```

```
Dim httpreq As HttpWebRequest = _
   CType(WebRequest.Create("http://www.contoso.com/"), HttpWebRequest)
```

另請參閱

- 可插式通訊協定程式設計

衍生自 WebRequest

2020/3/20 • • Edit Online

WebRequest 類別是抽象的基底類別，提供基本的方法和屬性以建立特定通訊協定要求，其符合 .NET Framework 插入式通訊協定模型的處理常式。使用 WebRequest 類別的應用程式可以使用任何支援的通訊協定要求資料，不需要指定使用的通訊協定。

為使特定通訊協定類別用為插入式通訊協定，必須符合兩個準則：類別必須實作 IWebRequestCreate 介面，且必須使用 WebRequest.RegisterPrefix 方法登錄。類別必須覆寫 WebRequest 所有的抽象方法和屬性，才能提供插入式介面。

WebRequest 執行個體僅供單次使用，如果您想要提出另一項要求，請建立新的 WebRequest。WebRequest 支援 ISerializable 介面，可讓開發人員序列化範本 WebRequest，然後重新建構其他要求的範本。

IWebRequest 建立方法

Create 方法負責初始化特定通訊協定類別的新執行個體。建立新的 WebRequest 時，WebRequest.Create 方法會比對要求的 URI 和以 RegisterPrefix 方法登錄的 URI 前置詞。正確特定通訊協定子代的 Create 方法，必須傳回能夠執行通訊協定標準要求/回應交易的子代初始化執行個體，不需要修改特定通訊協定的任何欄位。

ConnectionGroupName 屬性

ConnectionGroupName 屬性用於命名一組對資源的連線，以便透過單一連線提出多項要求。若要實作連線共用，您必須使用共用和指派連線的特定通訊協定方法。例如，提供的 ServicePointManager 類別會實作 HttpWebRequest 類別的連線共用。ServicePointManager 類別會建立 ServicePoint，為每個連線群組提供特定伺服器的連線。

ContentLength 屬性

ContentLength 屬性指定的資料位元組數目，會在上傳資料時傳送到伺服器。

一般必須設定 Method 屬性，以指出當 ContentLength 屬性設定為大於零的值時，上傳正在進行中。

ContentType 屬性

ContentType 屬性會提供您的通訊協定要求您傳送至伺服器的任何特殊資訊，以識別您要傳送的內容類型。這通常是任何上傳資料的 MIME 內容類型。

Credentials 屬性

Credentials 屬性包含向伺服器驗證要求所需要的資訊。您必須為您的通訊協定實作驗證程序的詳細資料。AuthenticationManager 類別負責驗證要求及提供驗證權杖。提供您通訊協定所用認證的類別，必須實作 ICredentials 介面。

Headers 屬性

Headers 屬性包含與要求建立關聯的中繼資料名稱/值組任意集合。Headers 屬性可以包含可表示為名稱/值組的通訊協定所需要的任何中繼資料。通常必須先設定這項資訊，再呼叫 GetRequestStream 或 GetResponse 方法；一旦提出要求，中繼資料即視為唯讀。

您不需要使用 Headers 屬性，也能使用標頭中繼資料。通訊協定特定的中繼資料可以公開為屬性。例

如，HttpWebRequest.UserAgent 屬性會公開 User-Agent HTTP 標頭。當您將標頭中繼資料公開為屬性時，您不應該允許使用 Headers 屬性設定相同的屬性。

Method 屬性

Method 屬性包含要求會要求伺服器執行的動詞或動作。Method 屬性的預設值必須啟用標準的要求/回應動作，不需要設定任何通訊協定特有的屬性。例如，Method 方法預設為 GET，會從網頁伺服器要求資源並傳回回應。

當 Method 屬性設定為動詞或動作，指出上傳正在進行中時，ContentLength 屬性一般必須設定成大於零的值。

PreAuthenticate 屬性

應用程式設定 PreAuthenticate 屬性，指出驗證資訊應該隨初始要求一起傳送，而不是等待驗證挑戰。只有當通訊協定支援隨初始要求一起傳送驗證認證時，PreAuthenticate 屬性才有意義。

Proxy 屬性

Proxy 屬性包含存取要求資源所使用的 IWebProxy 介面。只有當通訊協定支援 Proxy 的要求時，Proxy 屬性才有意義。如果您的通訊協定要求，您就必須設定預設的 Proxy。

在某些環境中，例如公司的防火牆後端，您的通訊協定可能要求使用 Proxy。在此情況下，您必須實作 IWebProxy 介面來建立適用於您通訊協定的 proxy 類別。

RequestUri 屬性

RequestUri 屬性包含的 URI 已傳遞至 WebRequest.Create 方法。它是唯讀的，且一旦建立 WebRequest 即無法變更。如果您的通訊協定支援重新導向，回應可能來自不同 URI 所識別的資源。如果您需要提供對所回應 URI 的存取權，即必須提供包含該 URI 的額外屬性。

Timeout 屬性

Timeout 屬性包含要求逾時及擲回例外狀況所需等候的時間長度，以毫秒為單位。Timeout 僅適用於以 GetResponse 方法提出的同步要求，非同步要求必須使用 Abort 方法來取消暫止的要求。

只有當特定通訊協定類別實作逾時處理序時，設定 Timeout 屬性才有意義。

Abort 方法

Abort 方法會取消伺服器的暫止非同步要求。取消要求之後，呼叫 GetResponse、BeginGetResponse、EndGetResponse、GetRequestStream、BeginGetRequestStream 或 EndGetRequestStream 會擲回將 Status 屬性設為 WebExceptionStatus 的 WebException。

BeginGetRequestStream 和 EndGetRequestStream 方法

BeginGetRequestStream 方法會啟動資料流的非同步要求，用來將資料上傳到伺服器。EndGetRequestStream 方法會完成非同步要求，並傳回所要求的資料流。這些方法會實作使用標準 .NET Framework 非同步模式的 GetRequestStream 方法。

BeginGetResponse 和 EndGetResponse 方法

BeginGetResponse 方法會啟動伺服器的非同步要求。EndGetResponse 方法會完成非同步要求，並傳回所要求的回應。這些方法會實作使用標準 .NET Framework 非同步模式的 GetResponse 方法。

GetRequestStream 方法

GetRequestStream 方法會傳回資料流，用來將資料寫入要求的伺服器。傳回的資料流應該是不會搜尋的唯寫資料流，它原來是要寫入伺服器的單向資料流資料。資料流針對 CanRead 和 CanSeek 屬性傳回 false，針對 CanWrite 屬性傳回 true。

GetRequestStream 方法通常會開啟伺服器連線，並在傳回資料流之前，先傳送標頭資訊，指出正將該資料傳送到伺服器。因為是 GetRequestStream 開始的要求，所以通常不允許在呼叫 GetRequestStream 之後設定任何 Header 屬性或 ContentLength 屬性。

GetResponse 方法

GetResponse 方法會傳回 WebResponse 類別的特定通訊協定子代，表示伺服器的回應。除非 GetRequestStream 方法已起始要求，否則 GetResponse 方法會建立 RequestUri 所識別的資源連線，傳送標頭資訊，指出正在提出的要求類型，然後從資源接收回應。

一旦呼叫了 GetResponse，所有屬性都應該視為唯讀。WebRequest 執行個體僅供單次使用，如果您想要提出另一項要求，您應該建立新的 WebRequest。

GetResponse 方法負責建立適當的 WebResponse 子代，以包含傳入的回應。

另請參閱

- WebRequest
- HttpWebRequest
- FileWebRequest
- 可插式通訊協定程式設計
- 衍生自 WebResponse

衍生自 WebResponse

2020/3/20 • • Edit Online

WebResponse 類別是抽象的基底類別，提供基本的方法和屬性以建立有特定通訊協定回應，符合 .NET Framework 插入式通訊協定模型的處理常式。使用 WebRequest 類別向資源要求資料的應用程式，會收到 WebResponse 的回應。通訊協定特定的 WebResponse 子代必須實作 WebResponse 類別的抽象成員。

相關聯的 WebRequest 類別必須建立 WebResponse 子代。例如，只有呼叫 HttpWebRequest.GetResponse 或 HttpWebRequest.EndGetResponse 才會建立 HttpWebResponse 執行個體。每個 WebResponse 都包含向資源提出要求的結果，而且不能重複使用。

ContentLength 屬性

ContentLength 屬性指出從 GetResponseStream 方法傳回的資料流中可取得的資料位元組數目。ContentLength 屬性不會指出伺服器所傳回的標頭或中繼資料資訊的位元組數目，它只會指出所要求資源本身的資料位元組數目。

ContentType 屬性

ContentType 屬性會提供您的通訊協定要求您傳送至用戶端的任何特殊資訊，以識別伺服器要傳送的內容類型。這通常是任何傳回資料的 MIME 內容類型。

Headers 屬性

Headers 屬性包含與回應建立關聯的中繼資料名稱/值組任意集合。Headers 屬性可以包含可表示為名稱/值組的通訊協定所需要的任何中繼資料。

您不需要使用 Headers 屬性，也能使用標頭中繼資料。通訊協定特定的中繼資料可以公開為屬性。例如，HttpWebResponse.LastModified 屬性會公開 Last-Modified HTTP 標頭。當您將標頭中繼資料公開為屬性時，您不應該允許使用 Headers 屬性設定相同的屬性。

ResponseUri 屬性

ResponseUri 屬性包含實際提供回應的資源 URI。至於不支援重新導向的通訊協定，ResponseUri 會和建立回應的 WebRequest 的 RequestUri 屬性相同。如果通訊協定支援重新導向要求，ResponseUri 會包含回應的 URI。

Close 方法

Close 方法關閉要求和回應所建立的任何連線，並清除回應所使用的資源。Close 方法會關閉回應所使用的任何資料流執行個體，但如果 Stream.Close 方法的呼叫之前已關閉回應資料流，它不會擲回例外狀況。

GetResponseStream 方法

GetResponseStream 方法傳回的資料流會包含來自所要求資源的回應。回應資料流只包含資源傳回的資料，回應中包含的任何標頭或中繼資料都應從回應中移除，並透過通訊協定特有的屬性或 Headers 屬性向應用程式公開。

GetResponseStream 方法所傳回的資料流執行個體為應用程式所擁有，而且不用關閉 WebResponse 即可關閉。依照慣例，呼叫 WebResponse.Close 方法也會關閉 GetResponse 傳回的資料流。

另請參閱

- WebResponse

- HttpWebResponse
- FileWebResponse
- 可插式通訊協定程式設計
- 衍生自 WebRequest

使用應用程式通訊協定

2020/3/20 • • Edit Online

.NET Framework 支援常用的網際網路應用程式通訊協定。本節包含有關使用 HTTP、"TCP"和 "UDP" 通訊協定的資訊，以及有關使用 Windows Sockets 介面以實作自訂通訊協定的資訊。

另請參閱

- .NET 框架中的網路程式設計
- 網路程式設計範例

HTTP

2020/3/20 • • Edit Online

.NET Framework 可透過 HttpWebRequest 和 HttpWebResponse 類別，對於佔據所有網際網路流量絕大部分的 HTTP 通訊協定提供全面的支援。這些類別衍生自 WebRequest 和 WebResponse，每當靜態方法 WebRequest.Create 碰到以 "http" 或 "https" 開頭的 URI 時，預設會傳回這些類別。在大部分情況下，**WebRequest** 和 **WebResponse** 類別提供了提出要求所需的一切功能，但是如果您需要存取以屬性方式公開的 HTTP 特定功能，則可以將這些類別轉型為 **HttpWebRequest** 或 **HttpWebResponse**。

HttpWebRequest 和 **HttpWebResponse** 封裝了標準的 HTTP 要求與回應交易，並提供對一般 HTTP 標頭的存取。這些類別也支援大部分的 HTTP 1.1 功能，包括管線操作、以區塊方式傳送和接收資料 、驗證、預先驗證、加密、Proxy 支援、伺服器憑證驗證，以及連線管理。自訂標頭和未透過屬性提供的標頭則可透過 **Headers** 屬性來儲存和存取。

HttpWebRequest 是 **WebRequest** 所使用的預設類別，在您將 URI 傳遞至 **WebRequest.Create** 方法之前，不需要登錄此類別。

您可以將 AllowAutoRedirect 屬性設定為 **true** (預設值)，讓您的應用程式自動遵循 HTTP 重新導向。應用程式將重新導向要求，而 **HttpWebResponse** 的 ResponseUri 屬性會包含回應要求的實際 Web 資源。如果將 **AllowAutoRedirect** 設定為 **false**，則您的應用程式必須能夠處理重新導向當成 HTTP 通訊協定錯誤。

應用程式藉由將 Status 設為 WebExceptionStatus 來捕捉 WebException，以便接收 HTTP 通訊協定錯誤。Response 屬性包含伺服器所傳送的 **WebResponse**，並指出實際發生的 HTTP 錯誤。

另請參閱

- 通過代理訪問互聯網
- 使用應用程式通訊協定
- 何：存取 HTTP 特定屬性

如何：存取 HTTP 特定屬性

2020/3/20 • • Edit Online

這個範例示範如何關閉 HTTP **保持運作**行為，以及如何從 Web 伺服器取得通訊協定版本號碼。

範例

```
Dim HttpWReq As HttpWebRequest= _
    CType(WebRequest.Create("http://www.contoso.com"), HttpWebRequest)
' Turn off connection keep-alives.
HttpWReq.KeepAlive = False

Dim HttpWResp As HttpWebResponse = _
    CType(HttpWReq.GetResponse(), HttpWebResponse)

' Get the HTTP protocol version number returned by the server.
Dim ver As String = HttpWResp.ProtocolVersion.ToString()
HttpWResp.Close()
```

```
HttpWebRequest HttpWReq =
    (HttpWebRequest)WebRequest.Create("http://www.contoso.com");
// Turn off connection keep-alives.
HttpWReq.KeepAlive = false;

HttpWebResponse HttpWResp = (HttpWebResponse)HttpWReq.GetResponse();

// Get the HTTP protocol version number returned by the server.
String ver = HttpWResp.ProtocolVersion.ToString();
HttpWResp.Close();
```

編譯程式碼

這個範例需要：

- 對 System.Net 命名空間的參考。

另請參閱

- 通過代理訪問互聯網
- 使用應用程式通訊協定
- HTTP

管理連接

2020/3/20 • • Edit Online

使用 HTTP 連線至資料資源的應用程式可以使用 .NET Framework 的 ServicePoint 和 ServicePointManager 類別管理網際網路連線，以及協助它們達到最佳規模和效能。

ServicePoint 類別所提供的應用程式具有應用程式可連線以存取網際網路資源的端點。每個 ServicePoint 都會包含資訊來協助最佳化與網際網路伺服器的連線，方法是在連線之間共用最佳化資訊來改善效能。

每個 ServicePoint 都是透過統一資源識別項 (URI) 進行識別，並且根據 URI 的配置識別碼和主機片段來進行分類。例如，相同的 ServicePoint 執行個體會提供 URI `http://www.contoso.com/index.htm` 和 `http://www.contoso.com/news.htm?date=today` 的要求，因為它們具有相同的配置識別碼 (http) 和主機片段 (`www.contoso.com`)。如果應用程式已持續連線至伺服器 `www.contoso.com`，則會使用該連線來擷取這兩個要求，避免需要建立兩個連線。

ServicePointManager 是靜態類別，可管理 ServicePoint 執行個體的建立和解構。應用程式要求不在現有 ServicePoint 執行個體集合中的網際網路資源時，ServicePointManager 會建立 ServicePoint。ServicePoint 執行個體在超過其最大閒置時間或是現有 ServicePoint 執行個體數目超過應用程式的最大 ServicePoint 執行個體數目時即會予以終結。預設最大閒置時間和最大 ServicePoint 執行個體數目的控制方式是在 ServicePointManager 上設定 MaxServicePointIdleTime 和 MaxServicePoints 屬性。

用戶端與伺服器之間的連線數目會對應用程式輸送量造成很大的影響。根據預設，使用 HttpWebRequest 類別的應用程式最多會使用兩個與指定伺服器的持續性連線，但您可以設定個別應用程式的最大連線數目。

> **NOTE**
> HTTP/1.1 規格會將應用程式的連線數目限制成一部伺服器有兩個連線。

最佳連線數目取決於實際執行應用程式的條件。增加應用程式的可用連線數目，可能不會影響應用程式效能。若要判斷多個連線的影響，請執行效能測試，同時改變連線數目。您可以在應用程式初始化時變更 ServicePointManager 類別上的靜態 DefaultConnectionLimit 屬性，以變更應用程式所使用的連線數目，如下列程式碼範例所示。

```
// Set the maximum number of connections per server to 4.
ServicePointManager.DefaultConnectionLimit = 4;
```

```
' Set the maximum number of connections per server to 4.
ServicePointManager.DefaultConnectionLimit = 4
```

變更 ServicePointManager.DefaultConnectionLimit 屬性並不會影響先前初始化的 ServicePoint 執行個體。下列程式碼示範如何將伺服器 `http://www.contoso.com` 之現有 ServicePoint 的連線限制變更為 `newLimit` 中所儲存的值。

```
Uri uri = new Uri("http://www.contoso.com/");
ServicePoint sp = ServicePointManager.FindServicePoint(uri);
sp.ConnectionLimit = newLimit;
```

```
Dim uri As New Uri("http://www.contoso.com/")
Dim sp As ServicePoint = ServicePointManager.FindServicePoint(uri)
sp.ConnectionLimit = newLimit
```

另請參閱

- 連線群組
- 使用應用程式通訊協定

連接群組

2020/3/20 • • Edit Online

連線群組會建立與單一應用程式內所定義連接集區之特定要求的關聯。連線至代表使用者的後端伺服器並使用支援委派的驗證通訊協定 (例如 Kerberos) 的中介層應用程式，或者提供其專屬認證的中介層應用程式，可能需要這項作業，如下列範例所示。例如，假設使用者 Joe 瀏覽內部網站，以顯示其薪資資訊。驗證 Joe 之後，中介層應用程式伺服器會使用 Joe 的認證來連線至後端伺服器，以擷取其薪資資訊。接下來，Susan 會瀏覽網站，並要求其薪資資訊。因為中介層應用程式已經使用 Joe 的認證進行連線，所以後端伺服器會回應 Joe 的資訊。不過，如果應用程式將傳送至後端伺服器的每個要求指派給使用者名稱所組成的連線群組，則每位使用者都屬於不同的連接集區，因此不會意外與其他使用者共用驗證資訊。

若要將要求指派給特定連線群組，您必須先將名稱指派給 WebRequest 的 ConnectionGroupName 屬性，再提出要求。

另請參閱

- 管理連接
- 如何：將使用者資訊指派給群組連線

如何：將使用者資訊指派給群組連接

2020/3/20 • • Edit Online

下列範例示範如何將使用者資訊指派給群組連線，並假設應用程式會在呼叫這段程式碼之前設定 *UserName*、*SecurelyStoredPassword* 和 *Domain* 變數，且 *UserName* 為唯一。

將使用者資訊指派給群組連線

1. 建立連線群組名稱。

```
SHA1Managed Sha1 = new SHA1Managed();
Byte[] updHash = Sha1.ComputeHash(Encoding.UTF8.GetBytes(UserName + SecurelyStoredPassword + Domain));
String secureGroupName = Encoding.Default.GetString(updHash);
```

```
Dim Sha1 As New SHA1Managed()
Dim updHash As [Byte]() = Sha1.ComputeHash(Encoding.UTF8.GetBytes((UserName + SecurelyStoredPassword +
Domain)))
Dim secureGroupName As [String] = Encoding.Default.GetString(updHash)
```

2. 建立特定 URL 的要求。例如，下列程式碼會建立 URL `http://www.contoso.com.` 的要求

```
WebRequest myWebRequest=WebRequest.Create("http://www.contoso.com");
```

```
Dim myWebRequest As WebRequest = WebRequest.Create("http://www.contoso.com")
```

3. 設定 Web 要求的認證和連線群組名稱，並呼叫 **GetResponse** 來擷取 **WebResponse** 物件。

```
myWebRequest.Credentials = new NetworkCredential(UserName, SecurelyStoredPassword, Domain);
myWebRequest.ConnectionGroupName = secureGroupName;

WebResponse myWebResponse=myWebRequest.GetResponse();
```

```
myWebRequest.Credentials = New NetworkCredential(UserName, SecurelyStoredPassword, Domain)
myWebRequest.ConnectionGroupName = secureGroupName

Dim myWebResponse As WebResponse = myWebRequest.GetResponse()
```

4. 使用 WebRespose 物件之後，關閉回應資料流。

```
MyWebResponse.Close();
```

```
MyWebResponse.Close()
```

範例

```
// Create a connection group name.
SHA1Managed Sha1 = new SHA1Managed();
Byte[] updHash = Sha1.ComputeHash(Encoding.UTF8.GetBytes(UserName + SecurelyStoredPassword + Domain));
String secureGroupName = Encoding.Default.GetString(updHash);

// Create a request for a specific URL.
WebRequest myWebRequest=WebRequest.Create("http://www.contoso.com");

myWebRequest.Credentials = new NetworkCredential(UserName, SecurelyStoredPassword, Domain);
myWebRequest.ConnectionGroupName = secureGroupName;

WebResponse myWebResponse=myWebRequest.GetResponse();

// Insert the code that uses myWebResponse.

MyWebResponse.Close();
```

```
' Create a secure group name.
Dim Sha1 As New SHA1Managed()
Dim updHash As [Byte]() = Sha1.ComputeHash(Encoding.UTF8.GetBytes((UserName + SecurelyStoredPassword +
Domain)))
Dim secureGroupName As [String] = Encoding.Default.GetString(updHash)

' Create a request for a specific URL.
Dim myWebRequest As WebRequest = WebRequest.Create("http://www.contoso.com")

myWebRequest.Credentials = New NetworkCredential(UserName, SecurelyStoredPassword, Domain)
myWebRequest.ConnectionGroupName = secureGroupName

Dim myWebResponse As WebResponse = myWebRequest.GetResponse()

' Insert the code that uses myWebResponse.
MyWebResponse.Close()
```

另請參閱

- 管理連接
- 連線群組

TCP-UDP

2020/3/20 • • Edit Online

應用程式可以使用「傳輸控制通訊協定 (TCP)」和「使用者資料包通訊協定 (UDP)」服務，搭配 TcpClient、TcpListener 和 UdpClient 類別。這些通訊協定類別建立在 System.Net.Sockets.Socket 類別之上，並負責傳送資料的細節。

通訊協定類別使用 Socket 類別的同步方法，以提供簡單且直接的網路服務存取，不需要維護狀態資訊或了解設定通訊協定特定通訊端的詳細資料等成本。若要使用非同步 Socket 方法，您可以使用 NetworkStream 類別所提供的非同步方法。若要存取 Socket 類別中通訊協定類別未公開的功能，您必須使用 Socket 類別。

TcpClient 和 TcpListener 代表使用 NetworkStream 類別的網路。您使用 GetStream 方法來傳回網路資料流，然後呼叫資料流的 Read 和 Write 方法。NetworkStream 並未擁有通訊協定類別的基礎通訊端，因此關閉它並不會影響通訊端。

UdpClient 類別使用位元組陣列來保留 UDP 資料包。您使用 Send 方法將資料傳送到網路，及使用 Receive 方法來接收連入的資料包。

另請參閱

- 使用 TCP 服務
- 使用 UDP 服務
- 在網路上使用資料流
- 使用非同步伺服器通訊端
- 使用非同步用戶端通訊端
- 使用應用程式通訊協定

使用 TCP 服務

2020/3/20 • • Edit Online

TcpClient 類別會使用 TCP 向網際網路資源要求資料。**TcpClient** 的屬性和方法取出的詳細資料，可用來建立 Socket 以使用 TCP 要求和接收資料。因為遠端裝置的連線是以資料流表示，所以可以使用 .NET Framework 資料流處理技術來讀取和寫入資料。

TCP 通訊協定會建立與遠端端點的連線，然後使用該連接來傳送和接收資料封包。TCP 負責確保將資料封包傳送到端點，並在送達時以正確的順序組合。

若要建立 TCP 傳線，您必須知道裝載所需服務之網路裝置的位址，以及服務用來通訊的 TCP 連接埠。Internet Assigned Numbers Authority (Iana) 定義了通用服務的連接埠編號。(請參閱服務名稱與傳輸通訊協定連接埠號碼登錄 (英文))。不在 Iana 清單上之服務的連接埠編號範圍可以是 1,024 到 65,535。

下面的示例演示了設置**TcpClient**以連接到 TCP 埠 13 上的時間伺服器：

```
Imports System.Net.Sockets
Imports System.Text

Public Class TcpTimeClient
    Private const portNum As Integer = 13
    Private const hostName As String = "host.contoso.com"

    ' Entry point  that delegates to C-style main Private Function.
    Public Overloads Shared Sub Main()
        System.Environment.ExitCode = _
            Main(System.Environment.GetCommandLineArgs())
    End Sub

    Overloads Public Shared Function Main(args As String()) As Integer
        Try
            Dim client As New TcpClient(hostName, portNum)

            Dim ns As NetworkStream = client.GetStream()

            Dim bytes(1024) As Byte
            Dim bytesRead As Integer = ns.Read(bytes, 0, bytes.Length)

            Console.WriteLine(Encoding.ASCII.GetString(bytes, 0, bytesRead))

        Catch e As Exception
            Console.WriteLine(e.ToString())
        End Try

        client.Close()

        Return 0
    End Function
End Class
```

```
using System;
using System.Net.Sockets;
using System.Text;

public class TcpTimeClient
{
    private const int portNum = 13;
    private const string hostName = "host.contoso.com";

    public static int Main(String[] args)
    {
        try
        {
            var client = new TcpClient(hostName, portNum);

            NetworkStream ns = client.GetStream();

            byte[] bytes = new byte[1024];
            int bytesRead = ns.Read(bytes, 0, bytes.Length);

            Console.WriteLine(Encoding.ASCII.GetString(bytes,0,bytesRead));

            client.Close();

        }
        catch (Exception e)
        {
            Console.WriteLine(e.ToString());
        }

        return 0;
    }
}
```

TcpListener用於監視傳入請求的 TCP 埠，然後創建套接字或TcpClient來管理與用戶端的連接。Start 方法可啟用接聽，而 Stop 方法則可停用連接埠上的接聽。AcceptTcpClient 方法會接受連入連線要求，並建立 TcpClient 來處理要求，而 AcceptSocket 方法會接受連入連線要求，並建立 Socket 來處理要求。

下列範例示範如何建立使用 TcpListener 來監視 TCP 連接埠 13 的網路時間伺服器。接受連入連線要求時，時間伺服器會回應主機伺服器的目前日期和時間。

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Text

Public Class TcpTimeServer

    Private const portNum As Integer = 13

    ' Entry point that delegates to C-style main Private Function.
    Public Overloads Shared Sub Main()
        System.Environment.ExitCode = _
            Main(System.Environment.GetCommandLineArgs())
    End Sub

    Overloads Public Shared Function Main(args As String()) As Integer
        Dim done As Boolean = False

        Dim listener As New TcpListener(IPAddress.Any, portNum)

        listener.Start()

        While Not done
            Console.Write("Waiting for connection...")
            Dim client As TcpClient = listener.AcceptTcpClient()

            Console.WriteLine("Connection accepted.")
            Dim ns As NetworkStream = client.GetStream()

            Dim byteTime As Byte() = _
                Encoding.ASCII.GetBytes(DateTime.Now.ToString())

            Try
                ns.Write(byteTime, 0, byteTime.Length)
                ns.Close()
                client.Close()
            Catch e As Exception
                Console.WriteLine(e.ToString())
            End Try
        End While

        listener.Stop()

        Return 0
    End Function
End Class
```

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class TcpTimeServer
{

    private const int portNum = 13;

    public static int Main(String[] args)
    {
        bool done = false;

        var listener = new TcpListener(IPAddress.Any, portNum);

        listener.Start();

        while (!done)
        {
            Console.Write("Waiting for connection...");
            TcpClient client = listener.AcceptTcpClient();

            Console.WriteLine("Connection accepted.");
            NetworkStream ns = client.GetStream();

            byte[] byteTime = Encoding.ASCII.GetBytes(DateTime.Now.ToString());

            try
            {
                ns.Write(byteTime, 0, byteTime.Length);
                ns.Close();
                client.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }

        listener.Stop();

        return 0;
    }

}
```

使用 UDP 服務

2020/5/13 • • Edit Online

UdpClient 類別使用 UDP 與網路服務通訊。UdpClient 類別屬性和方法使用 UDP 取出建立 Socket 來要求和接收資料的詳細資料。

使用者資料包通訊協定 (UDP) 是一種簡單的通訊協定，致力於將資料傳遞到遠端主機。不過，因為 UDP 通訊協定是一種無連線的通訊協定，所以傳送到遠端端點的 UDP 資料包不保證會送達，也不保證會以傳送的相同順序送達。使用 UDP 應用程式必須準備好處理遺失、重複和不依順序的資料包。

若要使用 UDP 傳送資料包，您必須知道裝載所需服務的網路裝置的網路位址，以及服務用來通訊的 UDP 連接埠編號。網際網路指派的號碼授權單位（IANA）定義了泛型服務的埠號碼（請參閱服務名稱和傳輸通訊協定埠號碼登錄）。不在 IANA 清單上的服務，其埠號碼會在1024到65535的範圍內。

特殊的網路位址可用來支援 IP 型網路上的 UDP 廣播訊息。下列討論內容會以網際網路上使用的 IP 第 4 版位址系列作為範例。

IP 第 4 版位址會使用 32 位元來指定網路位址。對於使用網路遮罩 255.255.255.0 的類別 C 位址，這些位元會分成四個八位元。以十進位表示時，四個八位元會形成熟悉的四點區隔標記法，例如 192.168.100.2。前兩個八位元 (在此範例中為 192.168) 形成網路編號，第三個八位元 (100) 定義子網路，而最後一個八位元 (2) 主機識別碼。

將 IP 位址的所有位元設定為其中一個或 255.255.255.255，構成受限的廣播位址。將 UDP 資料包傳送到這個位址時，會將訊息傳遞到本機網路區段上的任何主機。因為路由器永遠不會轉送傳送到這個位址的訊息，所以只有網路區段上的主機會收到廣播訊息。

藉由設定主機識別碼的所有位元，即可將廣播導向至網路的特定部分。例如，若要將廣播傳送至開頭為 192.168.1 之 IP 位址所識別網路上的所有主機，請使用位址 192.168.1.255。

下列程式碼範例使用 UdpClient，在連接埠 11,000 上接聽 UDP 資料包。用戶端會接收訊息字串，並將訊息寫入至主控台。

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Text

Public Class UDPListener
   Private Const listenPort As Integer = 11000

   Private Shared Sub StartListener()
      Dim listener As New UdpClient(listenPort)
      Dim groupEP As New IPEndPoint(IPAddress.Any, listenPort)
      Try
         While True
            Console.WriteLine("Waiting for broadcast")
            Dim bytes As Byte() = listener.Receive(groupEP)
            Console.WriteLine("Received broadcast from {0} :", groupEP)
            Console.WriteLine(Encoding.ASCII.GetString(bytes, 0, bytes.Length))
         End While
      Catch e As SocketException
         Console.WriteLine(e)
      Finally
         listener.Close()
      End Try
   End Sub 'StartListener

   Public Shared Sub Main()
      StartListener()
   End Sub 'Main
End Class 'UDPListener
```

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class UDPListener
{
    private const int listenPort = 11000;

    private static void StartListener()
    {
        UdpClient listener = new UdpClient(listenPort);
        IPEndPoint groupEP = new IPEndPoint(IPAddress.Any, listenPort);

        try
        {
            while (true)
            {
                Console.WriteLine("Waiting for broadcast");
                byte[] bytes = listener.Receive(ref groupEP);

                Console.WriteLine($"Received broadcast from {groupEP} :");
                Console.WriteLine($" {Encoding.ASCII.GetString(bytes, 0, bytes.Length)}");
            }
        }
        catch (SocketException e)
        {
            Console.WriteLine(e);
        }
        finally
        {
            listener.Close();
        }
    }

    public static void Main()
    {
        StartListener();
    }
}
```

下列程式碼範例使用 Socket透過連接埠 11,000 將 UDP 資料包傳送到導向廣播位址 192.168.1.255。用戶端會傳送命令列上指定的訊息字串。

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Text

Public Class Program
    Public Shared Sub Main(args() As [String])
      Dim s As New Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp)
      Dim broadcast As IPAddress = IPAddress.Parse("192.168.1.255")
      Dim sendbuf As Byte() = Encoding.ASCII.GetBytes(args(0))
      Dim ep As New IPEndPoint(broadcast, 11000)
      s.SendTo(sendbuf, ep)
      Console.WriteLine("Message sent to the broadcast address")
   End Sub 'Main
End Class
```

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
        Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

        IPAddress broadcast = IPAddress.Parse("192.168.1.255");

        byte[] sendbuf = Encoding.ASCII.GetBytes(args[0]);
        IPEndPoint ep = new IPEndPoint(broadcast, 11000);

        s.SendTo(sendbuf, ep);

        Console.WriteLine("Message sent to the broadcast address");
    }
}
```

另請參閱

- UdpClient
- IPAddress

通訊端

2020/3/20 • • Edit Online

System.Net.Sockets 命名空間包含 Windows Sockets 介面的 Managed 實作。System.Net 命名空間中的所有其他網路存取類別都建立在此通訊端實作之上。

.NET Framework Socket 類別是 Winsock32 API 所提供的通訊端服務 Managed 程式碼版本。在大部分情況下，**Socket** 類別方法，只會將資料封送處理成原生 Win32 相對物，並處理任何必要的安全性檢查。

Socket 類別支援兩種基本的模式，同步和非同步。在同步模式中，呼叫執行網路作業的函式 (例如 Send 和 Receive) 會等候作業完成，然後才將控制權傳回給呼叫端程式。在非同步模式中，這些呼叫會立即傳回。

另請參閱

- 如何：建立通訊端
- 使用應用程式通訊協定

如何：建立通訊端

2020/3/20 • • Edit Online

在您使用通訊端與遠端裝置進行通訊之前，必須先使用通訊協定和網路位址資訊初始化通訊端。Socket 類別的建構函式所擁有的參數，可指定通訊端用來建立連線的位址家族、通訊端類型和通訊協定類型。

範例

下列範例會建立可在 TCP/IP 網路 (例如網際網路) 上用於通訊的通訊端。

```
Socket s = new Socket(AddressFamily.InterNetwork,
   SocketType.Stream, ProtocolType.Tcp);
```

```
Dim s as New Socket(AddressFamily.InterNetwork, _
   SocketType.Stream, ProtocolType.Tcp)
```

若要使用 UDP 而不是 TCP，請變更通訊協定類型，如下列範例所示：

```
Socket s = new Socket(AddressFamily.InterNetwork,
   SocketType.Dgram, ProtocolType.Udp);
```

```
Dim s as New Socket(AddressFamily.InterNetwork, _
   SocketType.Dgram, ProtocolType.Udp)
```

AddressFamily 列舉會指定 Socket 用來解析網路位址的標準位址系列 (例如，AddressFamily.InterNetwork 成員指定 IP 第 4 版位址系列)。

SocketType 列舉會指定通訊端類型 (例如，SocketType.Stream 成員指出使用流量控制傳送和接收資料的標準通訊端)。

ProtocolType 列舉會指定在 Socket 上進行通訊時，要使用的網路通訊協定 (例如，ProtocolType.Tcp 指出通訊端會使用 TCP；ProtocolType.Udp 則指出通訊端會使用 UDP)。

建立 Socket 之後，它可以初始化與遠端端點的連線或接收來自遠端裝置的連線。

另請參閱

- 使用用戶端通訊端
- 透過通訊端接聽

使用用戶端通訊端

2020/3/20 • • Edit Online

在透過 Socket 起始交談之前，您必須先建立應用程式和遠端裝置之間的資料管道。雖然有其他的網路位址系列和通訊協定存在，但此範例會示範如何建立遠端服務的 TCP/IP 連線。

TCP/IP 會使用網路位址和服務連接埠編號來唯一識別服務。網路位址可識別網路上的特定裝置；連接埠編號則可識別該裝置上要連線的特定服務。網路位址和服務連接埠的組合稱為端點，在 .NET Framework 中是以 EndPoint 類別表示。每個支援的位址系列已定義 **EndPoin** 的子系；對於 IP 位址系列，此類別是 IPEndPoint。

Dns 類別會提供網域名稱服務給使用 TCP/IP 網際網路服務的應用程式。Resolve 方法會查詢 DNS 伺服器，以將使用者易記的網域名稱 (例如 "host.contoso.com") 對應到數字的網際網路位址 (例如 192.168.1.1)。**Resolve** 會傳回 IPHostEntry，其中包含位址和所要求名稱之別名的清單。在大部分情況下，您可以使用 AddressList 陣列中傳回的第一個位址。下列程式碼可取得包含伺服器 host.contoso.com 之 IP 位址的 IPAddress。

```
Dim ipHostInfo As IPHostEntry = Dns.Resolve("host.contoso.com")
Dim ipAddress As IPAddress = ipHostInfo.AddressList(0)
```

```
IPHostEntry ipHostInfo = Dns.Resolve("host.contoso.com");
IPAddress ipAddress = ipHostInfo.AddressList[0];
```

Internet Assigned Numbers Authority (Iana) 定義了通用服務的連接埠編號 (如需詳細資訊，請參閱服務名稱與傳輸通訊協定連接埠號碼登錄 (英文))。其他服務的登錄連接埠號碼範圍可以是 1,024 到 65,535。下列程式碼會合併 host.contoso.com 的 IP 位址與連接埠編號，以建立連線的遠端端點。

```
Dim ipe As New IPEndPoint(ipAddress, 11000)
```

```
IPEndPoint ipe = new IPEndPoint(ipAddress,11000);
```

在判斷遠端裝置的位址並選擇要用於連線的連接埠之後，應用程式可以嘗試建立與遠端裝置的連線。下列範例會使用現有 **IPEndPoint** 連線到遠端裝置，並捕捉任何擲回的例外狀況。

```
Try
    s.Connect(ipe)
Catch ae As ArgumentNullException
    Console.WriteLine("ArgumentNullException : {0}", _
        ae.ToString())
Catch se As SocketException
    Console.WriteLine("SocketException : {0}", se.ToString())
Catch e As Exception
    Console.WriteLine("Unexpected exception : {0}", e.ToString())
End Try
```

```
try {
    s.Connect(ipe);
} catch(ArgumentNullException ae) {
    Console.WriteLine("ArgumentNullException : {0}", ae.ToString());
} catch(SocketException se) {
    Console.WriteLine("SocketException : {0}", se.ToString());
} catch(Exception e) {
    Console.WriteLine("Unexpected exception : {0}", e.ToString());
}
```

另請參閱

- 使用同步用戶端通訊端
- 使用非同步用戶端通訊端
- 如何：建立通訊端
- 通訊端

使用同步用戶端通訊端

2020/3/20 • • Edit Online

在網路作業完成時，同步用戶端通訊端會暫止應用程式。同步通訊端不適用於大量使用網路以進行作業的應用程式，但它們可以啟用其他應用程式的網路服務簡單存取。

若要傳送資料，請傳遞位元組陣列給其中一個 Socket 類別的資料傳送方法 (Send 和 SendTo)。下列範例會使用 Encoding.ASCII 屬性將字串編碼成位元組陣列緩衝區，然後使用 Send 方法將緩衝區傳送給網路裝置。Send 方法會傳回送給網路裝置的位元組數目。

```
Dim msg As Byte() = _
    System.Text.Encoding.ASCII.GetBytes("This is a test.")
Dim bytesSent As Integer = s.Send(msg)
```

```
byte[] msg = System.Text.Encoding.ASCII.GetBytes("This is a test");
int bytesSent = s.Send(msg);
```

Send 方法從緩衝區移除位元組，並將它們佇列到要傳送至網路裝置的網路介面。網路介面可能不會立即傳送資料，但它終究會傳送，只要以 Shutdown 方法正常關閉連線。

若要從網路裝置接收資料，請傳遞緩衝區給其中一個 Socket 類別的資料接收方法 (Receive 和 ReceiveFrom)。同步通訊端會暫停應用程式，直到從網路收到位元組或通訊端關閉為止。下列範例會從網路接收資料，再將它顯示在主控台上。此範例假設來自網路的資料為 ASCII 編碼的文字。Receive 方法會傳回從網路收到的位元組數目。

```
Dim bytes(1024) As Byte
Dim bytesRec = s.Receive(bytes)
Console.WriteLine("Echoed text = {0}", _
    System.Text.Encoding.ASCII.GetString(bytes, 0, bytesRec))
```

```
byte[] bytes = new byte[1024];
int bytesRec = s.Receive(bytes);
Console.WriteLine("Echoed text = {0}",
    System.Text.Encoding.ASCII.GetString(bytes, 0, bytesRec));
```

當不再需要通訊端時，您需要藉由呼叫 Shutdown 方法，然後呼叫 Close 方法釋放它。下列範例會釋放**通訊端**。SocketShutdown 列舉定義常數，表示傳送、接收或是兩者時，是否應該關閉通訊端。

```
s.Shutdown(SocketShutdown.Both)
s.Close()
```

```
s.Shutdown(SocketShutdown.Both);
s.Close();
```

另請參閱

- 使用非同步用戶端通訊端
- 透過通訊端接聽
- 同步用戶端通訊端範例

使用非同步用戶端通訊端

2020/3/20 • • Edit Online

非同步用戶端通訊端等待網路作業完成時，不會暫停應用程式。相反地，它會使用標準 .NET Framework 非同步程式設計模型，在一個執行緒上處理網路連線，同時應用程式繼續在原始執行緒上執行。非同步通訊端適用於大量使用網路的應用程式，或是無法等候網路作業完成再繼續進行的應用程式。

Socket 類別遵循非同步方法的 .NET Framework 命名模式；例如，同步 Receive 方法對應於非同步 BeginReceive 和 EndReceive 方法。

非同步作業需要回呼方法以傳回作業的結果。如果您的應用程式不需要知道結果，則不需要回呼方法。本節的範例程式碼示範如何使用方法來開始連線到網路裝置以及使用回呼方法完成連線、使用方法開始傳送資料以及使用回呼方法完成傳送，還有使用方法開始接收資料以及使用回呼方法結束接收資料。

非同步通訊端使用系統執行緒集區中的多個執行緒來處理網路連線。一個執行緒負責初始化資料的傳送或接收；其他執行緒則完成與網路裝置的連線並且傳送或接收資料。在下列範例中，System.Threading.ManualResetEvent 類別的執行個體會暫停執行主執行緒，並在可以繼續執行時發出訊號。

在下列範例中，為了將非同步通訊端連線到網路裝置，`Connect` 方法會初始化**通訊端**，然後呼叫 Socket.Connect 方法、傳遞代表網路裝置的遠端端點、連線回呼方法和狀態物件 (用戶端**通訊端**)，這用來在非同步呼叫之間傳遞狀態資訊。此範例會實作 `Connect` 方法，將指定的**通訊端**連線到指定的端點。它會假設一個名為 `connectDone` 的全域 **ManualResetEvent**。

```
Public Shared Sub Connect(remoteEP As EndPoint, client As Socket)
    client.BeginConnect(remoteEP, _
       AddressOf ConnectCallback, client)

    connectDone.WaitOne()
End Sub 'Connect
```

```
public static void Connect(EndPoint remoteEP, Socket client) {
    client.BeginConnect(remoteEP,
        new AsyncCallback(ConnectCallback), client );

   connectDone.WaitOne();
}
```

連線回呼方法 `ConnectCallback` 會實作 AsyncCallback 委派。它在遠端裝置可用時連線到遠端裝置，然後設定**ManualResetEvent** `connectDone`，通知應用程式執行緒連線已完成。下例程式碼會實作 `ConnectCallback` 方法。

```
Private Shared Sub ConnectCallback(ar As IAsyncResult)
    Try
        ' Retrieve the socket from the state object.
        Dim client As Socket = CType(ar.AsyncState, Socket)

        ' Complete the connection.
        client.EndConnect(ar)

        Console.WriteLine("Socket connected to {0}", _
            client.RemoteEndPoint.ToString())

        ' Signal that the connection has been made.
        connectDone.Set()
    Catch e As Exception
        Console.WriteLine(e.ToString())
    End Try
End Sub 'ConnectCallback
```

```
private static void ConnectCallback(IAsyncResult ar) {
    try {
        // Retrieve the socket from the state object.
        Socket client = (Socket) ar.AsyncState;

        // Complete the connection.
        client.EndConnect(ar);

        Console.WriteLine("Socket connected to {0}",
            client.RemoteEndPoint.ToString());

        // Signal that the connection has been made.
        connectDone.Set();
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

範例方法 `Send` 將指定的字串資料以 ASCII 格式編碼，並將它以非同步方式傳送到指定通訊端所代表的網路裝置。下列範例會實作 `Send` 方法。

```
Private Shared Sub Send(client As Socket, data As [String])
    ' Convert the string data to byte data using ASCII encoding.
    Dim byteData As Byte() = Encoding.ASCII.GetBytes(data)

    ' Begin sending the data to the remote device.
    client.BeginSend(byteData, 0, byteData.Length, SocketFlags.None, _
        AddressOf SendCallback, client)
End Sub 'Send
```

```
private static void Send(Socket client, String data) {
    // Convert the string data to byte data using ASCII encoding.
    byte[] byteData = Encoding.ASCII.GetBytes(data);

    // Begin sending the data to the remote device.
    client.BeginSend(byteData, 0, byteData.Length, SocketFlags.None,
        new AsyncCallback(SendCallback), client);
}
```

傳送回呼方法 `SendCallback` 會實作 AsyncCallback 委派。網路裝置準備好接收時，它會傳送資料。下列範例會示範 `SendCallback` 方法的實作。它會假設一個名為 `sendDone` 的全域 ManualResetEvent。

```
Private Shared Sub SendCallback(ar As IAsyncResult)
    Try
        ' Retrieve the socket from the state object.
        Dim client As Socket = CType(ar.AsyncState, Socket)

        ' Complete sending the data to the remote device.
        Dim bytesSent As Integer = client.EndSend(ar)
        Console.WriteLine("Sent {0} bytes to server.", bytesSent)

        ' Signal that all bytes have been sent.
        sendDone.Set()
    Catch e As Exception
        Console.WriteLine(e.ToString())
    End Try
End Sub 'SendCallback
```

```
private static void SendCallback(IAsyncResult ar) {
    try {
        // Retrieve the socket from the state object.
        Socket client = (Socket) ar.AsyncState;

        // Complete sending the data to the remote device.
        int bytesSent = client.EndSend(ar);
        Console.WriteLine("Sent {0} bytes to server.", bytesSent);

        // Signal that all bytes have been sent.
        sendDone.Set();
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

從用戶端通訊端讀取資料時，需要在非同步呼叫之間傳遞值的狀態物件。下列類別是從用戶端通訊端接收資料的範例狀態物件。它包含用戶端通訊端的欄位、已接收資料的緩衝區，以及儲存進入之資料字串的 StringBuilder。將這些欄位放置在狀態物件，可允許在多個呼叫間保留其值，以從用戶端通訊端讀取資料。

```
Public Class StateObject
    ' Client socket.
    Public workSocket As Socket = Nothing
    ' Size of receive buffer.
    Public BufferSize As Integer = 256
    ' Receive buffer.
    Public buffer(256) As Byte
    ' Received data string.
    Public sb As New StringBuilder()
End Class 'StateObject
```

```
public class StateObject {
    // Client socket.
    public Socket workSocket = null;
    // Size of receive buffer.
    public const int BufferSize = 256;
    // Receive buffer.
    public byte[] buffer = new byte[BufferSize];
    // Received data string.
    public StringBuilder sb = new StringBuilder();
}
```

`Receive` 方法範例會設定狀態物件，然後呼叫 BeginReceive 方法，以非同步方式從用戶端通訊端讀取資料。下列範例會實作 `Receive` 方法。

```
Private Shared Sub Receive(client As Socket)
    Try
        ' Create the state object.
        Dim state As New StateObject()
        state.workSocket = client

        ' Begin receiving the data from the remote device.
        client.BeginReceive(state.buffer, 0, state.BufferSize, 0, _
            AddressOf ReceiveCallback, state)
    Catch e As Exception
        Console.WriteLine(e.ToString())
    End Try
End Sub 'Receive
```

```
private static void Receive(Socket client) {
    try {
        // Create the state object.
        StateObject state = new StateObject();
        state.workSocket = client;

        // Begin receiving the data from the remote device.
        client.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
            new AsyncCallback(ReceiveCallback), state);
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

接收回呼方法 `ReceiveCallback` 會實作 AsyncCallback 委派。它會從網路裝置接收資料，並建立訊息字串。它會將來自網路的一或多個位元組資料讀取到資料緩衝區，然後重新呼叫 BeginReceive 方法，直到用戶端所傳送的資料完整為止。從用戶端讀取所有資料之後，`ReceiveCallback` 會設定 ManualResetEvent `sendDone` 通知應用程式執行緒資料已完整。

下列範例程式碼會實作 `ReceiveCallback` 方法。它會假設名為 `response` 的全域字串負責保存收到的字串，以及一個名為 `receiveDone` 的全域 ManualResetEvent。伺服器必須先依正常程序關閉用戶端通訊端，結束網路工作階段。

```
Private Shared Sub ReceiveCallback(ar As IAsyncResult)
    Try
        ' Retrieve the state object and the client socket
        ' from the asynchronous state object.
        Dim state As StateObject = CType(ar.AsyncState, StateObject)
        Dim client As Socket = state.workSocket

        ' Read data from the remote device.
        Dim bytesRead As Integer = client.EndReceive(ar)

        If bytesRead > 0 Then
            ' There might be more data, so store the data received so far.
            state.sb.Append(Encoding.ASCII.GetString(state.buffer, 0, _
                bytesRead))

            '  Get the rest of the data.
            client.BeginReceive(state.buffer, 0, state.BufferSize, 0, _
                AddressOf ReceiveCallback, state)
        Else
            ' All the data has arrived; put it in response.
            If state.sb.Length > 1 Then
                response = state.sb.ToString()
            End If
            ' Signal that all bytes have been received.
            receiveDone.Set()
        End If
    Catch e As Exception
        Console.WriteLine(e.ToString())
    End Try
End Sub 'ReceiveCallback
```

```
private static void ReceiveCallback( IAsyncResult ar ) {
    try {
        // Retrieve the state object and the client socket
        // from the asynchronous state object.
        StateObject state = (StateObject) ar.AsyncState;
        Socket client = state.workSocket;
        // Read data from the remote device.
        int bytesRead = client.EndReceive(ar);
        if (bytesRead > 0) {
            // There might be more data, so store the data received so far.
            state.sb.Append(Encoding.ASCII.GetString(state.buffer,0,bytesRead));
                //  Get the rest of the data.
            client.BeginReceive(state.buffer,0,StateObject.BufferSize,0,
                new AsyncCallback(ReceiveCallback), state);
        } else {
            // All the data has arrived; put it in response.
            if (state.sb.Length > 1) {
                response = state.sb.ToString();
            }
            // Signal that all bytes have been received.
            receiveDone.Set();
        }
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

另請參閱

- 使用同步用戶端通訊端
- 透過通訊端接聽
- 非同步用戶端通訊端範例

透過通訊端接聽

2020/3/20 • • Edit Online

接聽程式或伺服器通訊端會開啟網路的連接埠，等候用戶端連接到該連接埠。雖然有其他的網路位址系列和通訊協定存在，但此範例會示範如何建立遠端服務的 TCP/IP 網路連線。

定義 TCP/IP 服務唯一位址的方式，是結合主機的 IP 位址與服務的連接埠號碼，建立服務的端點。Dns 類別提供的方法，會傳回區域網路裝置支援的網路位址相關資訊。當區域網路裝置有多個網路位址，或本機系統支援多部網路裝置時，Dns 類別會傳回所有網路位址的相關資訊，而應用程式必須選擇適當的服務位址。互聯網分配號碼管理局（Iana）為公共服務定義埠號;有關詳細資訊，請參閱服務名稱和傳輸協議埠號註冊表。其他服務的登錄連接埠號碼範圍可以是 1,024 到 65,535。

下列範例會結合主機電腦 Dns 傳回的第一個 IP 位址，和從已登錄連接埠號碼範圍中選擇的連接埠號碼，為伺服器建立 IPEndPoint。

```
Dim ipHostInfo As IPHostEntry = Dns.GetHostEntry(Dns.GetHostName())
Dim ipAddress As IPAddress = ipHostInfo.AddressList(0)
Dim localEndPoint As New IPEndPoint(ipAddress, 11000)
```

```
IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
IPAddress ipAddress = ipHostInfo.AddressList[0];
IPEndPoint localEndPoint = new IPEndPoint(ipAddress, 11000);
```

決定本機端點之後，Socket 必須使用 Bind 方法建立與該端點的關聯，並使用 Listen 方法設定接聽端點。如已使用特定位址和連接埠的組合，Bind 會擲回例外狀況。下列範例會示範建立**通訊端**與 IPEndPoint 的關聯。

```
Dim listener As New Socket(ipAddress.AddressFamily, _
    SocketType.Stream, ProtocolType.Tcp)
listener.Bind(localEndPoint)
listener.Listen(100)
```

```
Socket listener = new Socket(ipAddress.AddressFamily,
    SocketType.Stream, ProtocolType.Tcp);
listener.Bind(localEndPoint);
listener.Listen(100);
```

Listen 方法會採用單一參數，指定在伺服器忙碌錯誤傳回至連線的用戶端之前，允許的**通訊端**暫止連線數目。在本例中，在伺服器忙碌回應傳回至用戶端編號 101 之前，連線佇列中最多放置 100 個用戶端。

另請參閱

- 使用同步伺服器通訊端
- 使用非同步伺服器通訊端
- 使用用戶端通訊端
- 如何：建立通訊端
- 通訊端

使用同步伺服器通訊端

2020/3/20 • • Edit Online

同步伺服器通訊端會暫停應用程式執行，直到在通訊端上收到連線要求為止。同步伺服器通訊端不適用於大量使用網路以進行作業的應用程式，但它們可能適合簡單網路應用程式。

使用 Bind 和 Listen 方法設定 Socket 以接聽端點之後，便已準備好使用 Accept 方法接受連入的連線要求。應用程式會暫停，直到呼叫 Accept 方法收到連線要求為止。

收到連線要求時，Accept 會傳回與連線用戶端建立關聯的新 Socket 執行個體。下列範例會從用戶端讀取資料、將它顯示在主控台中，然後將資料回應傳回給用戶端。Socket 未指定任何傳訊通訊協定，因此字串 "<EOF>" 會標記訊息資料的結束。它假設名為 `listener` 的**通訊端**已初始化並繫結至端點。

```
Console.WriteLine("Waiting for a connection...")
Dim handler As Socket = listener.Accept()
Dim data As String = Nothing

While True
    bytes = New Byte(1024) {}
    Dim bytesRec As Integer = handler.Receive(bytes)
    data += Encoding.ASCII.GetString(bytes, 0, bytesRec)
    If data.IndexOf("<EOF>") > - 1 Then
        Exit While
    End If
End While

Console.WriteLine("Text received : {0}", data)

Dim msg As Byte() = Encoding.ASCII.GetBytes(data)
handler.Send(msg)
handler.Shutdown(SocketShutdown.Both)
handler.Close()
```

```
Console.WriteLine("Waiting for a connection...");
Socket handler = listener.Accept();
String data = null;

while (true) {
    bytes = new byte[1024];
    int bytesRec = handler.Receive(bytes);
    data += Encoding.ASCII.GetString(bytes,0,bytesRec);
    if (data.IndexOf("<EOF>") > -1) {
        break;
    }
}

Console.WriteLine( "Text received : {0}", data);

byte[] msg = Encoding.ASCII.GetBytes(data);
handler.Send(msg);
handler.Shutdown(SocketShutdown.Both);
handler.Close();
```

另請參閱

- 使用非同步伺服器通訊端

- 同步伺服器通訊端範例
- 透過通訊端接聽

使用非同步伺服器通訊端

2020/3/20 • • Edit Online

非同步伺服器通訊端會使用 .NET Framework 非同步程式設計模型來處理網路服務要求。Socket 類別會遵循標準 .NET Framework 非同步命名模式；例如，同步 Accept 方法對應於非同步 BeginAccept 和 EndAccept 方法。

非同步伺服器通訊端需要一個用來開始接受網路連線要求的方法、一個用來處理連線要求並開始接收網路資料的回呼方法，以及一個用來結束接收資料的回呼方法。本節會進一步討論所有這些方法。

在下列範例中，若要開始接受網路的連線要求，`StartListening` 方法會初始化**通訊端**，然後使用 BeginAccept 方法來開始接受新連線。在通訊端上收到新的連線要求時，就會呼叫接受回呼方法。它會負責取得將處理連線的**通訊端**執行個體，並將該**通訊端**移交給將處理要求的執行緒。接受回呼方法會實作 AsyncCallback 委派；它會傳回 void，並採用 IAsyncResult 類型的單一參數。下列範例是接受回呼方法的殼層。

```
Sub AcceptCallback(ar As IAsyncResult)
    ' Add the callback code here.
End Sub 'AcceptCallback
```

```
void AcceptCallback(IAsyncResult ar)
{
    // Add the callback code here.
}
```

BeginAccept 方法採用兩個參數：指向接受回呼方法的 AsyncCallback 委派和用來將狀態資訊傳遞至回呼方法的物件。在下列範例中，接聽**通訊端**會透過 *stat* 參數傳遞至回呼方法。這個範例會建立 AsyncCallback 委派並開始接受來自網路的連線。

```
listener.BeginAccept( _
    New AsyncCallback(SocketListener.AcceptCallback),_
    listener)
```

```
listener.BeginAccept(new AsyncCallback(SocketListener.AcceptCallback), listener);
```

非同步通訊端使用系統執行緒集區中的執行緒來處理連入連線。其中一個執行緒負責接受連線、另一個執行緒用來處理每個連入連線，還有一個執行緒負責接收來自連線的資料。這些可能是相同的執行緒，視執行緒集區所指派的執行緒而定。在下列範例中，System.Threading.ManualResetEvent 類別會暫停執行主執行緒，並在可以繼續執行時發出訊號。

下列範例示範非同步方法，用來在本機電腦上建立非同步的 TCP/IP 通訊端，並開始接受連接。它假設有一個名為 `allDone` 的全域 **ManualResetEvent**且該方法是 `SocketListener` 類別的成員，以及已定義名為 `AcceptCallback` 的回呼方法。

```
Public Sub StartListening()
    Dim ipHostInfo As IPHostEntry = Dns.Resolve(Dns.GetHostName())
    Dim localEP = New IPEndPoint(ipHostInfo.AddressList(0), 11000)

    Console.WriteLine($"Local address and port : {localEP.ToString()}")

    Dim listener As New Socket(localEP.Address.AddressFamily, _
       SocketType.Stream, ProtocolType.Tcp)

    Try
        listener.Bind(localEP)
        listener.Listen(10)

        While True
            allDone.Reset()

            Console.WriteLine("Waiting for a connection...")
            listener.BeginAccept(New _
                AsyncCallback(SocketListener.AcceptCallback), _
                listener)

            allDone.WaitOne()
        End While
    Catch e As Exception
        Console.WriteLine(e.ToString())
    End Try
    Console.WriteLine("Closing the listener...")
End Sub 'StartListening
```

```
public void StartListening()
{
    IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName());
    IPEndPoint localEP = new IPEndPoint(ipHostInfo.AddressList[0], 11000);

    Console.WriteLine($"Local address and port : {localEP.ToString()}");

    Socket listener = new Socket(localEP.Address.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

    try
    {
        listener.Bind(localEP);
        listener.Listen(10);

        while (true)
        {
            allDone.Reset();

            Console.WriteLine("Waiting for a connection...");
            listener.BeginAccept(new AsyncCallback(SocketListener.AcceptCallback), listener);

            allDone.WaitOne();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }

    Console.WriteLine("Closing the listener...");
}
```

接受回呼方法 (在上述範例中為 `AcceptCallback`) 負責指示主要應用程式執行緒繼續處理、建立與用戶端的連線，以及開始從用戶端非同步讀取資料。下列範例是 `AcceptCallback` 方法實作的第一部分。該方法的這個區段會指示主要應用程式執行緒繼續處理，並建立與用戶端的連線。它會假設一個名為 `allDone` 的全域

ManualResetEvent。

```
Public Sub AcceptCallback(ar As IAsyncResult)
    allDone.Set()

    Dim listener As Socket = CType(ar.AsyncState, Socket)
    Dim handler As Socket = listener.EndAccept(ar)

    ' Additional code to read data goes here.
End Sub 'AcceptCallback
```

```
public void AcceptCallback(IAsyncResult ar)
{
    allDone.Set();

    Socket listener = (Socket) ar.AsyncState;
    Socket handler = listener.EndAccept(ar);

    // Additional code to read data goes here.
}
```

從用戶端通訊端讀取資料時，需要在非同步呼叫之間傳遞值的狀態物件。下列範例會實作從遠端用戶端接收字串的狀態物件。其包含用於用戶端通訊端的欄位、用來接收資料的資料緩衝區，以及用來建立用戶端所傳送資料字串的 StringBuilder。將這些欄位放置在狀態物件，可允許在多個呼叫間保留其值，以從用戶端通訊端讀取資料。

```
Public Class StateObject
    Public workSocket As Socket = Nothing
    Public BufferSize As Integer = 1024
    Public buffer(BufferSize) As Byte
    Public sb As New StringBuilder()
End Class 'StateObject
```

```
public class StateObject
{
    public Socket workSocket = null;
    public const int BufferSize = 1024;
    public byte[] buffer = new byte[BufferSize];
    public StringBuilder sb = new StringBuilder();
}
```

開始從用戶端通訊端接收資料的 `AcceptCallback` 方法的個區段首先會初始化 `StateObject` 類別的執行個體，然後呼叫 BeginReceive 方法，開始以非同步方式從用戶端通訊端讀取資料。

下列範例示範完整的 `AcceptCallback` 方法。它假設有一個名為 `allDone`，的全域 ManualResetEvent，且其 `StateObject` 類別已定義，以及已在名為 `SocketListener` 的類別中定義 `ReadCallback` 方法。

```
Public Shared Sub AcceptCallback(ar As IAsyncResult)
    ' Get the socket that handles the client request.
    Dim listener As Socket = CType(ar.AsyncState, Socket)
    Dim handler As Socket = listener.EndAccept(ar)

    ' Signal the main thread to continue.
    allDone.Set()

    ' Create the state object.
    Dim state As New StateObject()
    state.workSocket = handler
    handler.BeginReceive(state.buffer, 0, state.BufferSize, 0, _
        AddressOf AsynchronousSocketListener.ReadCallback, state)
End Sub 'AcceptCallback
```

```
public static void AcceptCallback(IAsyncResult ar)
{
    // Get the socket that handles the client request.
    Socket listener = (Socket) ar.AsyncState;
    Socket handler = listener.EndAccept(ar);

    // Signal the main thread to continue.
    allDone.Set();

    // Create the state object.
    StateObject state = new StateObject();
    state.workSocket = handler;
    handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
        new AsyncCallback(AsynchronousSocketListener.ReadCallback), state);
}
```

非同步通訊端伺服器必須實作的最後一個方法是讀取回呼方法，以傳回用戶端所傳送的資料。如同接受回呼方法一樣，讀取回呼方法是 AsyncCallback 委派。這個方法會從用戶端通訊端讀取一或多個位元組到資料緩衝區，然後重新呼叫 BeginReceive 方法，直到用戶端所傳送的資料完整為止。從用戶端讀取整個訊息後，字串就會顯示在主控台上，並關閉處理用戶端連線的伺服器通訊端。

下列範例會實作 `ReadCallback` 方法。它假設已定義了 `StateObject` 類別。

```
Public Shared Sub ReadCallback(ar As IAsyncResult)
    Dim state As StateObject = CType(ar.AsyncState, StateObject)
    Dim handler As Socket = state.workSocket

    ' Read data from the client socket.
    Dim read As Integer = handler.EndReceive(ar)

    ' Data was read from the client socket.
    If read > 0 Then
        state.sb.Append(Encoding.ASCII.GetString(state.buffer, 0, read))
        handler.BeginReceive(state.buffer, 0, state.BufferSize, 0, _
            AddressOf ReadCallback, state)
    Else
        If state.sb.Length > 1 Then
            ' All the data has been read from the client;
            ' display it on the console.
            Dim content As String = state.sb.ToString()
            Console.WriteLine($"Read {content.Length} bytes from socket. {ControlChars.Cr} Data : {content}")
        End If
    End If
End Sub 'ReadCallback
```

```
public static void ReadCallback(IAsyncResult ar)
{
    StateObject state = (StateObject) ar.AsyncState;
    Socket handler = state.WorkSocket;

    // Read data from the client socket.
    int read = handler.EndReceive(ar);

    // Data was read from the client socket.
    if (read > 0)
    {
        state.sb.Append(Encoding.ASCII.GetString(state.buffer,0,read));
        handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
            new AsyncCallback(ReadCallback), state);
    }
    else
    {
        if (state.sb.Length > 1)
        {
            // All the data has been read from the client;
            // display it on the console.
            string content = state.sb.ToString();
            Console.WriteLine($"Read {content.Length} bytes from socket.\n Data : {content}");
        }
        handler.Close();
    }
}
```

另請參閱

- 使用同步伺服器通訊端
- 非同步伺服器通訊端範例
- 執行緒處理
- 透過通訊端接聽

通訊端程式碼範例

2020/3/20 • • Edit Online

下列程式碼範例示範如何使用 Socket 類別作為用戶端連接到遠端網路服務，以及作為伺服器接聽來自遠端用戶端的連線。

本節內容

同步用戶端通訊端範例
示範如何實作同步的 Socket 用戶端，連接到伺服器，並顯示從伺服器傳回的資料。

同步伺服器通訊端範例
示範如何實作同步的 Socket 伺服器，接受來自用戶端的連線，並回應從用戶端收到的資料。

非同步用戶端通訊端範例
示範如何實作非同步的 Socket 用戶端，連接到伺服器，並顯示從伺服器傳回的資料。

非同步伺服器通訊端範例
示範如何實作非同步的 Socket 伺服器，接受來自用戶端的連線，並回應從用戶端收到的資料。

相關章節

通訊端
提供有關 System.Net.Sockets 命名空間和 Socket 類別的基本資訊。

網路程式設計的安全性
描述如何使用標準網際網路安全性和驗證技術。

同步用戶端通訊端範例

2020/3/20 • • Edit Online

下列範例程式會建立連線到伺服器的用戶端。伺服器已內建非同步通訊端，因此在伺服器傳回回應之前，會暫停執行用戶端應用程式。應用程式會將字串傳送到伺服器，然後在主控台上顯示伺服器所傳回的字串。

```
Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.Text

Public Class SynchronousSocketClient

    Public Shared Sub Main()
        ' Data buffer for incoming data.
        Dim bytes(1024) As Byte

        ' Connect to a remote device.

        ' Establish the remote endpoint for the socket.
        ' This example uses port 11000 on the local computer.
        Dim ipHostInfo As IPHostEntry = Dns.GetHostEntry(Dns.GetHostName())
        Dim ipAddress As IPAddress = ipHostInfo.AddressList(0)
        Dim remoteEP As New IPEndPoint(ipAddress, 11000)

        ' Create a TCP/IP socket.
        Dim sender As New Socket(ipAddress.AddressFamily, _
            SocketType.Stream, ProtocolType.Tcp)

        ' Connect the socket to the remote endpoint.
        sender.Connect(remoteEP)

        Console.WriteLine("Socket connected to {0}", _
            sender.RemoteEndPoint.ToString())

        ' Encode the data string into a byte array.
        Dim msg As Byte() = _
            Encoding.ASCII.GetBytes("This is a test<EOF>")

        ' Send the data through the socket.
        Dim bytesSent As Integer = sender.Send(msg)

        ' Receive the response from the remote device.
        Dim bytesRec As Integer = sender.Receive(bytes)
        Console.WriteLine("Echoed test = {0}", _
            Encoding.ASCII.GetString(bytes, 0, bytesRec))

        ' Release the socket.
        sender.Shutdown(SocketShutdown.Both)
        sender.Close()
    End Sub

End Class 'SynchronousSocketClient
```

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class SynchronousSocketClient {

    public static void StartClient() {
        // Data buffer for incoming data.
        byte[] bytes = new byte[1024];

        // Connect to a remote device.
        try {
            // Establish the remote endpoint for the socket.
            // This example uses port 11000 on the local computer.
            IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
            IPAddress ipAddress = ipHostInfo.AddressList[0];
            IPEndPoint remoteEP = new IPEndPoint(ipAddress,11000);

            // Create a TCP/IP  socket.
            Socket sender = new Socket(ipAddress.AddressFamily,
                SocketType.Stream, ProtocolType.Tcp );

            // Connect the socket to the remote endpoint. Catch any errors.
            try {
                sender.Connect(remoteEP);

                Console.WriteLine("Socket connected to {0}",
                    sender.RemoteEndPoint.ToString());

                // Encode the data string into a byte array.
                byte[] msg = Encoding.ASCII.GetBytes("This is a test<EOF>");

                // Send the data through the socket.
                int bytesSent = sender.Send(msg);

                // Receive the response from the remote device.
                int bytesRec = sender.Receive(bytes);
                Console.WriteLine("Echoed test = {0}",
                    Encoding.ASCII.GetString(bytes,0,bytesRec));

                // Release the socket.
                sender.Shutdown(SocketShutdown.Both);
                sender.Close();

            } catch (ArgumentNullException ane) {
                Console.WriteLine("ArgumentNullException : {0}",ane.ToString());
            } catch (SocketException se) {
                Console.WriteLine("SocketException : {0}",se.ToString());
            } catch (Exception e) {
                Console.WriteLine("Unexpected exception : {0}", e.ToString());
            }

        } catch (Exception e) {
            Console.WriteLine( e.ToString());
        }
    }

    public static int Main(String[] args) {
        StartClient();
        return 0;
    }
}
```

另請參閱

- 同步伺服器通訊端範例
- 使用同步用戶端通訊端
- 通訊端程式碼範例

同步伺服器通訊端範例

2020/3/20 • • Edit Online

下列範例程式會建立從用戶端接收連線要求的伺服器。伺服器已內建非同步通訊端，因此在其等候來自用戶端的連接時，會暫停執行伺服器應用程式。應用程式會從用戶端收到一個字串，在主控台中顯示字串，然後將字串回應回用戶端。用戶端的字串必須包含字串 "<EOF>" 來表示訊息結束。

```
Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.Text
Imports Microsoft.VisualBasic

Public Class SynchronousSocketListener

    ' Incoming data from the client.
    Public Shared data As String = Nothing

    Public Shared Sub Main()
        ' Data buffer for incoming data.
        Dim bytes() As Byte = New [Byte](1024) {}

        ' Establish the local endpoint for the socket.
        ' Dns.GetHostName returns the name of the
        ' host running the application.
        Dim ipHostInfo As IPHostEntry = Dns.GetHostEntry(Dns.GetHostName())
        Dim ipAddress As IPAddress = ipHostInfo.AddressList(0)
        Dim localEndPoint As New IPEndPoint(ipAddress, 11000)

        ' Create a TCP/IP socket.
        Dim listener As New Socket(ipAddress.AddressFamily, _
            SocketType.Stream, ProtocolType.Tcp)

        ' Bind the socket to the local endpoint and
        ' listen for incoming connections.

        listener.Bind(localEndPoint)
        listener.Listen(10)

        ' Start listening for connections.
        While True
            Console.WriteLine("Waiting for a connection...")
            ' Program is suspended while waiting for an incoming connection.
            Dim handler As Socket = listener.Accept()
            data = Nothing

            ' An incoming connection needs to be processed.
            While True
                Dim bytesRec As Integer = handler.Receive(bytes)
                data += Encoding.ASCII.GetString(bytes, 0, bytesRec)
                If data.IndexOf("<EOF>") > -1 Then
                    Exit While
                End If
            End While
            ' Show the data on the console.
            Console.WriteLine("Text received : {0}", data)
            ' Echo the data back to the client.
            Dim msg As Byte() = Encoding.ASCII.GetBytes(data)
            handler.Send(msg)
            handler.Shutdown(SocketShutdown.Both)
            handler.Close()
        End While
    End Sub

End Class 'SynchronousSocketListener
```

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class SynchronousSocketListener {
```

```
    // Incoming data from the client.
    public static string data = null;

    public static void StartListening() {
        // Data buffer for incoming data.
        byte[] bytes = new Byte[1024];

        // Establish the local endpoint for the socket.
        // Dns.GetHostName returns the name of the
        // host running the application.
        IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
        IPAddress ipAddress = ipHostInfo.AddressList[0];
        IPEndPoint localEndPoint = new IPEndPoint(ipAddress, 11000);

        // Create a TCP/IP socket.
        Socket listener = new Socket(ipAddress.AddressFamily,
            SocketType.Stream, ProtocolType.Tcp );

        // Bind the socket to the local endpoint and
        // listen for incoming connections.
        try {
            listener.Bind(localEndPoint);
            listener.Listen(10);

            // Start listening for connections.
            while (true) {
                Console.WriteLine("Waiting for a connection...");
                // Program is suspended while waiting for an incoming connection.
                Socket handler = listener.Accept();
                data = null;

                // An incoming connection needs to be processed.
                while (true) {
                    int bytesRec = handler.Receive(bytes);
                    data += Encoding.ASCII.GetString(bytes,0,bytesRec);
                    if (data.IndexOf("<EOF>") > -1) {
                        break;
                    }
                }

                // Show the data on the console.
                Console.WriteLine( "Text received : {0}", data);

                // Echo the data back to the client.
                byte[] msg = Encoding.ASCII.GetBytes(data);

                handler.Send(msg);
                handler.Shutdown(SocketShutdown.Both);
                handler.Close();
            }

        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }

        Console.WriteLine("\nPress ENTER to continue...");
        Console.Read();

    }

    public static int Main(String[] args) {
        StartListening();
        return 0;
    }
}
```

另請參閱

- 同步用戶端通訊端範例
- 使用同步伺服器通訊端
- 通訊端程式碼範例

非同步用戶端通訊端範例

2020/3/20 • • Edit Online

下列範例程式會建立連線到伺服器的用戶端。伺服器已內建非同步通訊端，因此在伺服器傳回回應時，不會暫停執行用戶端應用程式。應用程式會將字串傳送到伺服器，然後在主控台上顯示伺服器所傳回的字串。

```
Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.Threading
Imports System.Text

' State object for receiving data from remote device.

Public Class StateObject
    ' Client socket.
    Public workSocket As Socket = Nothing
    ' Size of receive buffer.
    Public Const BufferSize As Integer = 256
    ' Receive buffer.
    Public buffer(BufferSize) As Byte
    ' Received data string.
    Public sb As New StringBuilder
End Class 'StateObject

Public Class AsynchronousClient
    ' The port number for the remote device.
    Private Const port As Integer = 11000

    ' ManualResetEvent instances signal completion.
    Private Shared connectDone As New ManualResetEvent(False)
    Private Shared sendDone As New ManualResetEvent(False)
    Private Shared receiveDone As New ManualResetEvent(False)

    ' The response from the remote device.
    Private Shared response As String = String.Empty

    Public Shared Sub Main()
        ' Establish the remote endpoint for the socket.
        ' For this example use local machine.
        Dim ipHostInfo As IPHostEntry = Dns.GetHostEntry(Dns.GetHostName())
        Dim ipAddress As IPAddress = ipHostInfo.AddressList(0)
        Dim remoteEP As New IPEndPoint(ipAddress, port)

        ' Create a TCP/IP socket.
        Dim client As New Socket(ipAddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp)

        ' Connect to the remote endpoint.
        client.BeginConnect(remoteEP, New AsyncCallback(AddressOf ConnectCallback), client)

        ' Wait for connect.
        connectDone.WaitOne()

        ' Send test data to the remote device.
        Send(client, "This is a test<EOF>")
        sendDone.WaitOne()

        ' Receive the response from the remote device.
        Receive(client)
        receiveDone.WaitOne()

        ' Write the response to the console.
```

```
        Console.WriteLine("Response received : {0}", response)

        ' Release the socket.
        client.Shutdown(SocketShutdown.Both)
        client.Close()
    End Sub 'Main

    Private Shared Sub ConnectCallback(ByVal ar As IAsyncResult)
        ' Retrieve the socket from the state object.
        Dim client As Socket = CType(ar.AsyncState, Socket)

        ' Complete the connection.
        client.EndConnect(ar)

        Console.WriteLine("Socket connected to {0}", client.RemoteEndPoint.ToString())

        ' Signal that the connection has been made.
        connectDone.Set()
    End Sub 'ConnectCallback

    Private Shared Sub Receive(ByVal client As Socket)

        ' Create the state object.
        Dim state As New StateObject
        state.workSocket = client

        ' Begin receiving the data from the remote device.
        client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0, New AsyncCallback(AddressOf
ReceiveCallback), state)
    End Sub 'Receive

    Private Shared Sub ReceiveCallback(ByVal ar As IAsyncResult)

        ' Retrieve the state object and the client socket
        ' from the asynchronous state object.
        Dim state As StateObject = CType(ar.AsyncState, StateObject)
        Dim client As Socket = state.workSocket

        ' Read data from the remote device.
        Dim bytesRead As Integer = client.EndReceive(ar)

        If bytesRead > 0 Then
            ' There might be more data, so store the data received so far.
            state.sb.Append(Encoding.ASCII.GetString(state.buffer, 0, bytesRead))

            ' Get the rest of the data.
            client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0, New AsyncCallback(AddressOf
ReceiveCallback), state)
        Else
            ' All the data has arrived; put it in response.
            If state.sb.Length > 1 Then
                response = state.sb.ToString()
            End If
            ' Signal that all bytes have been received.
            receiveDone.Set()
        End If
    End Sub 'ReceiveCallback

    Private Shared Sub Send(ByVal client As Socket, ByVal data As String)
        ' Convert the string data to byte data using ASCII encoding.
        Dim byteData As Byte() = Encoding.ASCII.GetBytes(data)

        ' Begin sending the data to the remote device.
        client.BeginSend(byteData, 0, byteData.Length, 0, New AsyncCallback(AddressOf SendCallback), client)
    End Sub 'Send

    Private Shared Sub SendCallback(ByVal ar As IAsyncResult)
        ' Retrieve the socket from the state object.
        Dim client As Socket = CType(ar.AsyncState, Socket)
```

```
        Dim client As Socket = CType(ar.AsyncState, Socket)

        ' Complete sending the data to the remote device.
        Dim bytesSent As Integer = client.EndSend(ar)
        Console.WriteLine("Sent {0} bytes to server.", bytesSent)

        ' Signal that all bytes have been sent.
        sendDone.Set()
    End Sub 'SendCallback
End Class 'AsynchronousClient
```

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Text;

// State object for receiving data from remote device.
public class StateObject {
    // Client socket.
    public Socket workSocket = null;
    // Size of receive buffer.
    public const int BufferSize = 256;
    // Receive buffer.
    public byte[] buffer = new byte[BufferSize];
    // Received data string.
    public StringBuilder sb = new StringBuilder();
}

public class AsynchronousClient {
    // The port number for the remote device.
    private const int port = 11000;

    // ManualResetEvent instances signal completion.
    private static ManualResetEvent connectDone =
        new ManualResetEvent(false);
    private static ManualResetEvent sendDone =
        new ManualResetEvent(false);
    private static ManualResetEvent receiveDone =
        new ManualResetEvent(false);

    // The response from the remote device.
    private static String response = String.Empty;

    private static void StartClient() {
        // Connect to a remote device.
        try {
            // Establish the remote endpoint for the socket.
            // The name of the
            // remote device is "host.contoso.com".
            IPHostEntry ipHostInfo = Dns.GetHostEntry("host.contoso.com");
            IPAddress ipAddress = ipHostInfo.AddressList[0];
            IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);

            // Create a TCP/IP socket.
            Socket client = new Socket(ipAddress.AddressFamily,
                SocketType.Stream, ProtocolType.Tcp);

            // Connect to the remote endpoint.
            client.BeginConnect( remoteEP,
                new AsyncCallback(ConnectCallback), client);
            connectDone.WaitOne();

            // Send test data to the remote device.
            Send(client,"This is a test<EOF>");
            sendDone.WaitOne();

            // Receive the response from the remote device.
```

```
            Receive(client);
            receiveDone.WaitOne();

            // Write the response to the console.
            Console.WriteLine("Response received : {0}", response);

            // Release the socket.
            client.Shutdown(SocketShutdown.Both);
            client.Close();

        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
    }

    private static void ConnectCallback(IAsyncResult ar) {
        try {
            // Retrieve the socket from the state object.
            Socket client = (Socket) ar.AsyncState;

            // Complete the connection.
            client.EndConnect(ar);

            Console.WriteLine("Socket connected to {0}",
                client.RemoteEndPoint.ToString());

            // Signal that the connection has been made.
            connectDone.Set();
        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
    }

    private static void Receive(Socket client) {
        try {
            // Create the state object.
            StateObject state = new StateObject();
            state.workSocket = client;

            // Begin receiving the data from the remote device.
            client.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
                new AsyncCallback(ReceiveCallback), state);
        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
    }

    private static void ReceiveCallback( IAsyncResult ar ) {
        try {
            // Retrieve the state object and the client socket
            // from the asynchronous state object.
            StateObject state = (StateObject) ar.AsyncState;
            Socket client = state.workSocket;

            // Read data from the remote device.
            int bytesRead = client.EndReceive(ar);

            if (bytesRead > 0) {
                // There might be more data, so store the data received so far.
            state.sb.Append(Encoding.ASCII.GetString(state.buffer,0,bytesRead));

                // Get the rest of the data.
                client.BeginReceive(state.buffer,0,StateObject.BufferSize,0,
                    new AsyncCallback(ReceiveCallback), state);
            } else {
                // All the data has arrived; put it in response.
                if (state.sb.Length > 1) {
                    response = state.sb.ToString();
                }
```

```
                // Signal that all bytes have been received.
                receiveDone.Set();
            }
        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
    }

    private static void Send(Socket client, String data) {
        // Convert the string data to byte data using ASCII encoding.
        byte[] byteData = Encoding.ASCII.GetBytes(data);

        // Begin sending the data to the remote device.
        client.BeginSend(byteData, 0, byteData.Length, 0,
            new AsyncCallback(SendCallback), client);
    }

    private static void SendCallback(IAsyncResult ar) {
        try {
            // Retrieve the socket from the state object.
            Socket client = (Socket) ar.AsyncState;

            // Complete sending the data to the remote device.
            int bytesSent = client.EndSend(ar);
            Console.WriteLine("Sent {0} bytes to server.", bytesSent);

            // Signal that all bytes have been sent.
            sendDone.Set();
        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
    }

    public static int Main(String[] args) {
        StartClient();
        return 0;
    }
}
```

另請參閱

- 非同步伺服器通訊端範例
- 使用同步伺服器通訊端
- 通訊端程式碼範例

非同步伺服器通訊端範例

2020/3/20 • • Edit Online

下列範例程式會建立從用戶端接收連線要求的伺服器。伺服器已內建非同步通訊端，因此在其等候來自用戶端的連接時，不會暫停執行伺服器應用程式。應用程式會從用戶端收到一個字串，在主控台中顯示字串，然後將字串回應回用戶端。用戶端的字串必須包含字串 "<EOF>" 來表示訊息結束。

```
Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.Text
Imports System.Threading
Imports Microsoft.VisualBasic

' State object for reading client data asynchronously

Public Class StateObject
    ' Client  socket.
    Public workSocket As Socket = Nothing
    ' Size of receive buffer.
    Public Const BufferSize As Integer = 1024
    ' Receive buffer.
    Public buffer(BufferSize) As Byte
    ' Received data string.
    Public sb As New StringBuilder
End Class 'StateObject

Public Class AsynchronousSocketListener
    ' Thread signal.
    Public Shared allDone As New ManualResetEvent(False)

    ' This server waits for a connection and then uses  asychronous operations to
    ' accept the connection, get data from the connected client,
    ' echo that data back to the connected client.
    ' It then disconnects from the client and waits for another client.
    Public Shared Sub Main()
        ' Establish the local endpoint for the socket.
        Dim ipHostInfo As IPHostEntry = Dns.GetHostEntry(Dns.GetHostName())
        Dim ipAddress As IPAddress = ipHostInfo.AddressList(0)
        Dim localEndPoint As New IPEndPoint(ipAddress, 11000)

        ' Create a TCP/IP socket.
        Dim listener As New Socket(ipAddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp)

        ' Bind the socket to the local endpoint and listen for incoming connections.
        listener.Bind(localEndPoint)
        listener.Listen(100)

        While True
            ' Set the event to nonsignaled state.
            allDone.Reset()

            ' Start an asynchronous socket to listen for connections.
            Console.WriteLine("Waiting for a connection...")
            listener.BeginAccept(New AsyncCallback(AddressOf AcceptCallback), listener)

            ' Wait until a connection is made and processed before continuing.
            allDone.WaitOne()
        End While
    End Sub 'Main

    Public Shared Sub AcceptCallback(ByVal ar As IAsyncResult)
```

```
    Public Shared Sub AcceptCallback(ByVal ar As IAsyncResult)
        ' Get the socket that handles the client request.
        Dim listener As Socket = CType(ar.AsyncState, Socket)
        ' End the operation.
        Dim handler As Socket = listener.EndAccept(ar)

        ' Create the state object for the async receive.
        Dim state As New StateObject
        state.workSocket = handler
        handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0, New AsyncCallback(AddressOf
ReadCallback), state)
    End Sub 'AcceptCallback

    Public Shared Sub ReadCallback(ByVal ar As IAsyncResult)
        Dim content As String = String.Empty

        ' Retrieve the state object and the handler socket
        ' from the asynchronous state object.
        Dim state As StateObject = CType(ar.AsyncState, StateObject)
        Dim handler As Socket = state.workSocket

        ' Read data from the client socket.
        Dim bytesRead As Integer = handler.EndReceive(ar)

        If bytesRead > 0 Then
            ' There  might be more data, so store the data received so far.
            state.sb.Append(Encoding.ASCII.GetString(state.buffer, 0, bytesRead))

            ' Check for end-of-file tag. If it is not there, read
            ' more data.
            content = state.sb.ToString()
            If content.IndexOf("<EOF>") > -1 Then
                ' All the data has been read from the
                ' client. Display it on the console.
                Console.WriteLine("Read {0} bytes from socket. " + vbLf + " Data : {1}", content.Length,
content)
                ' Echo the data back to the client.
                Send(handler, content)
            Else
                ' Not all data received. Get more.
                handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0, New AsyncCallback(AddressOf
ReadCallback), state)
            End If
        End If
    End Sub 'ReadCallback

    Private Shared Sub Send(ByVal handler As Socket, ByVal data As String)
        ' Convert the string data to byte data using ASCII encoding.
        Dim byteData As Byte() = Encoding.ASCII.GetBytes(data)

        ' Begin sending the data to the remote device.
        handler.BeginSend(byteData, 0, byteData.Length, 0, New AsyncCallback(AddressOf SendCallback), handler)
    End Sub 'Send

    Private Shared Sub SendCallback(ByVal ar As IAsyncResult)
        ' Retrieve the socket from the state object.
        Dim handler As Socket = CType(ar.AsyncState, Socket)

        ' Complete sending the data to the remote device.
        Dim bytesSent As Integer = handler.EndSend(ar)
        Console.WriteLine("Sent {0} bytes to client.", bytesSent)

        handler.Shutdown(SocketShutdown.Both)
        handler.Close()
        ' Signal the main thread to continue.
        allDone.Set()
    End Sub 'SendCallback
End Class 'AsynchronousSocketListener
```

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;

// State object for reading client data asynchronously
public class StateObject {
    // Client  socket.
    public Socket workSocket = null;
    // Size of receive buffer.
    public const int BufferSize = 1024;
    // Receive buffer.
    public byte[] buffer = new byte[BufferSize];
// Received data string.
    public StringBuilder sb = new StringBuilder();
}

public class AsynchronousSocketListener {
    // Thread signal.
    public static ManualResetEvent allDone = new ManualResetEvent(false);

    public AsynchronousSocketListener() {
    }

    public static void StartListening() {
        // Establish the local endpoint for the socket.
        // The DNS name of the computer
        // running the listener is "host.contoso.com".
        IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
        IPAddress ipAddress = ipHostInfo.AddressList[0];
        IPEndPoint localEndPoint = new IPEndPoint(ipAddress, 11000);

        // Create a TCP/IP socket.
        Socket listener = new Socket(ipAddress.AddressFamily,
            SocketType.Stream, ProtocolType.Tcp );

        // Bind the socket to the local endpoint and listen for incoming connections.
        try {
            listener.Bind(localEndPoint);
            listener.Listen(100);

            while (true) {
                // Set the event to nonsignaled state.
                allDone.Reset();

                // Start an asynchronous socket to listen for connections.
                Console.WriteLine("Waiting for a connection...");
                listener.BeginAccept(
                    new AsyncCallback(AcceptCallback),
                    listener );

                // Wait until a connection is made before continuing.
                allDone.WaitOne();
            }

        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }

        Console.WriteLine("\nPress ENTER to continue...");
        Console.Read();

    }

    public static void AcceptCallback(IAsyncResult ar) {
        // Signal the main thread to continue.
```

```
        // Signal the main thread to continue.
        allDone.Set();

        // Get the socket that handles the client request.
        Socket listener = (Socket) ar.AsyncState;
        Socket handler = listener.EndAccept(ar);

        // Create the state object.
        StateObject state = new StateObject();
        state.workSocket = handler;
        handler.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
            new AsyncCallback(ReadCallback), state);
    }

    public static void ReadCallback(IAsyncResult ar) {
        String content = String.Empty;

        // Retrieve the state object and the handler socket
        // from the asynchronous state object.
        StateObject state = (StateObject) ar.AsyncState;
        Socket handler = state.workSocket;

        // Read data from the client socket.
        int bytesRead = handler.EndReceive(ar);

        if (bytesRead > 0) {
            // There  might be more data, so store the data received so far.
            state.sb.Append(Encoding.ASCII.GetString(
                state.buffer, 0, bytesRead));

            // Check for end-of-file tag. If it is not there, read
            // more data.
            content = state.sb.ToString();
            if (content.IndexOf("<EOF>") > -1) {
                // All the data has been read from the
                // client. Display it on the console.
                Console.WriteLine("Read {0} bytes from socket. \n Data : {1}",
                    content.Length, content );
                // Echo the data back to the client.
                Send(handler, content);
            } else {
                // Not all data received. Get more.
                handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
                new AsyncCallback(ReadCallback), state);
            }
        }
    }

    private static void Send(Socket handler, String data) {
        // Convert the string data to byte data using ASCII encoding.
        byte[] byteData = Encoding.ASCII.GetBytes(data);

        // Begin sending the data to the remote device.
        handler.BeginSend(byteData, 0, byteData.Length, 0,
            new AsyncCallback(SendCallback), handler);
    }

    private static void SendCallback(IAsyncResult ar) {
        try {
            // Retrieve the socket from the state object.
            Socket handler = (Socket) ar.AsyncState;

            // Complete sending the data to the remote device.
            int bytesSent = handler.EndSend(ar);
            Console.WriteLine("Sent {0} bytes to client.", bytesSent);

            handler.Shutdown(SocketShutdown.Both);
            handler.Close();

        } catch (Exception e) {
```

```
        } catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
    }

    public static int Main(String[] args) {
        StartListening();
        return 0;
    }
}
```

另請參閱

- 非同步用戶端通訊端範例
- 使用非同步伺服器通訊端
- 通訊端程式碼範例

FTP

2020/3/20 • • Edit Online

.NET Framework 使用 FtpWebRequest 和 FtpWebResponse 類別提供 FTP 通訊協定的完整支援。這些類別衍生自 WebRequest 和 WebResponse。在大部分情況下，WebRequest 和 WebResponse 類別提供提出要求所需的一切功能，但是如果您需要存取以屬性方式公開的 FTP 特定功能，則可以將這些類別的類型轉換為 FtpWebRequest 或 FtpWebResponse。

範例

如需詳細資訊，請參閱下列主題：如何：透過 FTP 下載檔案、如何：透過 FTP 上傳檔案和如何：以 FTP 列出目錄內容。

FTP 和 Proxy

如果 Proxy (透過 Proxy 屬性所指定) 是 HTTP Proxy，則只支援 DownloadFile、ListDirectory 和 ListDirectoryDetails 命令。

另請參閱

- 通過代理訪問互聯網
- 網路程式設計範例
- 使用應用程式通訊協定

如何：透過 FTP 下載檔案

2020/3/20 • • Edit Online

這個範例示範如何從 FTP 伺服器下載檔案。

範例

```
using System;
using System.IO;
using System.Net;

namespace Examples.System.Net
{
    public class WebRequestGetExample
    {
        public static void Main ()
        {
            // Get the object used to communicate with the server.
            FtpWebRequest request = (FtpWebRequest)WebRequest.Create("ftp://www.contoso.com/test.htm");
            request.Method = WebRequestMethods.Ftp.DownloadFile;

            // This example assumes the FTP site uses anonymous logon.
            request.Credentials = new NetworkCredential("anonymous","janeDoe@contoso.com");

            FtpWebResponse response = (FtpWebResponse)request.GetResponse();

            Stream responseStream = response.GetResponseStream();
            StreamReader reader = new StreamReader(responseStream);
            Console.WriteLine(reader.ReadToEnd());

            Console.WriteLine($"Download Complete, status {response.StatusDescription}");

            reader.Close();
            response.Close();
        }
    }
}
```

```
Imports System.IO
Imports System.Net

Namespace Examples.System.Net
    Public Module WebRequestGetExample
        Public Sub Main()
            ' Get the object used to communicate with the server.
            Dim request As FtpWebRequest = CType(WebRequest.Create("ftp://www.contoso.com/test.htm"),
FtpWebRequest)
            request.Method = WebRequestMethods.Ftp.DownloadFile

            ' This example assumes the FTP site uses anonymous logon.
            request.Credentials = New NetworkCredential("anonymous", "janeDoe@contoso.com")

            Dim response As FtpWebResponse = CType(request.GetResponse(), FtpWebResponse)

            Dim responseStream As Stream = response.GetResponseStream()
            Dim reader As StreamReader = New StreamReader(responseStream)
            Console.WriteLine(reader.ReadToEnd())

            Console.WriteLine($"Download Complete, status {response.StatusDescription}")

            reader.Close()
            response.Close()
        End Sub
    End Module
End Namespace
```

如何：透過 FTP 上傳檔案

2020/3/20 • • Edit Online

這個範例示範如何將檔案上傳至 FTP 伺服器。

範例

```
using System;
using System.IO;
using System.Net;
using System.Text;

namespace Examples.System.Net
{
    public class WebRequestGetExample
    {
        public static void Main ()
        {
            // Get the object used to communicate with the server.
            FtpWebRequest request = (FtpWebRequest)WebRequest.Create("ftp://www.contoso.com/test.htm");
            request.Method = WebRequestMethods.Ftp.UploadFile;

            // This example assumes the FTP site uses anonymous logon.
            request.Credentials = new NetworkCredential("anonymous", "janeDoe@contoso.com");

            // Copy the contents of the file to the request stream.
            byte[] fileContents;
            using (StreamReader sourceStream = new StreamReader("testfile.txt"))
            {
                fileContents = Encoding.UTF8.GetBytes(sourceStream.ReadToEnd());
            }

            request.ContentLength = fileContents.Length;

            using (Stream requestStream = request.GetRequestStream())
            {
                requestStream.Write(fileContents, 0, fileContents.Length);
            }

            using (FtpWebResponse response = (FtpWebResponse)request.GetResponse())
            {
                Console.WriteLine($"Upload File Complete, status {response.StatusDescription}");
            }
        }
    }
}
```

```
Imports System.IO
Imports System.Net
Imports System.Text

Namespace Examples.System.Net
    Public Module WebRequestGetExample
        Public Sub Main()
            ' Get the object used to communicate with the server.
            Dim request As FtpWebRequest = CType(WebRequest.Create("ftp://www.contoso.com/test.htm"),
FtpWebRequest)
            request.Method = WebRequestMethods.Ftp.UploadFile

            ' This example assumes the FTP site uses anonymous logon.
            request.Credentials = New NetworkCredential("anonymous", "janeDoe@contoso.com")

            ' Copy the contents of the file to the request stream.
            Dim fileContents As Byte()

            Using sourceStream As StreamReader = New StreamReader("testfile.txt")
                fileContents = Encoding.UTF8.GetBytes(sourceStream.ReadToEnd())
            End Using

            request.ContentLength = fileContents.Length

            Using requestStream As Stream = request.GetRequestStream()
                requestStream.Write(fileContents, 0, fileContents.Length)
            End Using

            Using response As FtpWebResponse = CType(request.GetResponse(), FtpWebResponse)
                Console.WriteLine($"Upload File Complete, status {response.StatusDescription}")
            End Using
        End Sub
    End Module
End Namespace
```

如何：以 FTP 列出目錄內容

2020/3/20 • • Edit Online

這個範例示範如何列出 FTP 伺服器的目錄內容。

範例

```
using System;
using System.IO;
using System.Net;

namespace Examples.System.Net
{
    public class WebRequestGetExample
    {
        public static void Main ()
        {
            // Get the object used to communicate with the server.
            FtpWebRequest request = (FtpWebRequest)WebRequest.Create("ftp://www.contoso.com/");
            request.Method = WebRequestMethods.Ftp.ListDirectoryDetails;

            // This example assumes the FTP site uses anonymous logon.
            request.Credentials = new NetworkCredential ("anonymous","janeDoe@contoso.com");

            FtpWebResponse response = (FtpWebResponse)request.GetResponse();

            Stream responseStream = response.GetResponseStream();
            StreamReader reader = new StreamReader(responseStream);
            Console.WriteLine(reader.ReadToEnd());

            Console.WriteLine($"Directory List Complete, status {response.StatusDescription}");

            reader.Close();
            response.Close();
        }
    }
}
```

```
Imports System.IO
Imports System.Net

Namespace Examples.System.Net
    Public Module WebRequestGetExample
        Public Sub Main()
            ' Get the object used to communicate with the server.
            Dim request As FtpWebRequest = CType(WebRequest.Create("ftp://www.contoso.com/"), FtpWebRequest)
            request.Method = WebRequestMethods.Ftp.ListDirectoryDetails

            ' This example assumes the FTP site uses anonymous logon.
            request.Credentials = New NetworkCredential("anonymous", "janeDoe@contoso.com")

            Dim response As FtpWebResponse = CType(request.GetResponse(), FtpWebResponse)

            Dim responseStream As Stream = response.GetResponseStream()
            Dim reader As StreamReader = New StreamReader(responseStream)
            Console.WriteLine(reader.ReadToEnd())

            Console.WriteLine($"Directory List Complete, status {response.StatusDescription}")

            reader.Close()
            response.Close()
        End Sub
    End Module
End Namespace
```

如果您要列出特定的目錄，只需將目錄新增至您在 WebRequest.Create 方法中使用之 URI 的結尾：

```
FtpWebRequest request = (FtpWebRequest)WebRequest.Create("ftp://www.contoso.com/your_preferred_directory");
```

```
Dim request As FtpWebRequest = CType(WebRequest.Create("ftp://www.contoso.com/your_preferred_directory"),
FtpWebRequest)
```

了解 WebRequest 問題和例外狀況

2020/3/20 • • Edit Online

WebRequest 和其衍生的類別 (HttpWebRequest、FtpWebRequest和 FileWebRequest) 會擲回例外狀況，以表示發生異常狀況。有時候這些問題的解決方式並不明顯。

方案

請檢查 WebException 的 Status 屬性來判斷問題。下表顯示數個狀態值和一些可能的解決方式。

‖	‖‖	‖‖
SendFailure -或- ReceiveFailure	基礎通訊端有問題。連接可能已重設。	重新連線，然後重新傳送要求。 確定已安裝最新的 Service Pack。 增加 ServicePointManager.MaxServicePointIdleTime 屬性的值。 將 HttpWebRequest.KeepAlive 設定為 `false`。 使用 DefaultConnectionLimit 屬性增加最大連線數目。 檢查 Proxy 設定。 如果使用 SSL，請確定伺服器處理序有權存取憑證存放區。 如果要傳送大量資料，請將 AllowWriteStreamBuffering 設為 `false`。
TrustFailure	無法驗證伺服器憑證。	嘗試使用 Internet Explorer 開啟 URI。解決 IE 顯示的任何安全性警示。如果您無法解決安全性警示，可以建立憑證原則類別，實作 ICertificatePolicy 以傳回 `true`，並將它傳遞給 CertificatePolicy。 請參閱 https://support.microsoft.com/?id=823177。 請確定簽署伺服器憑證之憑證授權單位的憑證，已新增在 Internet Explorer 的 [信任的憑證授權單位] 清單。 請確定 URL 中的主機名稱符合伺服器憑證上的一般名稱。

[illegible]	[illegible]	[illegible]
SecureChannelFailure	在 SSL 交易中發生錯誤，或有憑證問題。	.NET Framework 1.1 版僅支援 SSL 3.0 版。如果伺服器只使用 TLS 1.0 版或 SSL 2.0 版時，會擲回例外狀況。升級至 .NET Framework 2.0 版，並設定 SecurityProtocol 以符合伺服器。 用戶端憑證是由伺服器不信任的憑證授權單位 (CA) 所簽署。在伺服器上安裝 CA 的憑證。請參閱 <https://support.microsoft.com/?id=332077>。 確定已安裝最新的 Service Pack。
ConnectFailure	連接失敗。	防火牆或 Proxy 封鎖連線。修改防火牆或 Proxy 以允許連線。 通過調用WebProxyWebProxy建構函式 () `WebServiceProxyClass.Proxy = new WebProxy("http://server:80", true)` 顯式指定 用戶端應用程式中的 。 執行 Filemon 或 Regmon，確保工作者處理序身分識別具有存取 WSPWSP.dll、HKLM\System\CurrentControlSet\Services\DnsCache 或 HKLM\System\CurrentControlSet\Services\WinSock2 的必要權限。
NameResolutionFailure	網域名稱服務無法解析主機名稱。	正確設定 Proxy。請參閱 <https://support.microsoft.com/?id=318140>。 確定任何已安裝的防毒軟體或防火牆未封鎖連線。
RequestCanceled	已呼叫 Abort，或發生錯誤。	此問題可能起因於用戶端或伺服器上的負載過重。請降低負載。 增加 DefaultConnectionLimit 設定。 請參閱 https://support.microsoft.com/?id=821268 來修改 Web 服務效能設定。
ConnectionClosed	應用程式嘗試寫入已經關閉的通訊端。	用戶端或伺服器已超載。請降低負載。 增加 DefaultConnectionLimit 設定。 請參閱 https://support.microsoft.com/?id=821268 來修改 Web 服務效能設定。
MessageLengthLimitExceeded	已超過對訊息長度所設定的限制 (MaximumResponseHeadersLength)。	增加 MaximumResponseHeadersLength 屬性的值。

[illegible]	[illegible]	[illegible]
ProxyNameResolutionFailure	網域名稱服務無法解析 Proxy 主機名稱。	正確設定 Proxy。請參閱 <https://support.microsoft.com/?id=318140>。 藉由將 Proxy 屬性設為 `null`，強制 HttpWebRequest 不要使用 Proxy。
ServerProtocolViolation	伺服器的回應不是有效的 HTTP 回應。.NET Framework 偵測到伺服器回應不符合 HTTP 1.1 RFC 時，就會發生這個問題。回應包含不正確的標頭或不正確的標頭分隔符號時，可能會發生這個問題。RFC 2616 定義了 HTTP 1.1 和伺服器回應的有效格式。如需詳細資訊，請參閱網際網路工程任務推動小組 (IETF)網站上的 RFC 2616 - 超文字傳輸通訊協定 -- HTTP/1.1。	取得交易的網路追蹤，並檢查回應中的標頭。 如果您的應用程式需要伺服器回應而不剖析 (這可能是個安全性問題)，請在組態檔中將 `useUnsafeHeaderParsing` 設為 `true`。請參閱<HTTPWeb 請求>元素 (網路設置)。

另請參閱

- HttpWebRequest
- HttpWebResponse
- Dns

網際網路通訊協定第 6 版

2020/3/20 • • Edit Online

網際網路通訊協定第 6 版 (IPv6) 是網際網路網路層級的新標準通訊協定套件。IPv6 旨在解決網際網路通訊協定當前版本 (稱為 IPv4) 有關位址耗竭、安全性、自動組態和擴充性等等的許多問題。IPv6 展開網際網路的功能，以啟用新種類的應用程式，包括點對點與行動應用程式。以下是目前 IPv4 通訊協定的主要問題：

- 快速消耗位址空間。

  這導致使用網路位址轉譯器 (NAT) 將多個私人位址對應到單一公用 IP 位址。這項機制引起的主要問題是處理額外負荷以及缺乏端對端連線。

- 缺少階層式支援。

  因為其固有的預先定義類別組織，IPv4 缺少真正的階層式支援。不可能以真正對應網路拓撲的方式來建立 IP 位址的結構。此一重大的設計缺陷創造了對大型路由表的需求，以將 IPv4 封包傳遞至網際網路上的任何位置。

- 複雜的網路組態。

  使用 IPv4，必須以靜態方式或使用 DHCP 等組態通訊協定指派位址。理想的情況下，主機可能不必依賴 DHCP 基礎結構的管理。相反地，它們可以根據所在的網路區段自行設定。

- 缺少內建的驗證和機密性。

  IPv4 不需要任何提供交換資料驗證或加密之機制的支援。這會隨著 IPv6 變更。網際網路通訊協定安全性 (IPSec) 是 IPv6 支援需求。

新的通訊協定套件必須滿足下列基本需求：

- 低額外負荷的大規模路由和定址。
- 各種連線情況的自動組態。
- 內建的驗證和機密性。

如需詳細資訊，請參閱 IPv6 定址、IPv6 路由、IPv6 自動組態、啟用和停用 IPv6 以及如何：修改電腦組態檔以啟用 IPv6 支援。

參考

以下是您可在網際網路工程任務推動小組 (IETF) 網站中找到的精選 RFC 文件：

- RFC 1287，前進未來網際網路架構。
- RFC 1454，下一版 IP 的建議比較。
- RFC 2373，IP 第 6 版定址架構。
- RFC 2374，IPv6 彙總全域單點傳播位址格式。

您也可以在 IP 版本 6 (IPv6) 找到 IPv6 的相關資訊。

另請參閱

- IPv6 通訊端範例

- 網路程式設計範例
- 通訊端

IPv6 定址

2020/3/20 • • Edit Online

網際網路通訊協定第 6 版 (IPv6) 的位址長度為 128 個位元長。使用這類大型位址空間的其中一個原因是，將可用的位址細分到路由網域的階層，反映網際網路的拓撲。另一個原因是，將連線裝置之網路網路介面卡 (或介面) 的位址對應至網路。IPv6 特有的固有功能，是在其最低層級解析位址，即網路介面層級，且也擁有自動組態功能。

文字表示法

下列是三種使用文字字串表示 IPv6 位址的慣例格式：

- **冒號:十六進位格式**。這是慣用的格式 n:n:n:n:n:n:n:n。每個 n 代表位址的八個 16 位元項目的一個十六進位值。例如：`3FFE:FFFF:7654:FEDA:1245:BA98:3210:4562`。
- **壓縮格式**。因為位址長度的原因，位址通常會包含由零組成的長字串。若要簡化撰寫這些地址，請使用壓縮格式，其中單一連續序列的 0 區塊會以雙冒號符號 (::) 表示。這個符號在位址中只會出現一次。例如，多點傳送位址 `FFED:0:0:0:0:BA98:3210:4562` 的壓縮格式是 `FFED::BA98:3210:4562`。單點傳播位址 `3FFE:FFFF:0:0:8:800:20C4:0` 的壓縮格式是 `3FFE:FFFF::8:800:20C4:0`。回送位址 `0:0:0:0:0:0:0:1` 的壓縮格式是 `::`1。未指定的位址 `0:0:0:0:0:0:0:0` 的壓縮格式是 `::`。
- **混合格式**。這種格式結合了 IPv4 和 IPv6 位址。在此情況下，位址格式是 n:n:n:n:n:n:d.d.d.d，每個 n 代表六個 IPv6 高序位 16 位元位址項目的十六進位值，每個 d 代表 IPv4 位址的十進位值。

位址類型

位址中的前置位元會定義特定的 IPv6 位址類型。包含這些前置位元的變數長度欄位稱為格式首碼 (FP)。

IPv6 單點傳播位址分為兩個部分。第一個部分包含位址前置詞，第二個部分包含介面識別碼。表示 IPv6 位址/前置詞組合的簡潔方式如下：ipv6 位址/前置詞長度。

下例是具有 64 位元前置詞的位址。

`3FFE:FFFF:0:CD30:0:0:0:0/64`.

本例中的前置詞為 `3FFE:FFFF:0:CD30`。位址也可以壓縮格式寫入，如 `3FFE:FFFF:0:CD30::/64`。

IPv6 會定義下列位址類型：

- **單點傳播位址**。單一介面的識別碼。傳送到此位址的封包會被傳遞到已識別的介面。單點傳播位址是高序位八位元值從多點傳送位址中區隔出來的。多點傳送位址之高序位八位元的十六進位值為 FF。此八位元的任何其他值會識別單點傳播位址。以下是不同類型的單點傳播位址：
  - **連結-本機位址**。這些位址是用在單一連結，且具有下列格式：FE80::*InterfaceID*。連結上的節點之間在處理自動位址組態、鄰居探索，或沒有路由器時，會使用連結-本機位址。連結-本機位址主要是用在啟動時，以及系統尚未取得較大範圍的位址時。
  - **網站-本機位址**。這些位址是用在單一網站，且具有下列格式：FEC0::*SubnetID*:*InterfaceID*。網站-本機位址可用來在網站中定址，不需要全域前置詞。
  - **全域 IPv6 單點傳播位址**。這些位址可以用在整個網際網路上，而且具有下列格式：010(FP，3 位元) TLA ID (13 位元) 保留項目 (8 位元) NLA ID (24 位元) SLA ID (16 位元) *InterfaceID* (64 位元)。
- **多點傳送位址**。一組介面 (通常屬於不同的節點) 的識別碼。傳送到此位址的封包會被傳遞到該位址所識別的所有介面。多點傳送位址類型取代 IPv4 廣播位址。

- **任一傳播位址**。一組介面 (通常屬於不同的節點) 的識別碼。傳送到此位址的封包只會被傳遞該位址所識別的一個介面。這是路由計量所識別的最近介面。任一傳播位址皆是取自單點傳播位址空間，而且語法上沒有分別。位址介面會將單點傳播與任一傳播位址之間的差異當成其組態的函式執行。

一般情況下，節點一律會有連結-本機位址。它可能有網站-本機位址和一或多個全域位址。

另請參閱

- 互聯網協定版本 6
- 通訊端

IPv6 路由

2020/3/20 • • Edit Online

IPv6 的優點是彈性的路由機制。因為 IPv4 網路識別碼過去和現在的配置方式，大型的路由表需要由位於網際網路骨幹的路由器維護。這些路由器必須知道所有路由，才能轉送在網際網路上可能導向至任何節點的封包。因為其能彙總位址，IPv6 可讓您彈性定址，並大幅降低路由表大小。在這個新的定址架構中，中繼路由器必須只持續追蹤其網路的本機部分，才能正確地轉寄訊息。

鄰居搜索

鄰居探索提供的部分功能包括：

- 路由器探索。這可讓主機識別本機路由器。
- 位址解析。這可讓節點解析連結層位址，以對應下個躍點位址 (位址解析通訊協定 [ARP] 的取代項目)。
- 處理自動組態。這可讓主機自動設定網站-本機和全域位址。

鄰居探索使用 IPv6 的網際網路控制訊息通訊協定 (ICMPv6) 訊息，包括：

- 路由器通告。由路由器依虛擬週期傳送，或回應路由器請求。IPv6 路由器使用路由器通告公告其可用性、位址前置詞和其他參數。
- 路由器請求。由主機傳送，以要求連結上的路由器立即傳送路由器通告。
- 芳鄰請求。由節點傳送，以處理位址解析、重複位址偵測，或驗證是否仍然可以連線到芳鄰。
- 芳鄰通告。由節點傳送以回應請求，或通知芳鄰連結層位址中的變更。
- 重新導向。由路由器傳送，以指出傳送節點之特定目的地的下一個較佳躍點位址。

另請參閱

- 互聯網協定版本 6
- 通訊端

IPv6 自動設定

2020/3/20 • • Edit Online

IPv6 的一個重要目標是支援節點隨插即用。亦即，節點應該可能插入 IPv6 網路，並且可以自動設定，不需要人為介入。

自動組態的類型

IPv6 支援下列類型的自動組態：

- **具狀態的自動組態**。這種組態需要某種程度的人為介入，因為它需要 IPv6 的動態主機設定通訊協定 (DHCPv6) 伺服器來安裝與管理節點。DHCPv6 伺服器會保留一份要提供組態資訊的節點。它也會維護狀態資訊，讓伺服器知道每個位址使用多長時間，以及何時可供重新指派。
- **無狀態的自動組態**。這種組態適合小型組織和個人。在此情況下，每部主機都會從接收到的路由器通告內容中判斷其位址。使用 IEEE EUI-64 標準定義位址的網路識別碼部分，就可合理假設主機位址在連結上的唯一性。

無論以何方式判斷位址，節點都必須確認其潛在位址對本機連結而言是唯一的。這是透過將芳鄰請求訊息傳送給潛在位址所完成。如果節點收到任何回應，它會知道位址已在使用中，必須判斷另一個位址。

IPv6 行動性

行動裝置的激增造成了新的需求：裝置必須能夠在 IPv6 網際網路上任意變更位置，但卻仍然能維持現有的連線。為提供這項功能，要指派給行動節點一個隨時可以連線的主目錄位址。當行動節點在主目錄時，它會連接到主目錄連結，並使用主目錄位址。當行動節點離開主目錄時，主目錄代理程式 (通常是路由器)，會在行動節點和它通訊的節點之間轉送訊息。

另請參閱

- 互聯網協定版本 6
- 通訊端

啟用和停用 IPv6

2020/3/20 • • Edit Online

若要使用 IPv6 通訊協定，請確定您執行的作業系統版本支援 IPv6，並確認已正確設定作業系統和網路類別。

設定步驟

下表列出各種組態

ıɛɛɛɛɛɛɛ IPV6？	ıɛɛɛɛɛɛɛ IPV6？	ıı
否	否	可以剖析 IPv6 位址。
否	是	可以剖析 IPv6 位址。
是	否	使用未標記為已淘汰的名稱解析方法，可以剖析 IPv6 位址並解析 IPv6 位址。
是	是	使用所有的方法，包括標記為已淘汰的在內，可以剖析和解析 IPv6 位址。

請注意，若要在 System.Net 命名空間中啟用所有類別的 IPv6 支援，您必須修改電腦組態檔或應用程式的組態檔。應用程式組態檔的優先順序高於電腦組態檔。

如需如何修改電腦組態檔 *machine.config* 的範例，以啟用 Ipv6 支援，請參閱如何：修改電腦組態檔以啟用 Ipv6 支援。此外，確定作業系統已啟用 IPv6 支援。

.NET Framework 在組態檔中設定的組態參數，如下所示：

```
<system.net>
  ...
  <settings>
    ...
    <ipv6 enabled="true"/>
    ...
  </settings>
  ...
</system.net>
```

如果是 .NET Framework 1.1 版及更早版本，[已啟用 IPv6]**** 組態參數的值會指定 System.Net.Dns 類別的成員是否傳回 IPv6 位址。

針對 .NET Framework 2.0 版及更新版本，如果 Windows 支援 IPv6，則 System.Net.Dns 類別的成員 (例如，Dns.GetHostEntry 方法) 將會傳回含有一個限制的 IPv6 位址。DNS System.Net.Dns 已淘汰的成員 (例如，Dns.Resolve 方法) 會讀取並辨識組態檔中啟用 ipv6 設定的值。

另請參閱

- 互聯網協定版本 6
- 通訊端
- 網路設置架構
- <ipv6>元素（網路設置）

如何：修改電腦設定檔案以啟用 IPv6 支援

2020/3/20 • • Edit Online

下列程式碼範例示範如何修改電腦組態檔 *machine.config* 來啟用 IPv6 支援。*machine.config* 檔案是儲存在 Windows 安裝目錄中的 *%Windir%\Microsoft.NET\Framework* 資料夾下。*在 %Windir%_Microsoft.NET_Framework*的資料夾中，有一個單獨的*電腦.config*檔，用於電腦上安裝的每個版本的 .NET Framework（例如，*C:_WINDOWS_Microsoft.NET_Framework_v2.0.50727_機器.config*）。

您也可以在應用程式的組態檔中進行這些設定，它的優先順序高於電腦組態檔。

如果是 .NET Framework 1.1 版及更早版本，[已啟用 IPv6]**** 組態參數的值會指定 System.Net.Dns 類別的成員是否傳回 IPv6 位址。

如果是 .NET Framework 2.0 版及更新版本，如果 Windows 支援 IPv6，則 System.Net.Dns 類別的所有成員 (例如，Dns.GetHostEntry 方法) 將會傳回含有一個限制的 IPv6 位址。System.Net.Dns 類別的過時成員 (例如，Dns.Resolve 方法) 會讀取並辨識組態檔中的值。

> **NOTE**
> 如果是 .NET Framework 2.0 版及更新版本，預設會啟用 IPv6。如果是 .NET Framework 1.1 版及更早版本，預設會停用 IPv6。

範例

```
<system.net>
    ……
    <settings>
        ……
        <ipv6 enabled="true"/>
    ……
    </settings>
    ……
</system.net>
```

另請參閱

- IPv6 定址
- 網路設置架構
- <ipv6>元素（網路設置）

設定網際網路應用程式

2020/3/20 • • Edit Online

<system.Net>元素(網路設置)配置元素包含應用程式的網路設定資訊。使用<system.Net>元素(網路設置)元素，可以設置代理伺服器、設置連接管理參數，並在應用程式中包括自訂身份驗證和請求模組。

預設代理>元素(網路設置)元素定義類返回的代理伺服器。< `GlobalProxySelection` 任何未將其專屬 Proxy 屬性設定為特定值的 HttpWebRequest 都會使用預設 Proxy。除了設定 Proxy 位址之外，您還可以建立將不會使用 Proxy 的伺服器位址清單，以及指出不應該將 Proxy 用於本機位址。

請務必注意，Microsoft Internet Explorer 設定是與組態設定一起使用，但優先使用後者。

下列範例會將預設 Proxy 伺服器位址設定為 `http://proxyserver`、指出不應該將 Proxy 用作本機位址，以及指定對 contoso.com 網域所在伺服器的所有要求都應該略過 Proxy。

```
<configuration>
    <system.net>
        <defaultProxy>
            <proxy
                usesystemdefault = "false"
                proxyaddress = "http://proxyserver:80"
                bypassonlocal = "true"
            />
            <bypasslist>
                <add address="http://[a-z]+\.contoso\.com/" />
            </bypasslist>
        </defaultProxy>
    </system.net>
</configuration>
```

使用<連接管理>元素(網路設置)元素配置可連接到特定伺服器或所有其他伺服器的持久連接數。下列範例會設定應用程式使用兩個與 `www.contoso.com` 伺服器的持續連線、四個 IP 位址為 192.168.1.2 之伺服器的持續連線，以及一個與所有其他伺服器的持續連線。

```
<configuration>
    <system.net>
        <connectionManagement>
            <add address="http://www.contoso.com" maxconnection="2" />
            <add address="192.168.1.2" maxconnection="4" />
            <add address="*" maxconnection="1" />
        </connectionManagement>
    </system.net>
</configuration>
```

自訂身份驗證模組配置了<身份驗證模組>元素(網路設置)元素。自訂驗證模組必須實作 IAuthenticationModule 介面。

下列範例設定自訂驗證模組。

```
<configuration>
    <system.net>
        <authenticationModules>
            <add type="MyAuthModule, MyAuthModule.dll" />
        </authenticationModules>
    </system.net>
</configuration>
```

您可以使用<WebRequestModule>元素（網路設置）元素來配置應用程式，以便使用特定于協定的自訂模組從 Internet 資源請求資訊。指定的模組必須實作 IWebRequestCreate 介面。您可以在組態檔中指定自訂模組來覆寫預設 HTTP、HTTPS 和檔案要求模組，如下列範例所示。

```
<configuration>
    <system.net>
        <webRequestModules>
            <add
                prefix="HTTP"
                type = "MyHttpRequest.dll, MyHttpRequestCreator"
            />
        </webRequestModules>
    </system.net>
</configuration>
```

另請參閱

- .NET 框架中的網路程式設計
- 網路設置架構
- <system.Net>元素（網路設置）

以 .NET Framework 進行網路追蹤

2020/3/20 • • Edit Online

針對方法叫用及 Managed 應用程式所產生的網路流量，.NET Framework 中的網路追蹤能提供對這些相關資訊的存取。若要為開發中的應用程式進行偵錯，以及分析已部署的應用程式，此功能會非常有用。您可以自訂網路追蹤所提供的輸出，在程式開發時期和實際執行環境中支援不同的使用案例。

若要啟用 .NET Framework 中的網路追蹤，您必須為追蹤輸出選取一個目的地，並在應用程式或電腦組態檔中加入網路追蹤組態設定。如需組態檔和其使用方式的描述，請參閱組態檔。如需如何啟用網路追蹤的資訊，請參閱啟用網路追蹤。如需必須新增至組態檔之設定的資訊，請參閱如何：設定網路追蹤。

啟用追蹤之後，您就可以擷取 System.Net 類別所輸出的追蹤資訊。可產生追蹤資訊的網路類別成員在它們的 .NET Framework 類別庫文件的＜備註＞一節中會包含下面附註：

> **NOTE**
> 在應用程式中啟用網路追蹤時，這個成員會輸出追蹤資訊。如需詳細資訊，請參閱＜網路追蹤＞。

另請參閱

- 啟用網路追蹤
- 如何：設定網路追蹤
- 解譯網路追蹤
- 追蹤和稽核應用程式

解譯網路追蹤

2020/3/20 • • Edit Online

啟用網路追蹤時，您可以使用追蹤來擷取應用程式對各種 System.Net 類別成員的呼叫。這些呼叫的輸出可能類似下列範例。

```
[588]   (4357)   Entering Socket#33574638::Send()
[588]   (4387)   Exiting Socket#33574638::Send()-> 61#61
```

在上述範例中，[588] 是目前執行緒的唯一識別碼。(4357) 和 (4387) 時間戳記表示自應用程式啟動後所經歷的毫秒數。時間戳記後面的資料會顯示應用程式進入和結束 Socket.Send 方法。執行 Send 方法之物件的唯一識別碼是 33574638。方法結束追蹤包含傳回值 (上例中為 61)。

網路追蹤可以擷取您的應用程式使用應用程式層級通訊協定，例如超文字傳輸通訊協定 (HTTP)，所傳送或接收的網路流量。此資料可以擷取成文字或十六進位資料。當您指定 includehex 作為 tracemode 屬性的值時，可以使用十六進位資料。（有關此屬性的詳細資訊，請參閱：配置網路跟蹤。以下示例跟蹤是使用include x生成的。

```
[1692] (1142) 00000000 : 47 45 54 20 2F 77 70 61-64 2E 64 61 74 20 48 54 : GET /wpad.dat HT

[1692] (1142) 00000010 : 54 50 2F 31 2E 31 0D 0A-48 6F 73 74 3A 20 69 74 : TP/1.1..Host: it

[1692] (1142) 00000020 : 67 70 72 6F 78 79 0D 0A-43 6F 6E 6E 65 63 74 69 : gproxy..Connecti

[1692] (1142) 00000030 : 6F 6E 3A 20 43 6C 6F 73-65 0D 0A 0D 0A : on: Close....
```

若要省略十六進位資料，請指定 protocolonly 作為 tracemode 屬性的值。當指定 protocolonly 時，下列範例會顯示追蹤。

```
[2444] (594) Data from ConnectStream#33574638::WriteHeaders<<GET /wpad.dat HTTP/1.1

Host: itgproxy

Connection: Close
```

另請參閱

- 啟用網路追蹤
- 如何：設定網路追蹤
- 以 .NET Framework 進行網路追蹤

啟用網路追蹤

2020/3/20 • • Edit Online

針對方法叫用及 Managed 應用程式所產生的網路流量，網路追蹤能提供對這些相關資訊的存取。您必須完成下列工作，才能在您的應用程式中啟用網路追蹤：

- 編譯程式碼並啟用追蹤。如需啟用追蹤所需之編譯器參數的詳細資訊，請參閱如何：使用追蹤和偵錯進行條件式編譯。
- 指定追蹤輸出的目的地。
- 設定網路追蹤的行為。如需詳細資訊，請參閱如何：設定網路追蹤。

最常見的追蹤目的地，也稱為追蹤接聽項，是預設的接聽程式和記錄檔。

如未指定追蹤接聽項，追蹤就會使用預設的接聽程式。您可以在啟用了 Managed 程式碼的偵錯工具中，例如隨附於 .NET Framework SDK 的 CLR 偵錯工具，或隨附於 Windows SDK 的 DBwin32.exe，執行您的程式碼，檢視傳送到預設接聽程式的訊息。使用 CLR 偵錯工具，追蹤訊息會出現在 [輸出]**** 視窗中。

如果您偏好使用檔案來接收追蹤，您可以使用組態設定來指定記錄檔，如下列範例所示。(如需組態檔的一般討論，請參閱組態檔。)

若要將追蹤傳送至記錄檔，請將以下節點新增至適當 (應用程式或電腦) 組態檔的 `<system.diagnostics>` 節點。您可以變更檔案名稱 (trace.log) 以符合您的需求。

```
<system.diagnostics>
  <trace autoflush="true" indentsize="4">
    <listeners>
      <add name="file" type="System.Diagnostics.TextWriterTraceListener" initializeData="trace.log"/>
    </listeners>
  </trace>
</system.diagnostics>
```

另請參閱

- 解譯網路追蹤
- 以 .NET Framework 進行網路追蹤
- 追蹤和稽核應用程式

如何：配置網路跟蹤

2020/3/20 • • Edit Online

應用程式或電腦組態檔都會保存可決定網路追蹤格式和內容的設定。在執行這個程序之前，請確認已啟用追蹤。有關詳細資訊，請參閱啟用網路跟蹤。

電腦設定檔，*電腦.config*，存儲在 *%windir%\Microsoft.NET_Framework*資料夾中。*在 %windir%_Microsoft.NET_Framework*的資料夾中，對於電腦上安裝的每個版本的 .NET Framework，有一個單獨的*電腦.config*檔，例如：

- *C:_WINDOWS_微軟.NET_Framework_v2.0.50727_機器。*
- *C:_WINDOWS_微軟.NET_Framework64_v4.0.30319\機器。*

您也可以在應用程式的組態檔中進行這些設定，它的優先順序高於電腦組態檔。

配置網路跟蹤

要配置網路跟蹤，請將以下行添加到相應的設定檔。下表中會說明這些設定的值和選項。

```
<configuration>
  <system.diagnostics>
    <sources>
      <source name="System.Net" tracemode="includehex" maxdatasize="1024">
        <listeners>
          <add name="System.Net"/>
        </listeners>
      </source>
      <source name="System.Net.Cache">
        <listeners>
          <add name="System.Net"/>
        </listeners>
      </source>
      <source name="System.Net.Http">
        <listeners>
          <add name="System.Net"/>
        </listeners>
      </source>
      <source name="System.Net.Sockets">
        <listeners>
          <add name="System.Net"/>
        </listeners>
      </source>
      <source name="System.Net.WebSockets">
        <listeners>
          <add name="System.Net"/>
        </listeners>
      </source>
   </sources>
    <switches>
      <add name="System.Net" value="Verbose"/>
      <add name="System.Net.Cache" value="Verbose"/>
      <add name="System.Net.Http" value="Verbose"/>
      <add name="System.Net.Sockets" value="Verbose"/>
      <add name="System.Net.WebSockets" value="Verbose"/>
    </switches>
    <sharedListeners>
      <add name="System.Net"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="network.log"
        traceOutputOptions="ProcessId, DateTime"
      />
    </sharedListeners>
    <trace autoflush="true"/>
  </system.diagnostics>
</configuration>
```

從方法跟蹤輸出

當您將名稱加入至 `<switches>` 區塊時，追蹤輸出就會包括來自與該名稱相關的某些方法的資訊。下表描述了輸出：

ㄖ	ㄖㄖㄖ
`System.Net.Sockets`	Socket、TcpListener和TcpClientDns類的一些公共方法。
`System.Net`	HttpWebRequest、HttpWebResponse和FtpWebRequest類以及FtpWebResponseSSL 調試資訊（無效證書、缺少頒發者清單和用戶端憑證錯誤）的某些公共方法。
`System.Net.HttpListener`	HttpListener、HttpListenerRequest 和 HttpListenerResponse 類別的一些公用方法。

■■	■■■■
`System.Net.Cache`	`System.Net.Cache` 中的一些私用和內部方法。
`System.Net.Http`	HttpClient、DelegatingHandler、HttpClientHandler、HttpMessageHandler、MessageProcessingHandler 和 WebRequestHandler 類別的一些公用方法。
`System.Net.WebSockets.WebSocket`	ClientWebSocket 和 WebSocket 類別的一些公用方法。

跟蹤輸出屬性

下表中列出的屬性配置跟蹤輸出：

■■■■	■■■
`value`	必要的 String 屬性。設定輸出的詳細等級。合法值為 `Critical`、`Error`、`Verbose`、`Warning` 和 `Information`。 必須在■■元素的■元素上設置此屬性。如果在▮元素上設置了此屬性，則引發異常。 範例：`<add name="System.Net" value="Verbose"/>`
`maxdatasize`	選擇性 Int32 屬性。設定每一行追蹤所包含之網路資料的最大位元組數。預設值為 1024。 必須在▮元素上設置此屬性。如果在■■元素下的元素上設置此屬性，則引發異常。 範例： `<source name="System.Net" tracemode="includehex" maxdatasize="1024">`
`tracemode`	選擇性 String 屬性。設定為 `includehex` 以便使用十六進位和文字格式來顯示通訊協定追蹤。設定為 `protocolonly` 則只會顯示文字。預設值是 `includehex`。 必須在▮元素上設置此屬性。如果在■■元素下的元素上設置此屬性，則引發異常。 範例： `<source name="System.Net" tracemode="includehex" maxdatasize="1024">`

另請參閱

- 解譯網路追蹤
- 以 .NET Framework 進行網路追蹤
- 啟用網路追蹤
- 追蹤和稽核應用程式

網路應用程式的快取管理

2020/3/20 • • Edit Online

本主題和其相關子主題描述如何使用 WebClient、WebRequest、HttpWebRequest 和 FtpWebRequest 類別所取得資源的快取。

快取提供應用程式已要求的資源暫時儲存位置。如果應用程式多次要求相同的資源，則可以從快取傳回資源，避免從伺服器重新要求它的額外負荷。快取可以透過減少取得所要求資源所需的時間，來改善應用程式效能。快取也可以減少往返伺服器的次數，來降低網路流量。雖然快取可改善效能，但是會增加傳回給應用程式的資源過時的風險，這表示它與未使用快取時由伺服器所傳送的資源不同。

快取可讓未經授權的使用者或處理序讀取敏感性資料。可以從快取擷取所快取的已驗證回應，而不需要額外授權。如果啟用快取，請將 CachePolicy 變更為 BypassCache 或 NoCacheNoStore，以停用此要求的快取。

基於安全性考量，**不**建議針對中介層案例進行快取。

本節內容

緩存策略
說明快取原則是什麼，以及如何定義快取原則。

以位置為基礎的快取原則
定義可用於超文字傳輸通訊協定 (http 和 https) 資源的每種以位置為基礎的快取原則。

Time-Based Cache Policies
描述可用來自訂以時間為基礎的快取原則的條件。

設定網路應用程式的快取功能
描述如何以程式設計方式建立快取原則以及使用快取的要求。

參考

System.Net.Cache
可定義類型和列舉，這些類型和列舉是用來定義使用 WebRequest、HttpWebRequest 和 FtpWebRequest 類別所取得之資源的快取原則。

快取原則

2020/3/20 • • Edit Online

快取原則所定義的規則用來判斷是否可以使用所要求資源的快取複本來滿足要求。應用程式指定有效期限的用戶端快取需求，但有效的快取原則是由用戶端快取需求、伺服器內容到期需求和伺服器重新驗證需求所決定。用戶端快取原則與伺服器需求的互動一律會導致最保守的快取原則，協助確保將最新內容傳回給用戶端應用程式。

快取原則是以位置為基礎或以時間為基礎。以位置為基礎的快取原則會根據所要求資源可以使用的位置，定義快取項目的有效期限。以時間為基礎的快取原則會使用擷取資源的時間、與資源一起傳回的標頭，以及目前時間，定義快取項目的有效期限。大部分的應用程式可以使用以時間為基礎的預設快取原則，以實作可從網際網路工程任務推動小組 (IETF) 網站取得之 RFC 2616 中所指定的快取原則。

下表中所述的類別是用來指定快取原則。

HttpRequestCachePolicy	針對使用 HttpWebRequest 物件所要求的資源，代表以位置為基礎和以時間為基礎的快取原則。
RequestCachePolicy	針對使用 WebRequest 物件所要求的資源，代表以位置為基礎的快取原則或以 Default 時間為基礎的快取原則。
HttpCacheAgeControl	指定值，用來建立以時間為基礎之 HttpRequestCachePolicy 物件。
HttpRequestCacheLevel	指定值，用來建立以位置為基礎和以時間為基礎之 HttpRequestCachePolicy 物件。
RequestCacheLevel	指定值，用來建立以位置為基礎或 Default 以時間為基礎的 RequestCachePolicy 物件。

您可以定義應用程式所提出之所有要求或個別要求的快取原則。當您同時指定應用程式層級快取原則和要求層級快取原則時，會使用要求層級原則。您可以透過程式設計方式或是使用應用程式或電腦組態檔，來指定應用程式層級快取原則。有關詳細資訊，請參閱<請求緩存>元素（網路設置）。

若要建立快取原則，您必須建立 RequestCachePolicy 或 HttpRequestCachePolicy 類別的執行個體來建立原則物件。若要在要求上指定原則，請將要求的 CachePolicy 屬性設定為原則物件。以程式設計方式設定應用程式層級原則時，請將 DefaultCachePolicy 屬性設定為原則物件。

如需示範如何建立和使用快取原則的程式碼範例，請參閱設定網路應用程式的快取功能。

另請參閱

- 網路應用程式的快取管理
- 以位置為基礎的快取原則
- Time-Based Cache Policies
- 設定網路應用程式的快取功能

以位置為基礎的快取原則

2020/3/20 • • Edit Online

以位置為基礎的快取原則，會根據所要求資源可以使用的位置，定義有效快取項目的有效期限。如果使用它不違反伺服器指定的重新驗證需求，快取的資源即為有效。使用 RequestCachePolicy 或 HttpRequestCachePolicy 類別建構函式可以程式設計方式建立以位置為基礎的快取原則。以位置為基礎的原則類型，是使用 RequestCacheLevel 或 HttpRequestCacheLevel 列舉值傳遞至建構函式。如需建立以位置為基礎之快取原則的程式碼範例，請參閱如何：為應用程式設定以位置為基礎的快取原則。下列各節說明超文字傳輸通訊協定 (http 和 https) 資源的每種以位置為基礎的快取原則。

可用時快取原則

如果有效的要求資源位於本機快取，則會使用快取的資源；否則，會向伺服器傳送資源要求。如果在用戶端與伺服器之間的任何快取中都能取得要求的資源，中繼快取即可滿足要求。

僅限快取原則

如果有效的要求資源位在本機快取，則會使用快取的資源。當指定這個快取原則層級時，如果項目不在本機快取，就會擲回 WebException 例外狀況。

僅限快取或下一個快取原則

如果有效的要求資源位在本機快取或區域網路的中繼快取，則會使用快取的資源。否則會擲回 WebException 例外狀況。在 HTTP 快取通訊協定中，這是使用 only-if-cached 的快取控制指示詞所達成。

無快取且無存放區原則

要求的資源絕不會從任何快取使用，也絕不會放在任何快取中。如果要求的資源出現在本機快取中，它會被移除。這個原則層級指出也應該移除資源的中繼快取。在 HTTP 快取通訊協定中，這是使用 no-store 的快取控制指示詞所達成。

重新整理原則

如果要求的資源是從伺服器取得，或在本機快取以外的快取中找到，就可以使用。在中繼快取滿足要求之前，該快取必須重新向伺服器驗證其快取的項目。在 HTTP 快取通訊協定中，這是使用 max-age = 0 的快取控制指示詞和 no-cache Pragma 標頭所達成。

重新載入原則

必須從伺服器取得要求的資源。回應可能是儲存在本機快取中。在 HTTP 快取通訊協定中，這是使用 no-cache 的快取控制指示詞和 no-cache Pragma 標頭所達成。

重新驗證原則

比較快取中的資源複本和伺服器上的複本。如果伺服器上的複本較新，就會用它來滿足要求，並取代快取中的複本。如果快取中的複本和伺服器上的版本相同，就使用快取的複本。在 HTTP 快取通訊協定中，使用條件式要求即可達成。

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- Time-Based Cache Policies
- 設定網路應用程式的快取功能
- <請求緩存>元素（網路設置）

以時間為基礎的快取原則

2020/3/20 • • Edit Online

以時間為基礎的快取原則會使用擷取資源的時間、與資源一起傳回的標頭，以及目前時間，定義快取項目的有效期限。設定以時間為基礎的快取原則時，您可以使用 Default 時間基礎原則，或建立自訂的時間基礎原則。為使用超文字傳輸通訊協定 (HTTP) 取得的資源，使用以時間為基礎的預設原則時，確切的快取行為是由快取回應中包含的標頭，以及 RFC 2616 的 13 與 14 一節中所指定的行為來，RFC 2616 可在網際網路工程任務推動小組 (IETF) 網站取得。如需示範如何為 HTTP 資源設定以時間為基礎之預設原則的程式碼範例，請參閱如何：為應用程式設定以時間為基礎的預設快取原則。如需示範如何建立和使用快取原則的程式碼範例，請參閱設定網路應用程式的快取功能。

判斷快取項目有效期限的準則

若要自訂以時間為基礎的快取原則，您可以指定要使用下列一或多個準則來判斷快取項目的有效期限：

- 最長使用期限
- 最長過時
- 最短有效期限
- 快取同步處理日期

> **NOTE**
>
> 使用時間為基礎的預設快取原則不應與設定應用程式的預設快取原則混淆。以時間為基礎的預設原則是可以用於要求或應用程式層級的特定原則。應用程式的預設快取原則是在要求上未設定原則時生效的原則 (以位置為基礎或以時間為基礎)。如需設定應用程式之預設快取原則的詳細資料，請參閱DefaultCachePolicy。

Maximum Age

最長使用期限原則準則指定可以使用資源快取複本的時間量。如果資源的快取複本超過指定的時間量，則必須藉由針對伺服器上的內容檢查資源，來重新驗證資源。如果最長使用期限允許在資源到期後還使用它，則除非同時指定最長過時值，否則不會採用此準則。

最長過時

最長過時原則準則指定在可以使用資源的快取複本，且在內容到期日之後的時間長度。這是唯一允許在資源過期之後仍使用資源的快取原則準則。

最短有效期限

最短有效期限原則準則指定在可以使用資源的快取複本，且在內容到期日之前的時間長度。此原則的效果會使快取項目在它的到期日之前便過期，因此最短有效期限和最長過時設定兩者互斥。

快取同步處理日期

快取同步處理日期原則準則會藉由針對伺服器上的內容檢查資源的快取複本，判斷它何時必須重新驗證。如果項目快取後內容已變更，則會從伺服器擷取它、儲存在快取，並傳回到應用程式。如果內容未變更，其時間戳記會更新，且應用程式會取得快取的內容。

快取同步處理日期可讓您指定必須重新驗證快取內容的絕對日期。如果新的快取項目上次重新驗證是在快取同步處理日期之前，仍會發生與伺服器的重新驗證。如果快取項目重新驗證是在快取同步處理日期之後，而且沒有其他有效期限或伺服器重新驗證需求使快取項目失效，則會使用來自快取的項目。如果快取同步處理日期設定為未

來的日期，則每次要求時都會重新驗證項目，直到快取同步處理日期已過為止。

下列主題提供結合以時間為基礎之快取原則準則的影響資訊：

- 快取原則互動 — 最長使用期限和最長過時
- 快取原則互動 — 最長使用期限和最小有效期限

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- 設定網路應用程式的快取功能
- <請求緩存>元素（網路設置）

快取原則互動 — 最長使用期限和最長過時

2020/3/20 • • Edit Online

為了協助確保將最新內容傳回給用戶端應用程式，用戶端快取原則與伺服器重新驗證需求的互動一律會導致最保守的快取原則。本主題中的所有範例都會說明在 1 月 1 日快取並在 1 月 4 日到期之資源的快取原則。

在下列範例中，最長過時值 (`maxStale`) 是與最長使用期限 (`maxAge`) 一起使用：

- 根據 `maxAge` 值，如果快取原則設定 `maxAge` = 5 天，但未指定 `maxStale` 值，則在 1 月 6 日之前可以使用內容。不過，根據伺服器的重新驗證需求，內容會在 1 月 4 日到期。因為內容到期日較為保守 (更快)，所以其優先順序高於 `maxAge` 原則。因此，內容會在 1 月 4 日到期，而且必須重新驗證，即使未達到其最長使用期限也是一樣。
- 根據 `maxAge` 值，如果快取原則設定 `maxAge` = 5 天且 `maxStale` = 3 天，則在1 月 6 日之前可以使用內容。根據 `maxStale` 值，在 1 月 7 日之前可以使用內容。因此，會在 1 月 6 日重新驗證內容。
- 根據 `maxAge` 值，如果快取原則設定 `maxAge` = 5 天且 `maxStale` = 1 天，則在 1 月 6 日之前可以使用內容。根據 `maxStale` 值，在 1 月 5 日之前可以使用內容。因此，會在 1 月 5 日重新驗證內容。

最長使用期限小於內容到期日時，一律會使用更保守的快取行為，而且最長過時值沒有任何作用。下列範例說明達到最長使用期限 (`maxAge`) 但在內容到期之前，設定最長過時 (`maxStale`) 值的效果：

- 如果快取原則設定 `maxAge` = 1 天，但未指定 `maxStale` 值的值，則會在 1 月 2 日重新驗證內容，即使未到期也是一樣。
- 如果快取原則設定 `maxAge` = 1 天且 `maxStale` = 3 天，則會在 1 月 2 日重新驗證內容，來強制執行更保守的原則設定。
- 如果快取原則設定 `maxAge` = 1 天且 `maxStale` = 1 天，則會在 1 月 2 日重新驗證內容。

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- Time-Based Cache Policies
- 設定網路應用程式的快取功能
- 快取原則互動 — 最長使用期限和最小有效期限

快取原則互動 — 最長使用期限和最小有效期限

2020/3/20 • • Edit Online

為了協助確保將最新內容傳回給用戶端應用程式，用戶端快取原則與伺服器重新驗證需求的互動一律會導致最保守的快取原則。本主題中的所有範例都會說明在 1 月 1 日快取並在 1 月 4 日到期之資源的快取原則。

下列範例說明最長使用期限 (`maxAge`) 與最小有效期限 (`minFresh`) 值的互動所造成的快取原則。

- 如果快取原則設定 `maxAge` = 2 天，但未指定 `minFresh`，則會在 1 月 3 日重新驗證內容。
- 根據 `maxAge`，如果快取原則設定 `maxAge` = 2 天且 `minFresh` = 1 天，則會在 1 月 3 日之前整理內容。根據 `minFresh`，會在 1 月 3 日之前整理內容。因此，必須在 1 月 3 日重新驗證內容。
- 根據 `maxAge`，如果快取原則設定 `maxAge` = 2 天且 `minFresh` = 2 天，則會在 1 月 3 日之前整理內容。根據 `minFresh`，會在 1 月 2 日之前整理內容。因此，必須在 1 月 2 日重新驗證內容。

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- Time-Based Cache Policies
- 設定網路應用程式的快取功能
- 快取原則互動 — 最長使用期限和最長過時

設定網路應用程式的快取功能

2020/3/20 • • Edit Online

若要設定快取，您必須指定應用程式或 WebRequest 層級的快取原則。下列主題所提供的程式碼範例示範如何設定應用程式和要求來使用快取。

- 如何：為應用程式設定以位置為基礎的快取原則
- 如何：為應用程式設定以時間為基礎的預設快取原則
- 如何：自訂以時間為基礎的快取原則
- 如何：設定要求的快取原則

您也可以設定使用應用程式或電腦組態檔的快取原則。有關詳細資訊，請參閱|<請求緩存>元素（網路設置）。

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- Time-Based Cache Policies

如何：為應用程式設定以位置為基礎的快取原則

2020/3/20 • • Edit Online

以位置為基礎的快取原則，可讓應用程式明確地定義根據所要求資源位置的快取行為。本主題將示範如何以程式設計方式設定快取原則。有關使用設定檔為應用程式設定策略的資訊，請參閱<請求緩存>元素（網路設置）。

為應用程式設定以位置為基礎的快取原則

1. 建立 RequestCachePolicy 或 HttpRequestCachePolicy 物件。
2. 設定原則物件作為應用程式定義域的預設值。

設定從快取中取得所要求資源的原則

- 藉由將快取層級設定為 CacheIfAvailable 來建立下列原則：從快取中取得可用的所要求資源，否則將要求傳送到伺服器。用戶端與伺服器之間的任何快取都可以滿足要求，包括遠端快取。

```
public static void UseCacheIfAvailable()
{
    HttpRequestCachePolicy policy = new HttpRequestCachePolicy
        (HttpRequestCacheLevel.CacheIfAvailable);
    HttpWebRequest.DefaultCachePolicy = policy;
}
```

```
Public Shared Sub UseCacheIfAvailable()
    Dim policy As New HttpRequestCachePolicy _
         (HttpRequestCacheLevel.CacheIfAvailable)
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

設定防止任何快取提供資源的原則

- 藉由將快取層級設定為 NoCacheNoStore 來建立下列原則：防止任何快取提供所要求的資源。此原則層級會從本機快取中移除存在的資源，並且向遠端快取指出它們也應該移除資源。

```
public static void DoNotUseCache()
{
HttpRequestCachePolicy policy = new HttpRequestCachePolicy
        (HttpRequestCacheLevel.NoCacheNoStore);
    HttpWebRequest.DefaultCachePolicy = policy;
}
```

```
Public Shared Sub DoNotUseCache()
    Dim policy As New HttpRequestCachePolicy _
        (HttpRequestCacheLevel.NoCacheNoStore)
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

設定唯有所要求的資源位於本機快取中時，才會傳回這些資源的原則

- 藉由將快取層級設定為 CacheOnly 來建立下列原則：唯有所要求的資源位於本機快取中時，才會傳回這些資源。如果所要求的資源不在快取中，則會擲回 WebException 例外狀況。

```
public static void OnlyUseCache()
{
    HttpRequestCachePolicy policy = new HttpRequestCachePolicy
        (HttpRequestCacheLevel.CacheOnly);
    HttpWebRequest.DefaultCachePolicy = policy;
}
```

```
Public Shared Sub OnlyUseCache()
    Dim policy As New HttpRequestCachePolicy _
        (HttpRequestCacheLevel.CacheOnly)
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

設定防止本機快取提供資源的原則

- 藉由將快取層級設定為 Refresh 來建立下列原則：防止本機快取提供所要求的資源。如果所要求的資源位於中繼快取中，而且已成功重新驗證，則中繼快取可以提供所要求的資源。

```
public static void DoNotUseLocalCache()
{
 HttpRequestCachePolicy policy = new HttpRequestCachePolicy
        (HttpRequestCacheLevel.Refresh);
    HttpWebRequest.DefaultCachePolicy = policy;
}
```

```
Public Shared Sub DoNotUseLocalCache()
    Dim policy As New HttpRequestCachePolicy _
        (HttpRequestCacheLevel.Refresh)
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

設定防止任何快取提供所要求資源的原則

- 藉由將快取層級設定為 Reload 來建立下列原則：防止任何快取提供所要求的資源。伺服器傳回的資源可以儲存在快取中。

```
public static void SendToServer()
{
HttpRequestCachePolicy policy = new HttpRequestCachePolicy
        (HttpRequestCacheLevel.Reload);
    HttpWebRequest.DefaultCachePolicy = policy;
}
```

```
Public Shared Sub SendToServer()
    Dim policy As New HttpRequestCachePolicy _
        (HttpRequestCacheLevel.Reload)
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

設定伺服器上的資源並未比快取複本更新時，可讓任何快取提供所要求資源的原則

- 藉由將快取層級設定為 Revalidate 來建立下列原則：伺服器上的資源並未比快取複本更新時，可讓任何快取提供所要求的資源。

```
public static void CheckServer()
{
HttpRequestCachePolicy policy = new HttpRequestCachePolicy
         (HttpRequestCacheLevel.Revalidate);
    HttpWebRequest.DefaultCachePolicy = policy;
}
```

```
Public Shared Sub CheckServer()
    Dim policy As New HttpRequestCachePolicy _
        (HttpRequestCacheLevel.Revalidate)
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- Time-Based Cache Policies
- <請求緩存>元素（網路設置）

如何：為應用程式設定以時間為基礎的預設快取原則

2020/3/20 • • Edit Online

以時間為基礎的預設快取原則，可讓應用程式擁有與快取資源一起傳送之標頭所定義的快取行為，以及 RFC 2616 的第 13 節與第 14 節中定義的快取行為 (可從網際網路工程任務推動小組 (IETF) 網站取得)。這是適用於大部分應用程式的快取行為。

設定應用程式的預設自動原則

1. 建立以時間為基礎的預設原則物件。

2. 設定原則物件作為應用程式定義域的預設值。

範例

本節中的兩個範例會產生相同的原則。

下列範例會建立以時間為基礎的預設原則，並將它設定為應用程式定義域的預設值。

```
public static void SetDefaultTimeBasedPolicy ()
{
    HttpRequestCachePolicy policy = new HttpRequestCachePolicy ();
    HttpWebRequest.DefaultCachePolicy = policy ;
}
```

```
Public Shared Sub SetDefaultTimeBasedPolicy ()
    Dim policy = New HttpRequestCachePolicy ()
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

您也可以使用 RequestCachePolicy 類別來建立以時間為基礎的預設快取原則，如下列範例所示：

```
public static void SetDefaultTimeBasedPolicy2()
{
    RequestCachePolicy policy = new RequestCachePolicy ();
    HttpWebRequest.DefaultCachePolicy = policy ;
}
```

```
Public Shared Sub SetDefaultTimeBasedPolicy2()
    Dim policy As New RequestCachePolicy()
    HttpWebRequest.DefaultCachePolicy = policy
End Sub
```

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- Time-Based Cache Policies

- <請求緩存>元素（網路設置）

如何：自訂以時間為基礎的快取原則

2020/3/20 • • Edit Online

在建立以時間為基礎的快取原則時，您可以藉由指定最長使用期限、最短有效期限、最長過時或快取同步處理日期的值來自訂快取行為。HttpRequestCachePolicy 物件所提供的建構函式可讓您指定這些值的有效組合。

建立使用快取同步處理日期之以時間為基礎的快取原則

創建基於時間的緩存策略，該策略通過將DateTime物件傳遞給HttpRequestCachePolicy建構函式來使用緩存同步日期：

```
public static HttpRequestCachePolicy CreateLastSyncPolicy(DateTime when)
{
    var policy = new HttpRequestCachePolicy(when);
    Console.WriteLine("When: {0}", when);
    Console.WriteLine(policy.ToString());
    return policy;
}
```

```
Public Shared Function CreateLastSyncPolicy([when] As DateTime) As HttpRequestCachePolicy
    Dim policy As New HttpRequestCachePolicy([when])
    Console.WriteLine("When: {0}", [when])
    Console.WriteLine(policy.ToString())
    Return policy
End Function
```

輸出大致如下：

```
When: 1/14/2004 8:07:30 AM
Level:Default CacheSyncDate:1/14/2004 8:07:30 AM
```

建立依據最短有效期限之以時間為基礎的快取原則

MinFresh通過指定 `cacheAgeControl` 參數值並將TimeSpan物件傳遞給HttpRequestCachePolicy建構函式，創建基於最小新鮮度基於時間的緩存策略：

```
public static HttpRequestCachePolicy CreateMinFreshPolicy(TimeSpan span)
{
    var policy = new HttpRequestCachePolicy(HttpCacheAgeControl.MinFresh, span);
    Console.WriteLine(policy.ToString());
    return policy;
}
```

```
Public Shared Function CreateMinFreshPolicy(span As TimeSpan) As HttpRequestCachePolicy
    Dim policy As New HttpRequestCachePolicy(HttpCacheAgeControl.MinFresh, span)
    Console.WriteLine(policy.ToString())
    Return policy
End Function
```

對於下列引動過程：

```
CreateMinFreshPolicy(new TimeSpan(1,0,0));
```

輸出如下：

```
Level:Default MinFresh:3600
```

建立依據最短有效期限和最長使用期限之以時間為基礎的快取原則

創建基於最短新鮮度和最大年齡的基於時間的緩存策略，MaxAgeAndMinFresh指定參數`cacheAgeControl`值並將兩TimeSpan個物件傳遞給HttpRequestCachePolicy建構函式，一個指定資源的最大年齡，第二個指定從緩存返回的物件允許的最低新鮮度：

```
public static HttpRequestCachePolicy CreateFreshAndAgePolicy(TimeSpan freshMinimum, TimeSpan ageMaximum)
{
    var policy = new HttpRequestCachePolicy(HttpCacheAgeControl.MaxAgeAndMinFresh, ageMaximum, freshMinimum);
    Console.WriteLine(policy.ToString());
    return policy;
}
```

```
Public Shared Function CreateFreshAndAgePolicy(freshMinimum As TimeSpan, ageMaximum As TimeSpan) As
HttpRequestCachePolicy
    Dim policy As New HttpRequestCachePolicy(HttpCacheAgeControl.MaxAgeAndMinFresh, ageMaximum, freshMinimum)
    Console.WriteLine(policy.ToString())
    Return policy
End Function
```

對於下列引動過程：

```
CreateFreshAndAgePolicy(new TimeSpan(5,0,0), new TimeSpan(10,0,0));
```

輸出如下：

```
Level:Default MaxAge:36000 MinFresh:18000
```

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- Time-Based Cache Policies
- <請求緩存>元素（網路設置）

如何：設定要求的快取原則

2020/3/20 • • Edit Online

下列範例示範如何設定要求的快取原則。範例的輸入是如 `http://www.contoso.com/` 之類的 URI。

範例

下列程式碼範例會建立一個快取原則，允許所要求的資源在快取中的時間尚未超過一天時，從快取使用該資源。此範例會顯示一個訊息，指出是否已從快取使用資源 (例如，`"The response was retrieved from the cache : False."`)，然後顯示該資源。用戶端與伺服器之間的任何快取都可以滿足要求。

```
using System;
using System.Net;
using System.Net.Cache;
using System.IO;

namespace Examples.System.Net.Cache
{
    public class CacheExample
    {
        public static void UseCacheForOneDay(Uri resource)
        {
            // Create a policy that allows items in the cache
            // to be used if they have been cached one day or less.
            HttpRequestCachePolicy requestPolicy =
                new HttpRequestCachePolicy (HttpCacheAgeControl.MaxAge,
                TimeSpan.FromDays(1));

            WebRequest request = WebRequest.Create (resource);
            // Set the policy for this request only.
            request.CachePolicy = requestPolicy;
            HttpWebResponse response = (HttpWebResponse)request.GetResponse();
            // Determine whether the response was retrieved from the cache.
            Console.WriteLine ("The response was retrieved from the cache : {0}.",
               response.IsFromCache);
            Stream s = response.GetResponseStream ();
            StreamReader reader = new StreamReader (s);
            // Display the requested resource.
            Console.WriteLine(reader.ReadToEnd());
            reader.Close ();
            s.Close();
            response.Close();
        }
        public static void Main(string[] args)
        {
            if (args == null || args.Length != 1)
            {
                Console.WriteLine ("You must specify the URI to retrieve.");
                return;
            }
            UseCacheForOneDay (new Uri(args[0]));
        }
    }
}
```

```
Imports System
Imports System.Net
Imports System.Net.Cache
Imports System.IO
Namespace Examples.System.Net.Cache
    Public Class CacheExample
        Public Shared Sub UseCacheForOneDay(ByVal resource As Uri)
            ' Create a policy that allows items in the cache
            ' to be used if they have been cached one day or less.
            Dim requestPolicy As New HttpRequestCachePolicy _
                  (HttpCacheAgeControl.MaxAge, TimeSpan.FromDays(1))
            Dim request As WebRequest = WebRequest.Create(resource)
            ' Set the policy for this request only.
            request.CachePolicy = requestPolicy
            Dim response As HttpWebResponse = _
             CType(request.GetResponse(), HttpWebResponse)
            ' Determine whether the response was retrieved from the cache.
            Console.WriteLine("The response was retrieved from the cache : {0}.", _
                response.IsFromCache)
            Dim s As Stream = response.GetResponseStream()
            Dim reader As New StreamReader(s)
            ' Display the requested resource.
            Console.WriteLine(reader.ReadToEnd())
            reader.Close()
            s.Close()
            response.Close()
        End Sub
        Public Shared Sub Main(ByVal args() As String)
            If args Is Nothing OrElse args.Length <> 1 Then
                Console.WriteLine("You must specify the URI to retrieve.")
                Return
            End If
            UseCacheForOneDay(New Uri(args(0)))
        End Sub
    End Class
End Namespace
```

另請參閱

- 網路應用程式的快取管理
- 緩存策略
- 以位置為基礎的快取原則
- Time-Based Cache Policies
- <請求緩存>元素（網路設置）

網路程式設計的安全性

2020/4/9 • • Edit Online

.NET Framework System.Net 命名空間類別會提供常用網際網路應用程式驗證機制和 .NET Framework 程式碼存取權限的內建支援。

本節內容

.NET Framework 的傳輸層安全性 (TLS) 最佳做法
使用 .NET 框架描述 TLS 最佳實務。

使用安全通訊端層
描述如何使用安全通訊端層 (SSL) 連線。

網際網路驗證
描述如何使用 HTTP 驗證方法建立已驗證的 HTTP 伺服器連線。

Web 和通訊端權限
描述如何設定使用網際網路連線之應用程式的程式碼存取安全性。

相關章節

.NET 框架中的網路程式設計
介紹 System.Net 和 System.Net.Sockets 命名空間中的類別。

.NET Framework 的傳輸層安全性 (TLS) 最佳做法

2020/3/26 • • Edit Online

傳輸層安全性 (TLS) 通訊協定為一項業界標準，其設計目的是用來協助保護透過網際網路所通訊之資訊的隱私權。TLS 1.2 (英文) 是可提供優於先前版本之安全性的標準。TLS 1.2 最終將被最新發行的標準 TLS 1.3 取代，後者速度更快，安全性更高。本文提供保護使用 TLS 通訊協定之 .NET Framework 應用程式的建議。

為了確保能維持 .NET Framework 應用程式的安全性，TLS 版本**不應**為硬式編碼。.NET Framework 應用程式應使用作業系統 (OS) 所支援的 TLS 版本。

本文件適用於下列開發人員：

- 直接使用 System.Net API (例如，System.Net.Http.HttpClient 和 System.Net.Security.SslStream) 的開發人員。
- 直接使用運用 System.ServiceModel 命名空間之 WCF 用戶端和服務的開發人員。

建議您：

- 在應用程式上以 .NET Framework 4.7 或更新版本作為目標。在 WCF 應用程式上以 .NET Framework 4.7.1 或更新版本作為目標。
- 不要指定 TLS 版本。設定程式碼以由 OS 決定 TLS 版本。
- 執行完整的程式碼稽核，以確認您沒有指定 TLS 或 SSL 版本。

當應用程式讓 OS 選擇 TLS 版本時：

- 它會自動運用於未來新增的新通訊協定，例如 TLS 1.3。
- OS 會封鎖被發現不安全的通訊協定。

對程式碼進行稽核並做出程式碼變更一節會涵蓋對程式碼進行稽核及更新的相關內容。

本文說明如何針對應用程式所執行的.NET Framework 版本，啟用可供使用的最強安全性。當應用程式明確設定安全性通訊協定及版本時，它將會退出所有其他替代方案，並退出 .NET Framework 及 OS 預設行為。如果您想讓應用程式能夠交涉 TLS 1.2 連線，明確設定至較低的 TLS 版本將會防止 TLS 1.2 連線。

如果您無法避免採取通訊協定版本的硬式編碼，我們強烈建議您指定 TLS 1.2。有關識別和刪除 TLS 1.0 依賴項的指導，請下載"解決 TLS 1.0 問題白皮書"。

在 .NET Framework 4.7 中，WCF 預設支援 TLS1.0、1.1 及 1.2。從 .NET Framework 4.7.1 開始，WCF 預設會使用作業系統所設定的版本。如果有應用程式是搭配 `SslProtocols.None` 進行明確設定，WCF 在使用 NetTcp 傳輸時，便會使用作業系統的預設設定。

您可以在 GitHub 問題 .NET Framework 的傳輸層安全性 (TLS) 最佳做法 (英文) 中，詢問與本文件相關的問題。

對程式碼進行稽核並做出程式碼變更

針對 ASP.NET 應用程式，請檢查 *web.config* 的 `<system.web><httpRuntime targetFramework>` 元素，以確認您是使用正確的 .NET Framework 版本。

針對 Windows Forms 及其他應用程式，請參閱如何：將 .NET Framework 的某個版本設定為目標。

使用下列小節來確認您沒有使用特定的 TLS 或 SSL 版本。

若應用程式是以 .NET Framework 4.7 或更新版本作為目標

下列小節會示範如何確認您沒有使用特定的 TLS 或 SSL 版本。

針對 HTTP 網路功能

ServicePointManager，使用 .NET 框架 4.7 和更高版本將使用作業系統中配置的預設安全協定。要獲取預設作業系統選項（如果可能，請不要為ServicePointManager.SecurityProtocol屬性設置值，該值預設為SecurityProtocolType.SystemDefault）。

由於SecurityProtocolType.SystemDefault該設置導致ServicePointManager使用作業系統配置的預設安全協定，因此應用程式可能會根據運行的作業系統以不同的方式運行。例如，Windows 7 SP1 使用 TLS 1.0，而 Windows 8 和 Windows 10 使用 TLS 1.2。

本文剩下的內容，與針對 HTTP 網路功能將 .NET Framework 4.7 或更新版本設為目標無關。

針對 TCP 通訊端網路功能

SslStream，在使用 .NET Framework 4.7 及更新版本的情況下，預設會讓 OS 選擇最佳的安全性通訊協定和版本。若要在可能的情況下取得最佳的預設 OS 選擇，請不要使用會採用明確 SslProtocols 參數之 SslStream 的方法多載。否則，請傳遞 SslProtocols.None。建議您不要使用 Default；設定 `SslProtocols.Default` 會強制使用 SSL 3.0/TLS 1.0 並防止 TLS 1.2。

不要為 SecurityProtocol 屬性 (針對 HTTP 網路功能) 設定值。

不要使用會採用明確 SslProtocols 參數 (針對 TCP 通訊端網路功能) 之 SslStream 的方法多載。當您將應用程式目標重新設為 .NET Framework 4.7 或更新版本時，即是遵循最佳做法建議。

本主題剩下的內容，與針對 TCP 通訊端網路功能將 .NET Framework 4.7 或更新版本設為目標無關。

針對搭配憑證認證使用傳輸安全性的 WCF TCP 傳輸

WCF 會使用和其他 .NET Framework 相同的網路堆疊。

若您是以 4.7.1 為目標，WCF 預設便已設定為允許 OS 選擇最佳的安全性通訊協定，除非另外透過下列方式明確設定：

- 在您的應用程式組態檔中。
- **或**在您應用程式的原始程式碼中。

根據預設，.NET Framework 4.7 及更新版本已設定為使用 TLS 1.2，並允許使用 TLS 1.1 或 TLS 1.0 的連線。特過設定繫結以使用 SslProtocols.None，來設定 WCF 以允許 OS 選擇最佳的安全性通訊協定。這可以在 SslProtocols 上設定。`SslProtocols.None` 可以從 Transport 存取。`NetTcpSecurity.Transport` 可以從 Security 存取。

若您是使用自訂繫結：

- 特過設定 SslProtocols 以使用 SslProtocols.None，來設定 WCF 以允許 OS 選擇最佳的安全性通訊協定。
- **或**透過設定路徑 `system.serviceModel/bindings/customBinding/binding/sslStreamSecurity:sslProtocols` 來設定所使用的通訊協定。

若您**沒有**使用自訂繫結，**且**您是使用設定來設定 WCF 繫結，請透過設定路徑 `system.serviceModel/bindings/netTcpBinding/binding/security/transport:sslProtocols` 來設定所使用的通訊協定。

針對具有憑證認證的 WCF 訊息安全性

.NET Framework 4.7 及更新版本預設會使用於 SecurityProtocol 屬性中指定的通訊協定。當 AppContextSwitch `Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols` 設定為 `true` 時，WCF 會選擇最佳的通訊協定 (最高版本為 TLS 1.0)。

若應用程式是以 .NET Framework 4.7 之前的版本為目標

使用下列小節來對程式碼進行稽核，以確認您沒有使用特定的 TLS 或 SSL 版本：

針對 .NET Framework 4.6 - 4.6.2 且非 WCF

將 `DontEnableSystemDefaultTlsVersions``AppContext` 開關設置為 `false`。請參閱透過 AppContext 參數設定安全性。

針對使用搭配憑證認證使用 TCP 傳輸安全性之 .NET Framework 4.6 至 4.6.2 的 WCF

您必須安裝最新的 OS 修補程式。請參閱安全性更新。

WCF 架構會自動選擇可用的最高版本通訊協定 (最高版本為 TLS 1.2)，除非您明確設定通訊協定版本。如需詳細資訊，請參閱先前的針對搭配憑證認證使用傳輸安全性的 WCF TCP 傳輸一節。

針對 .NET Framework 3.5 至 4.5.2 且非 WCF

我們建議您將應用程式升級至 .NET Framework 4.7 或更新版本。如果您無法升級，請採取下列步驟。

將 SchUseStrongCrypto 和 SystemDefaultTlsVersions 登錄機碼設定為 1。請參閱透過 Windows 登錄來設定安全性。.NET Framework 3.5 版只有在傳遞明確 TLS 值時，才會支援 `SchUseStrongCrypto` 旗標。

若您是執行 .NET Framework 3.5，便需要安裝修補程式，使您的程式可以指定 TLS 1.2：

KB3154518	[illegible] HR-1605 - [illegible] TLS [illegible] WINDOWS 7 SP1 [illegible] SERVER 2008 R2 SP1 [illegible] .NET FRAMEWORK 3.5.1 [illegible]
KB3154519	可靠性彙總套件 HR-1605 - 針對 TLS 系統預設版本的支援已包含在 Windows Server 2012 上的 .NET Framework 3.5 中
KB3154520	可靠性彙總套件 HR-1605 - 針對 TLS 系統預設版本的支援已包含在 Windows 8.1 和 Windows Server 2012 R2 的 .NET Framework 3.5 中
KB3156421	適用於 Windows 上的 .NET Framework 4.5.2 和 4.5.1 的 1605 Hotfix 彙總套件 3154521

針對使用搭配憑證認證使用 TCP 傳輸安全性之 .NET Framework 3.5 至 4.5.2 的 WCF

這些版本的 WCF 架構具有使用 SSL 3.0 和 TLS 1.0 值的硬式編碼。這些值不能變更。您必須更新並將目標重新設定為 NET Framework 4.6 或更新版本，以使用 TLS 1.1 和 1.2。

若應用程式是以 .NET Framework 3.5 作為目標

如果必須顯式設置安全協定，而不是讓 .NET 或作業系統選擇安全協定，請向代碼添加 `SecurityProtocolTypeExtensions` 和 `SslProtocolsExtension` 枚舉。`SecurityProtocolTypeExtensions` 和 `SslProtocolsExtension` 包含適用於 `Tls12`、`Tls11` 的值，以及 `SystemDefault` 值。有關詳細資訊，請參閱支援Windows 8.1 上的 .NET 框架 3.5 和 Windows 伺服器 2012 R2 中包含的 TLS 系統預設版本。

透過 AppContext 參數設定安全性 (適用於 .NET Framework 4.6 或更新版本)

只有在您的應用程式是以 .NET Framework 4.6 或更新版本為目標 (或是在其上執行) 的情況下，才會與描述於本節的 AppContext 參數具關連性。無論是根據預設設定，或是透過明確設定這些參數，它們在可能的情況下都應為 `false`。若您想要透過其中一個或同時透過這兩個參數來設定安全性，請不要在程式碼中指定安全性通訊協定，因為這麼做將會覆寫這些參數。

無論您是執行 HTTP 網路功能 (ServicePointManager) 或 TCP 通訊端網路功能 (SslStream)，這些參數都具有相同的效果。

Switch.System.Net.DontEnableSchUseStrongCrypto

將 `Switch.System.Net.DontEnableSchUseStrongCrypto` 設定為 `false` 值，會導致您的應用程式使用強式加密。將 `DontEnableSchUseStrongCrypto` 設定為 `false` 值，會使用更安全的網路通訊協定 (TLS 1.2、TLS 1.1 及 TLS 1.0)，並封鎖不安全的通訊協定。如需詳細資訊，請參閱 SCH_USE_STRONG_CRYPTO 旗標。設定為 `true` 值會停用應用程式的強式加密。

若應用程式是以 .NET Framework 4.6 或更新版本作為目標，此參數預設會設定為 `false`。那是安全的預設值，也是我們建議的選項。若您的應用程式是在 .NET Framework 4.6 上執行，但是以較舊的版本為目標，此參數預設會設定為 `true`。在此情況下，您應該明確地將它設定為 `false`。

若您需要連線至不支援強式加密且無法升級的舊版服務，則 `DontEnableSchUseStrongCrypto` 應該僅設定為 `true` 值。

Switch.System.Net.DontEnableSystemDefaultTlsVersions

將 `Switch.System.Net.DontEnableSystemDefaultTlsVersions` 設為 `false` 值，會導致您的應用程式允許作業系統選擇通訊協定。設定為 `true` 值會導致您的應用程式使用由 .NET Framework 所選取的通訊協定。

若應用程式是以 .NET Framework 4.7 或更新版本作為目標，此參數預設會設定為 `false`。這是建議的安全預設值。若您的應用程式是在 .NET Framework 4.7 或更新版本上執行，但是以較舊的版本為目標，此參數預設會設定為 `true`。在此情況下，您應該明確地將它設定為 `false`。

Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols

將 `Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols` 設定為 `false` 值，會導致您的應用程式針對使用憑證認證的訊息安全性，使用定義於 `ServicePointManager.SecurityProtocols` 中的值。設定為 `true` 值會使用可用的最高版本通訊協定 (最高版本為 TLS1.0)

針對以 .NET Framework 4.7 及更新版本為目標的應用程式，此值預設會設定為 `false`。針對以 .NET Framework 4.6.2 及較舊版本為目標的應用程式，此值預設會設定為 `true`。

Switch.System.ServiceModel.DontEnableSystemDefaultTlsVersions

將 `Switch.System.ServiceModel.DontEnableSystemDefaultTlsVersions` 設為 `false` 值，會設定預設設定以允許作業系統選擇通訊協定。設定為 `true` 值會將預設值設定為可用的最高版本通訊協定 (最高版本為 TLS1.2)。

針對以 .NET Framework 4.7.1 及更新版本為目標的應用程式，此值預設會設定為 `false`。針對以 .NET Framework 4.7 及較舊版本為目標的應用程式，此值預設會設定為 `true`。

如需 TLS 通訊協定的詳細資訊，請參閱風險降低：TLS 通訊協定。有關 `AppContext` 交換器的詳細資訊，請參閱 `<AppContextSwitchOverrides> Element`。

透過 Windows 登錄來設定安全性

> **WARNING**
> 設定登錄機碼會影響系統上的所有應用程式。只有當您具有機器的完整控制權，並且可以控制登錄上的變更時，才使用此選項。

如果您無法設定其中一個 `AppContext` 參數 (或是兩個都無法設定)，則可以透過本節中描述的 Windows 登錄機碼來控制應用程式所使用的安全性通訊協定。如果您的應用程式是在 .NET Framework 4.5.2 版或更早版本上執行，或是無法編輯組態檔，便可能無法使用其中一個 `AppContext` 參數，或是兩個都無法使用。若您想要透過登錄來設定安全性，請不要在程式碼中指定安全性通訊協定，因為這麼做將會覆寫這些登錄設定。

登錄機碼的名稱與相對應的 `AppContext` 參數類似，但名稱前方不會有 `DontEnable`。例如，`AppContext` 參數 `DontEnableSchUseStrongCrypto` 就是稱為 SchUseStrongCrypto 的登錄機碼。

這些機碼已透過近日的安全性修補程式，於所有 .NET Framework 版本中提供。請參閱安全性更新。

無論您是執行 HTTP 網路功能 (ServicePointManager) 或 TCP 通訊端網路功能 (SslStream)，下列所描述的登錄機碼都具有相同的效果。

SchUseStrongCrypto

`HKEY_LOCAL_MACHINE\SOFTWARE\[Wow6432Node\]Microsoft\.NETFramework\<VERSION>: SchUseStrongCrypto` 登錄機碼具有

DWORD 類型的值。將值設為 1 會導致您的應用程式使用強式加密。強式加密會使用更安全的網路通訊協定 (TLS 1.2、TLS 1.1 及 TLS 1.0)，並封鎖不安全的通訊協定。將值設為 0 會停用強式加密。如需詳細資訊，請參閱 SCH_USE_STRONG_CRYPTO 旗標。

若應用程式是以 .NET Framework 4.6 或更新版本作為目標，此機碼預設會設定為 1。這是建議的安全預設值。如果應用的目標是 .NET Framework 4.5.2 或早期版本，則金鑰預設為 0。在此情況下，您應該明確地將它的值設定為 1。

若您需要連線至不支援強式加密且無法升級的舊版服務，則此機碼的值應該僅設定為 0。

SystemDefaultTlsVersions

`HKEY_LOCAL_MACHINE\SOFTWARE\[Wow6432Node\]Microsoft\.NETFramework\<VERSION>: SystemDefaultTlsVersions` 登錄機碼具有 DWORD 類型的值。將值設為 1 會導致您的應用程式允許作業系統選擇通訊協定。將值設為 0 會導致您的應用程式使用由 .NET Framework 所選取的通訊協定。

`<VERSION>` 必須是 v4.0.30319 (針對 .NET Framework 4 及更新版本) 或 v2.0.50727 (針對 .NET Framework 3.5)。

若應用程式是以 .NET Framework 4.7 或更新版本作為目標，此機碼預設會設定為 1。這是建議的安全預設值。如果應用的目標是 .NET Framework 4.6.1 或早期版本，則金鑰預設為 0。在此情況下，您應該明確地將它的值設定為 1。

如需詳細資訊，請參閱 Windows 10 1511 版和 Windows Server 2016 Technical Preview 4 的累積更新：2016 年 5 月 10 日。

有關 .NET 框架 3.5.1 的詳細資訊，請參閱支援 Windows 7 SP1 和伺服器 2008 R2 SP1 上 .NET 框架 3.5.1 中包含的 TLS 系統預設版本。

下列 *.REG* 檔案會將登錄機碼及其變體設定為最安全的值：

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\.NETFramework\v2.0.50727]
"SystemDefaultTlsVersions"=dword:00000001
"SchUseStrongCrypto"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\.NETFramework\v4.0.30319]
"SystemDefaultTlsVersions"=dword:00000001
"SchUseStrongCrypto"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\v2.0.50727]
"SystemDefaultTlsVersions"=dword:00000001
"SchUseStrongCrypto"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\v4.0.30319]
"SystemDefaultTlsVersions"=dword:00000001
"SchUseStrongCrypto"=dword:00000001
```

在 Windows 登錄中設定安全通道通訊協定

您可以使用登錄以對您用戶端和/或伺服器應用程式交涉的通訊協定進行細微的控制。您應用程式的網路功能會透過安全通道 (英文) 進行。透過設定 `Schannel`，您便可以設定應用程式的行為。

請從 `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols` 登錄機碼開始。在該機碼底下，您可以在 `SSL 2.0`、`SSL 3.0`、`TLS 1.0`、`TLS 1.1` 及 `TLS 1.2` 集合中建立任何子機碼。在那些子機碼底下，您可以建立子機碼 `Client` 和/或 `Server`。在 `Client` 和 `Server` 底下，您可以建立 DWORD 值 `DisabledByDefault` (0 或 1) 和 `Enabled` (0 或 0xFFFFFFFF)。

The SCH_USE_STRONG_CRYPTO 旗標

啟用 `SCH_USE_STRONG_CRYPTO` 旗標時 (預設會由 `AppContext` 參數或 Windows 登錄啟用)，.NET Framework 會在您的應用程式要求 TLS 安全性通訊協定時使用此旗標。`SCH_USE_STRONG_CRYPTO` 旗標可以依預設啟用、搭配 `AppContext` 參數啟用，或是搭配登錄啟用。OS 會將旗標傳遞至 `Schannel`，以指示它停用已知的弱式加密演算法、加密套件，以及 TLS/SSL 通訊協定版本；若不這樣做，系統可能會為了取得更佳的互通性而啟用這些項目。如需詳細資訊，請參閱

- 安全通道 (英文)
- SCHANNEL_CRED 結構 (英文)

當您明確使用 SecurityProtocolType 或 SslProtocols 的 `Tls` (TLS 1.0)、`Tls11` 或 `Tls12` 列舉值時，`SCH_USE_STRONG_CRYPTO` 旗標也會傳遞至 `Schannel`。

安全性更新

本文中的最佳做法取決於所安裝的最新安全性更新。這些更新包括使用進階 .NET Framework 4.7 和更新版本功能的能力。如果您的應用程式是在 .NET Framework 4.7 和更新版本上執行，則最新的安全性更新十分重要 (即使應用程式是以舊版為目標)。

若要更新 .NET Framework 以允許作業系統選擇要使用的最佳 TLS 版本，您至少必須安裝：

- .NET Framework 2017 年 8 月品質彙總套件預覽 (英文)。
- **或者** .NET Framework 2017 年 9 月安全性與品質彙總套件 (英文)。

另請參閱：

- .NET Framework 版本和相依性
- 如何：確定哪些 .NET 框架版本已安裝。

支援 TLS 1.2

若要讓您的應用程式交涉 TLS 1.2，作業系統和 .NET Framework 版本都需要支援 TLS 1.2。

用以支援 TLS 1.2 的作業系統需求

若要在支援 TLS 1.2 和/或 TLS 1.1 的系統上啟用或重新啟用它們，請參閱傳輸層安全性 (TLS) 登錄設定。

tttt	TLS 1.2 tt
Windows 10 Windows Server 2016	支援，而且已預設為啟用。
Windows 8.1 Windows Server 2012 R2	支援，而且已預設為啟用。
Windows 8.0 Windows Server 2012	支援，而且已預設為啟用。
Windows 7 SP1 Windows Server 2008 R2 SP1	支援，但預設為不啟用。如需如何啟用 TLS 1.2 的詳細資訊，請參閱傳輸層安全性 (TLS) 登錄設定網頁。
Windows Server 2008	支援 TLS 1.2 和 TLS 1.1 需要更新。請參閱在 Windows Server 2008 SP2 中加入 TLS 1.1 和 TLS 1.2 支援的更新。
Windows Vista	不支援。

如需每個 Windows 版本上所預設啟用 TLS/SSL 通訊協定的相關資訊，請參閱 TLS/SSL (Schannel SSP) 中的通訊協定 (英文)。

用以支援 TLS 1.2 與 .NET Framework 3.5 的需求

此表格顯示使用 .NET Framework 3.5 支援 TLS 1.2 所需的作業系統更新。建議您套用所有的作業系統更新。

□□□□	□□ .NET FRAMEWORK 3.5 □□ TLS 1.2 □□□□□□
Windows 10 Windows Server 2016	適用於 Windows 10 1511 版和 Windows Server 2016 Technical Preview 4 的累積更新：2016 年 5 月 10 日
Windows 8.1 Windows Server 2012 R2	支援 Windows 8.1 和 Windows Server 2012 R2 上 .NET Framework 3.5 所包含的 TLS 系統預設版本 (機器翻譯)
Windows 8.0 Windows Server 2012	支援 Windows Server 2012 上 .NET Framework 3.5 所包含的 TLS 系統預設版本 (機器翻譯)
Windows 7 SP1 Windows Server 2008 R2 SP1	支援 Windows 7 SP1 和 Server 2008 R2 SP1 上 .NET Framework 3.5.1 所包含的 TLS 系統預設版本 (機器翻譯)
Windows Server 2008	支援 Windows Vista SP2 和 Server 2008 SP2 上 .NET Framework 2.0 SP2 所包含的 TLS 系統預設版本 (機器翻譯)
Windows Vista	不支援

使用安全通訊端層

2020/3/20 • • Edit Online

System.Net 類別會使用安全通訊端層 (SSL) 來加密數個網路通訊協定的連線。

若為 HTTP 連線，WebRequest 和 WebResponse 類別會使用 SSL 來與支援 SSL 的 Web 主機通訊。決定使用 SSL 與否是 WebRequest 類別根據給定的 URI 來進行。如果 URI 開頭是 "https:"，則使用 SSL。如果 URI 開頭是 "http:"，則使用未加密的連線。

若要搭配使用 SSL 與檔案傳輸通訊協定 (FTP)，請將 EnableSsl 屬性設定為 true，才能呼叫 GetResponse()。同樣地，若要搭配使用 SSL 與簡易郵件傳輸通訊協定 (SMTP)，請將 EnableSsl 屬性設定為 true，然後傳送電子郵件。

SslStream 類別提供 SSL 以資料流為基礎的抽象概念，並提供許多方式來設定 SSL 交握。

範例

程式碼

```
Dim MyURI As String = "https://www.contoso.com/"
Dim Wreq As WebRequest = WebRequest.Create(MyURI)

Dim serverUri As String = "ftp://ftp.contoso.com/file.txt"
Dim request As FtpWebRequest = CType(WebRequest.Create(serverUri), FtpWebRequest)
request.Method = WebRequestMethods.Ftp.DeleteFile
request.EnableSsl = True
Dim response As FtpWebResponse = CType(request.GetResponse(), FtpWebResponse)
```

```
String MyURI = "https://www.contoso.com/";
WebRequest WReq = WebRequest.Create(MyURI);

String serverUri = "ftp://ftp.contoso.com/file.txt"
FtpWebRequest request = (FtpWebRequest)WebRequest.Create(serverUri);
request.EnableSsl = true;
request.Method = WebRequestMethods.Ftp.DeleteFile;
FtpWebResponse response = (FtpWebResponse)request.GetResponse();
```

編譯程式碼

這個範例需要：

- 對 System.Net 命名空間的參考。

另請參閱

- 網路程式設計的安全性
- .NET 框架中的網路程式設計
- 憑證的選取和驗證

憑證的選取和驗證

2020/3/20 • • Edit Online

System.Net 類別支援數種方式來選取並驗證 System.Security.Cryptography.X509Certificates，以進行安全通訊端層 (SSL) 連線。用戶端可以選取一或多個憑證，以向伺服器驗證它自己。伺服器可能需要用戶端憑證具有一或多個特定屬性，來進行驗證。

定義

憑證是一個 ASCII 位元組資料流，包含公開金鑰、屬性 (例如版本號碼、序號和到期日) 以及來自憑證授權單位的數位簽章。憑證是用來建立加密連線，或向伺服器驗證用戶端。

用戶端憑證選取和驗證

用戶端可以選取特定 SSL 連線的一或多個憑證。用戶端憑證可以與網頁伺服器或 SMTP 郵件伺服器的 SSL 連線建立關聯。用戶端會將憑證新增至 X509Certificate 或 X509Certificate2 類別物件集合。使用電子郵件作為範例，憑證集合是與 SmtpClient 類別之 ClientCertificates 屬性建立關聯的 X509CertificateCollection 執行個體。HttpWebRequest 類別具有類似的 ClientCertificates 屬性。

X509Certificate 與 X509Certificate2 類別之間的主要差異，在於私密金鑰必須位在 X509Certificate 類別的憑證存放區中。

即使將憑證新增至集合，並與特定 SSL 連線建立關聯，也不會將任何憑證傳送至伺服器，除非伺服器要求它們。如果在連線上設定多個用戶端憑證，將會根據演算法來使用最佳用戶端憑證，而演算法考慮伺服器所提供的憑證簽發者清單與用戶端憑證簽發者名稱之間的相符項目。

SslStream 類別甚至可更進一步控制 SSL 交握。用戶端可以指定委派，以挑選要使用的用戶端憑證。

遠端伺服器可以確認用戶端憑證有效、最新並且是由適當的憑證授權單位所簽署。委派可以新增至 ServerCertificateValidationCallback，以強制執行憑證驗證。

用戶端憑證選擇

.NET Framework 使用下列方式選取要向伺服器呈現的用戶端憑證：

1. 如果先前向伺服器呈現用戶端憑證，則會在第一次呈現時快取憑證，並重複用於後續的用戶端憑證要求。
2. 如果具有委派，請一律使用委派結果作為可選取的用戶端憑證。可能的話，請嘗試使用快取的憑證；但是，如果委派已傳回 Null，而且憑證集合不是空的，則不會使用快取的匿名認證。
3. 如果這是用戶端憑證的第一個挑戰，Framework 會列舉與連線建立關聯之 X509Certificate 或 X509Certificate2 類別物件中的憑證，以尋找伺服器所提供的憑證簽發者清單與用戶端憑證簽發者名稱之間的相符項目。第一個符合的憑證會傳送至伺服器。如果沒有相符的憑證，或憑證集合是空的，則會將匿名認證傳送至伺服器。

憑證組態的工具

數個工具可進行用戶端和伺服器憑證組態。

Winhttpcertcfg.exe 工具可以用來設定用戶端憑證。*Winhttpcertcfg.exe* 工具隨附為 Windows Server 2003 Resource Kit 的其中一個工具。您可以在 www.microsoft.com 下載此工具，作為 Windows Server 2003 Resource Kit 工具的一部分。

HttpCfg.exe 工具可以用來設定 HttpListener 類別的伺服器憑證。*HttpCfg.exe* 工具隨附為 Windows Server 2003 和 Windows XP Service Pack 2 的其中一個支援工具。在 Windows Server 2003 或 Windows XP 上，預設不會安裝 *HttpCfg.exe* 和其他支援工具。在 Windows Server 2003 上。從 Windows Server 2003 CD-ROM 的下列資料夾和檔案中個別安裝支援工具：

\Support\Tools\Suptools.msi

若要與 Windows XP Service Pack 2 搭配使用，您可以從 www.microsoft.com 下載 Windows XP 支援工具。

HttpCfg.exe 工具版本的原始程式碼也會提供為 Windows Server SDK 的範例。在下列資料夾下，*HttpCfg.exe* 範例的原始程式碼預設會與網路範例一起安裝為 Windows SDK 的一部分：

C:\Program Files\Microsoft SDKs\Windows\v1.0\Samples\NetDS\http\serviceconfig

除了這些工具之外，X509Certificate 和 X509Certificate2 類別還會提供從檔案系統載入憑證的方法。

另請參閱

- 網路程式設計的安全性
- .NET 框架中的網路程式設計

網際網路驗證

2020/3/20 • • Edit Online

System.Net 類別支援各種不同的用戶端驗證機制，包括標準的網際網路驗證方法 (基本、摘要式、交涉式、NTLM 和 Kerberos 驗證) 以及您可以建立的自訂方法。

驗證認證儲存在 NetworkCredential 和 CredentialCache 類別中，能夠實作 ICredentials 介面。當這些類別的其中之一經查詢認證後，它會傳回 NetworkCredential 類別的執行個體。驗證程序是由 AuthenticationManager 類別管理，而實際的驗證程序是由實作 IAuthenticationModule 介面的驗證模組類別執行。您必須先向 AuthenticationManager 註冊自訂的驗證模組才能使用它，預設會註冊基本、摘要式、交涉式、NTLM 和 Kerberos 驗證方法模組。

NetworkCredential 儲存了一組與 URI 所識別之單一網際網路資源建立關聯的認證，並傳回它們以回應 GetCredential 方法的任何呼叫。存取有限數量網際網路資源的應用程式，或在所有情況下使用同一組認證的應用程式，通常會使用 NetworkCredential 類別。

CredentialCache 類別儲存各種 Web 資源認證的集合。呼叫 GetCredential 方法時，CredentialCache 會傳回適當的認證集，由 Web 資源 URI 和要求的驗證配置決定。使用各種網際網路資源和不同驗證配置的應用程式，得益於使用 CredentialCache 類別，因為它會儲存所有認證並依要求提供認證。

當網際網路資源要求驗證時，WebRequest.GetResponse 方法會將 WebRequest 以及認證要求傳送至 AuthenticationManager。然後根據下列程序驗證要求：

1. AuthenticationManager 會在每個已註冊的驗證模組上，依其註冊順序呼叫 Authenticate 方法。AuthenticationManager 使用不會傳回 null 的第一個模組執行驗證程序。程序的詳細資料隨所涉及的驗證模組類型而不同。
2. 完成驗證程序後，驗證模組會將 Authorization 傳回至 WebRequest，後者包含存取網際網路資源所需的資訊。

某些驗證配置可以驗證使用者，不需要先要求資源。應用程式可以預先驗證使用者和資源，至少消除一次伺服器往返，進而節省時間。或者，它可以在程式啟動期間執行驗證，以便稍後更能回應使用者。可以使用預先驗證的驗證配置，將 PreAuthenticate 屬性設為 true。

另請參閱

- 基本和摘要身份驗證
- NTLM 和 Kerberos 身份驗證
- 網路程式設計的安全性

基本和摘要式驗證

2020/3/20 • • Edit Online

基本和摘要式驗證的 System.Net 實作符合 RFC2617 - HTTP 驗證：基本和摘要式驗證 (可在全球資訊網協會的網站上取得)。

若要使用基本和摘要式驗證，應用程式必須在 WebRequest 物件的 Credentials 屬性中提供使用者名稱和密碼，以用來從網際網路要求資料，如下列範例所示。

```
Dim MyURI As String = "http://www.contoso.com/"
Dim WReq As WebRequest = WebRequest.Create(MyURI)
WReq.Credentials = New NetworkCredential(UserName, SecurelyStoredPassword)
```

```
String MyURI = "http://www.contoso.com/";
WebRequest WReq = WebRequest.Create(MyURI);
WReq.Credentials = new NetworkCredential(UserName, SecurelyStoredPassword);
```

Caution

使用基本與摘要式驗證傳送的資料不會經過加密，因此敵人可以看到資料。此外，基本驗證認證 (使用者名稱和密碼) 會以純文字傳送，而且可以被攔截。

另請參閱

- NTLM 和 Kerberos 身份驗證
- 網際網路驗證

NTLM 與 Kerberos 驗證

2020/3/20 • • Edit Online

預設 NTLM 驗證和 Kerberos 驗證使用與呼叫端應用程式建立關聯的 Microsoft Windows NT 使用者認證，以嘗試向伺服器進行驗證。使用非預設 NTLM 驗證時，應用程式會將驗證類型設為 NTLM，並使用 NetworkCredential 物件將使用者名稱、密碼和網域傳遞給主機，如下列範例所示。

```
Dim MyURI As String = "http://www.contoso.com/"
Dim WReq As WebRequest = WebRequest.Create(MyURI)
WReq.Credentials = _
    New NetworkCredential(UserName, SecurelyStoredPassword, Domain)
```

```
String MyURI = "http://www.contoso.com/";
WebRequest WReq = WebRequest.Create (MyURI);
WReq.Credentials =
    new NetworkCredential(UserName, SecurelyStoredPassword, Domain);
```

需要使用應用程式使用者認證來連線至網際網路服務的應用程式，可以利用使用者預設認證這麼做，如下列範例所示。

```
Dim MyURI As String = "http://www.contoso.com/"
Dim WReq As WebRequest = WebRequest.Create(MyURI)
WReq.Credentials = CredentialCache.DefaultCredentials
```

```
String MyURI = "http://www.contoso.com/";
WebRequest WReq = WebRequest.Create (MyURI);
WReq.Credentials = CredentialCache.DefaultCredentials;
```

交涉驗證模組可判斷遠端伺服器使用 NTLM 還是 Kerberos 驗證，並傳送適當的回應。

> **NOTE**
> NTLM 驗證未透過 Proxy 伺服器進行運作。

另請參閱

- 基本和摘要身份驗證
- 網際網路驗證

Web 和通訊端權限

2020/3/20 • • Edit Online

使用 System.Net 命名空間之應用程式的網際網路安全性是透過 WebPermission 和 SocketPermission 類別提供。WebPermission 類別可控制應用程式向 URI 要求資料，或提供 URI 給網際網路的權限。SocketPermission 類別則可控制應用程式根據通訊端的主機、連接埠編號和傳輸通訊協定，使用 Socket 在本機連接埠上接受資料，或使用另一個位址上的傳輸通訊協定連絡遠端裝置的權限。

使用的權限類別取決於您的應用程式類型。使用 WebRequest 和其子系的應用程式應使用 WebPermission 類別來管理權限。使用通訊端層級存取的應用程式應使用 SocketPermission 類別來管理權限。

WebPermission 和 SocketPermission 定義兩個權限：接受和連線。接受可授與應用程式回應來自另一方之連入連線的權限。連線可授與應用程式起始與另一方之連線的權限。

若是 SocketPermission 執行個體，接受表示應用程式可以在本機傳輸位址上接受連入連線；連線表示應用程式可以連線到某個遠端 (或本機) 傳輸位址。

若是 WebPermission 執行個體，接受表示應用程式可以將 WebPermission 所控制的 URI 匯出到全世界；連線表示應用程式可以存取該 URI (不論它是遠端或本機)。

另請參閱

- 安全性
- 網路程式設計的安全性

System.Net 類別的最佳作法

2020/3/20 • • Edit Online

下列建議將協助您善加利用 System.Net 中所含的類別：

- 如需傳輸層安全性 (TLS) 的最佳做法，請參閱 .NET Framework 的傳輸層安全性 (TLS) 最佳做法。
- 可能的話，請使用 WebRequest 和 WebResponse，而不是對子系類別進行類型轉換。使用 **WebRequest** 和 **WebResponse** 的應用程式可以利用新的網際網路通訊協定，而不需要大量的程式碼變更。
- 使用 **System.Net** 類別撰寫在伺服器上執行的 ASP.NET 應用程式時，其效能通常會比使用 GetResponse 和 GetResponseStream 的非同步方法還要好。
- 開啟至網際網路資源的連線數目，會對網路效能和輸送量造成明顯影響。**System.Net** 預設會為每個主機的每個應用程式使用兩個連線。在應用程式的 ServicePoint 中設定 ConnectionLimit 屬性，可能會針對特定主機增加這個數目。設定 ServicePointManager.DefaultPersistentConnectionLimit 屬性可以為所有主機增加這個預設值。
- 撰寫通訊端層級通訊協定時，請盡可能嘗試使用 TcpClient 或 UdpClient，而不是直接寫入至 Socket。這兩個用戶端類別會封裝 TCP 和 UDP 通訊端的建立，而不需要您處理連線的詳細資料。
- 存取需要認證的網站時，請使用 CredentialCache 類別建立認證的快取，而不是每個要求都提供它們。**CredentialCache** 類別會搜尋快取，以透過要求找到要呈現的適當認證，讓您不需要負責根據 URL 來建立和呈現認證。

另請參閱

- .NET 框架中的網路程式設計

透過 Proxy 存取網際網路

2020/3/20 • • Edit Online

如果您的網站使用 Proxy 來提供網際網路存取，則您必須設定 Proxy 執行個體，使您的應用程式能夠與 Web Proxy 進行通訊。

本節包含下列主題：

- 代理配置
- 自動 Proxy 偵測
- 如何：啟用 WebRequest 以使用 Proxy 與網際網路通訊
- 如何：覆寫全域 Proxy 的選取範圍

另請參閱

- 使用應用程式通訊協定
- .NET 框架中的網路程式設計

Proxy 組態

2020/3/20 • • Edit Online

Proxy 伺服器可處理資源的用戶端要求。Proxy 可從其快取傳回要求的資源，或將要求轉送至資源所在的伺服器。Proxy 可透過減少傳送至遠端伺服器的要求數目，來提升網路效能。Proxy 也可用來限制對資源的存取。

調適型 Proxy

.NET Framework 提供兩種 Proxy：調適型和靜態。調適型 Proxy 會在網路設定變更時調整其設定。例如，如果膝上型電腦使用者啟動撥接網路連線，調適型 Proxy 會認可這項變更、探索及執行其新的組態指令碼，然後適當地調整其設定。

調適型 Proxy 是由組態指令碼所設定 (請參閱自動 Proxy 偵測)。這個指令碼會產生一組應用程式通訊協定，並為每個通訊協定產生一個 Proxy。

網路環境的變更可能會要求系統使用一組新的 Proxy。如果網路連線中斷或已初始化新的網路連線，系統必須在新環境中找出組態指令碼的適當來源，並執行新的指令碼。

您可以使用設定檔中 `<proxy>` 元素 `usesystemdefault` 的屬性。 `usesystemdefault` 屬性控制是否應該從使用者的 Internet Explorer Proxy 設定讀取靜態 Proxy 設定 (Proxy 位址、略過清單和在本機上略過)。如果這個值設定為 `true`，則會使用來自 Internet Explorer 的靜態 Proxy 設定。如果這個值為 `false` 或未設定，則可以在組態中指定靜態 Proxy 設定，並且這個設定將覆寫 Internet Explorer Proxy 設定。若要啟用調適型 Proxy，也必須將這個值設定為 `false` 或不設定。

下列範例示範一般調適型 Proxy 組態。

```
<system.net>
    <defaultProxy>
      <proxy usesystemdefault="false" />
    </defaultProxy>
</system.net>
```

靜態 Proxy

靜態 Proxy 通常會由應用程式明確設定，或在應用程式或系統叫用組態檔時明確設定。靜態 proxy 可用於拓撲不常變更的網路中，例如連線至公司網路的桌面電腦。

您有數個選項可控制靜態 Proxy 的運作方式。您可以指定下列各項：

- Proxy 的位址。
- 是否應該讓本機位址略過 Proxy。
- 是否應該讓一組位址略過 Proxy。

下表顯示靜態 Proxy 的組態選項。

▯▯ (ATTRIBUTE)▯▯▯ (PROPERTY) ▯▯▯▯▯▯	▯▯
`proxyaddress` 或 Address	所要使用的 Proxy 位址。
`bypassonlocal` 或 BypassProxyOnLocal	控制本機位址是否要略過 Proxy。

屬性 (ATTRIBUTE)屬性 (PROPERTY) 屬屬屬屬	屬
`bypasslist` 或 BypassArrayList	使用規則運算式描述一組略過 Proxy 的位址。
`usesystemdefault`	控制是否應該從使用者的 Internet Explorer Proxy 設定讀取靜態 Proxy 設定 (Proxy 位址、略過清單和在本機上略過)。如果這個值設定為 `true`，則會使用來自 Internet Explorer 的靜態 Proxy 設定。在 .NET Framework 2.0 上，當這個值設定為 `true` 時，Internet Explorer Proxy 設定不會由組態檔中的其他 Proxy 設定覆寫。在 .NET Framework 1.1 上，Internet Explorer Proxy 設定可能會由組態檔中的其他 Proxy 設定覆寫。 如果這個值為 `false` 或未設定，則可以在組態中指定靜態 Proxy 設定；並且這個設定將覆寫 Internet Explorer Proxy 設定。若要啟用調適型 Proxy，也必須將這個值設定為 `false` 或不設定。

下列範例示範一般靜態 Proxy 組態。

```
<system.net>
    <defaultProxy>
        <proxy  proxyaddress="http://proxy.contoso.com:3128"
                bypassonlocal="true"
        />
        <bypasslist>
            <add address="[a-z]+.blueyonderairlines.com$" />
        </bypasslist>
    </defaultProxy>
</system.net>
```

另請參閱

- WebProxy
- GlobalProxySelection
- 自動 Proxy 偵測

自動 Proxy 偵測

2020/3/20 • • Edit Online

自動 Proxy 偵測是一種程序，透過此程序，系統會識別 Web Proxy 伺服器，並用來代表用戶端傳送要求。這項功能也稱為「Web Proxy 自動探索 (WPAD)」。啟用自動 Proxy 偵測時，系統會嘗試找出負責傳回可用於要求之這組 Proxy 的 Proxy 組態指令碼。如果找到 Proxy 組態指令碼，則會在針對使用 WebProxy 執行個體的要求取得 Proxy 資訊、要求資料流或回應時，在本機電腦上下載、編譯和執行指令碼。

自動 Proxy 偵測是透過 WebProxy 類別所執行，而且可以利用要求層級設定、組態檔中的設定，以及使用 Internet Explorer [區域網路 (LAN)]**** 對話方塊所指定的設定。

> **NOTE**
> 從 Internet Explorer 主功能表中選取 [工具]****，然後選取 [網際網路選項]****，即可顯示 Internet Explorer [區域網路 (LAN) 設定]**** 對話方塊。接下來，選取 [連線]**** 索引標籤，然後按一下 [區域網路設定]****。

啟用自動 Proxy 偵測時，WebProxy 類別會嘗試找到 Proxy 組態指令碼，如下所示：

1. WinINet `InternetQueryOption` 函式是用來尋找 Internet Explorer 最近偵測到的 Proxy 組態指令碼。
2. 如果找不到指令碼，則 WebProxy 類別會使用動態主機設定通訊協定 (DHCP) 來找到指令碼。DHCP 伺服器可以回應指令碼的位置 (主機名稱) 或指令碼的完整 URL。
3. 如果 DHCP 無法識別 WPAD 主機，則會查詢 DNS 中是否有 WPAD 為其名稱或別名的主機。
4. 如果找不到主機，但由 Internet Explorer 區域網路設定或組態檔指定 Proxy 組態指令碼的位置，則會使用此位置。

> **NOTE**
> 執行為 NT 服務或 ASP.NET 一部分的應用程式會使用叫用使用者的 Internet Explorer Proxy 伺服器設定 (可用時)。這些設定可能無法用於所有服務應用程式。

Proxy 是根據 connectoid 所設定。connectoid 是網路連線對話方塊中的項目，而且可以是實體網路裝置 (數據機或乙太網路卡) 或虛擬介面 (例如透過網路裝置執行的 VPN 連線)。connectoid 變更時 (例如，無線連線變更存取點，或啟用 VPN)，會再次執行 Proxy 偵測演算法。

預設會使用 Internet Explorer Proxy 設定來偵測 Proxy。如果應用程式在非互動式帳戶下運行（無需方便的方式配置 IE 代理設置），或者想要使用與 IE 設置不同的代理設置，則可以通過創建包含<預設代理>元素（網路設置）和<代理>元素（網路設置）元素的設定檔來配置代理。

針對您建立的要求，您可以搭配使用 Null Proxy 與您的要求來停用要求層級的自動 Proxy 偵測，如下列程式碼範例所示。

```
public static void DisableForMyRequest (Uri resource)
{
    WebRequest request = WebRequest.Create (resource);
    request.Proxy = null;
    WebResponse response = request.GetResponse ();
}
```

```
Public Shared Sub DisableForMyRequest(ByVal resource As Uri)
    Dim request As WebRequest = WebRequest.Create(resource)
    request.Proxy = Nothing
    Dim response As WebResponse = request.GetResponse()
    End Sub
```

沒有 Proxy 的要求會使用應用程式定義域的預設 Proxy，而其可在 DefaultWebProxy 屬性中取得。

另請參閱

- WebProxy
- WebRequest
- <system.Net>元素（網路設置）

如何：啟用 WebRequest 以使用 Proxy 與網際網路通訊

2020/3/20 • • Edit Online

這個範例會建立全域 Proxy 執行個體，可讓任何 WebRequest 使用 Proxy 與網際網路通訊。這個範例假設 Proxy 伺服器名為 `webproxy`，且在連接埠 80 (標準 HTTP 連接埠) 上進行通訊。

範例

```
var proxyObject = new WebProxy("http://webproxy:80/");
GlobalProxySelection.Select = proxyObject;
```

```
Dim proxyObject As New WebProxy("http://webproxy:80/")
GlobalProxySelection.Select = proxyObject
```

編譯程式碼

這個範例需要：

- System.Net命名空間 `using` 的 C# 指令。
- System.Net命名空間的可視基本 `Imports` 語句。

另請參閱

- 使用應用程式通訊協定
- 通過代理訪問互聯網

如何：覆寫全域 Proxy 的選取範圍

2020/3/20 • • Edit Online

這個範例會在連接埠 80 上將 **WebRequest** 傳送到 `www.contoso.com`，以使用名為 `alternateproxy` 的 Proxy 伺服器覆寫全域 Proxy 的選取範圍。

範例

```
WebRequest req = WebRequest.Create("http://www.contoso.com/");
req.Proxy = new WebProxy("http://alternateproxy:80/");
```

```
Dim req As WebRequest = WebRequest.Create("http://www.contoso.com/")
req.Proxy = New WebProxy("http://alternateproxy:80/")
```

編譯程式碼

這個範例需要：

- System.Net命名空間的 `using` 指令。

另請參閱

- 使用應用程式通訊協定
- 通過代理訪問互聯網

NetworkInformation

2020/3/20 • • Edit Online

System.Net.NetworkInformation 命名空間可讓您收集網路事件、變更、統計資料和屬性的相關資訊。您也可以使用 System.Net.NetworkInformation.Ping 類別，來判斷是否可連線至遠端主機。

網路可用性和事件

System.Net.NetworkInformation.NetworkChange 類別可讓您判斷網路位址或可用性是否已變更。若要使用此類別，請建立事件處理常式來處理變更，並將它與 NetworkAddressChangedEventHandler 或 NetworkAvailabilityChangedEventHandler 建立關聯。如需詳細資訊，請參閱**如何：偵測網路可用性和位址變更**。

網路統計資料和屬性

您可以根據介面或通訊協定來收集網路統計資料和屬性。NetworkInterface、NetworkInterfaceType 和 PhysicalAddress 類別提供特定網路介面的相關資訊，而 IPInterfaceProperties、IPGlobalProperties、IPGlobalStatistics、TcpStatistics 和 UdpStatistics 類別提供第 3 層和第 4 層封包的相關資訊。如需詳細資訊，請參閱**如何：取得介面和通訊協定資訊**。

判斷是否可以連線遠端主機

您可以使用 Ping 類別，判斷網路上的遠端主機是否啟動且可連線。如需詳細資訊，請參閱**如何：Ping 主機**。

另請參閱

- **網路程式設計範例**

如何：偵測網路可用性和位址變更

2020/3/20 • • Edit Online

這個範例示範如何偵測介面的網路位址變更。

範例

```
using System;
using System.Net;
using System.Net.NetworkInformation;

namespace Examples.Net.AddressChanges
{
    public class NetworkingExample
    {
        public static void Main()
        {
            NetworkChange.NetworkAddressChanged += new
             NetworkAddressChangedEventHandler(AddressChangedCallback);
            Console.WriteLine("Listening for address changes. Press any key to exit.");
            Console.ReadLine();
        }
        static void AddressChangedCallback(object sender, EventArgs e)
        {

            NetworkInterface[] adapters = NetworkInterface.GetAllNetworkInterfaces();
            foreach(NetworkInterface n in adapters)
            {
                Console.WriteLine("   {0} is {1}", n.Name, n.OperationalStatus);
            }
        }
    }
}
```

編譯程式碼

這個範例需要：

- 對 System.Net 命名空間的參考。

如何：取得介面和通訊協定資訊

2020/3/20 • • Edit Online

這個範例示範如何讀取網路介面的 TCP 統計資料。

範例

```
public static void ShowTcpStatistics(NetworkInterfaceComponent version)
{
    IPGlobalProperties properties =
          IPGlobalProperties.GetIPGlobalProperties();
    TcpStatistics tcpstat = null;
    Console.WriteLine("");
    switch (version)
    {
        case NetworkInterfaceComponent.IPv4:
             tcpstat = properties.GetTcpIPv4Statistics();
            Console.WriteLine("TCP/IPv4 Statistics:");
            break;
        case NetworkInterfaceComponent.IPv6:
            tcpstat = properties.GetTcpIPv6Statistics();
            Console.WriteLine("TCP/IPv6 Statistics:");
            break;
        default:
            throw new ArgumentException("version");
            break;
    }
    Console.WriteLine("  Minimum Transmission Timeout. : {0}",
        tcpstat.MinimumTransmissionTimeout);
    Console.WriteLine("  Maximum Transmission Timeout. : {0}",
        tcpstat.MaximumTransmissionTimeout);

    Console.WriteLine("  Connection Data:");
    Console.WriteLine("      Current : {0}",
    tcpstat.CurrentConnections);
    Console.WriteLine("      Cumulative : {0}",
        tcpstat.CumulativeConnections);
    Console.WriteLine("      Initiated  : {0}",
        tcpstat.ConnectionsInitiated);
    Console.WriteLine("      Accepted : {0}",
        tcpstat.ConnectionsAccepted);
    Console.WriteLine("      Failed Attempts : {0}",
        tcpstat.FailedConnectionAttempts);
    Console.WriteLine("      Reset : {0}",
        tcpstat.ResetConnections);

    Console.WriteLine("");
    Console.WriteLine("  Segment Data:");
    Console.WriteLine("      Received  ................... : {0}",
        tcpstat.SegmentsReceived);
    Console.WriteLine("      Sent : {0}",
        tcpstat.SegmentsSent);
    Console.WriteLine("      Retransmitted : {0}",
        tcpstat.SegmentsResent);

    Console.WriteLine("");
}
```

編譯程式碼

這個範例需要：

- 對 System.Net 命名空間的參考。

如何：Ping 主機

2020/3/20 • • Edit Online

這個範例示範如何 Ping 遠端主機。

範例

```
using System;
using System.Text;
using System.Net;
using System.Net.NetworkInformation;
using System.ComponentModel;
using System.Threading;

namespace Examples.System.Net.NetworkInformation.PingTest
{
    public class PingExample
    {
        public static void Main (string[] args)
        {
            if (args.Length == 0)
                throw new ArgumentException ("Ping needs a host or IP Address.");

            string who = args[0];
            AutoResetEvent waiter = new AutoResetEvent (false);

            Ping pingSender = new Ping ();

            // When the PingCompleted event is raised,
            // the PingCompletedCallback method is called.
            pingSender.PingCompleted += new PingCompletedEventHandler (PingCompletedCallback);

            // Create a buffer of 32 bytes of data to be transmitted.
            string data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
            byte[] buffer = Encoding.ASCII.GetBytes (data);

            // Wait 12 seconds for a reply.
            int timeout = 12000;

            // Set options for transmission:
            // The data can go through 64 gateways or routers
            // before it is destroyed, and the data packet
            // cannot be fragmented.
            PingOptions options = new PingOptions (64, true);

            Console.WriteLine ("Time to live: {0}", options.Ttl);
            Console.WriteLine ("Don't fragment: {0}", options.DontFragment);

            // Send the ping asynchronously.
            // Use the waiter as the user token.
            // When the callback completes, it can wake up this thread.
            pingSender.SendAsync(who, timeout, buffer, options, waiter);

            // Prevent this example application from ending.
            // A real application should do something useful
            // when possible.
            waiter.WaitOne ();
            Console.WriteLine ("Ping example completed.");
        }

        public static void PingCompletedCallback (object sender, PingCompletedEventArgs e)
```

```
        {
            // If the operation was canceled, display a message to the user.
            if (e.Cancelled)
            {
                Console.WriteLine ("Ping canceled.");

                // Let the main thread resume.
                // UserToken is the AutoResetEvent object that the main thread
                // is waiting for.
                ((AutoResetEvent)e.UserState).Set ();
            }

            // If an error occurred, display the exception to the user.
            if (e.Error != null)
            {
                Console.WriteLine ("Ping failed:");
                Console.WriteLine (e.Error.ToString ());

                // Let the main thread resume.
                ((AutoResetEvent)e.UserState).Set();
            }

            PingReply reply = e.Reply;

            DisplayReply (reply);

            // Let the main thread resume.
            ((AutoResetEvent)e.UserState).Set();
        }

        public static void DisplayReply (PingReply reply)
        {
            if (reply == null)
                return;

            Console.WriteLine ("ping status: {0}", reply.Status);
            if (reply.Status == IPStatus.Success)
            {
                Console.WriteLine ("Address: {0}", reply.Address.ToString ());
                Console.WriteLine ("RoundTrip time: {0}", reply.RoundtripTime);
                Console.WriteLine ("Time to live: {0}", reply.Options.Ttl);
                Console.WriteLine ("Don't fragment: {0}", reply.Options.DontFragment);
                Console.WriteLine ("Buffer size: {0}", reply.Buffer.Length);
            }
        }
    }
}
```

編譯程式碼

這個範例需要：

- 對 System.Net 命名空間的參考。

2.0 版中 System.Uri 命名空間的變更

2020/3/20 • • Edit Online

已對 System.Uri 類別進行數項變更。這些變更已修正不正確的行為、增強可用性和增強式安全性。

已淘汰和已取代的成員

建構函式:

- 所有具有 `dontEscape` 參數的建構函式。

方法:

- CheckSecurity
- Escape
- Canonicalize
- Parse
- IsReservedCharacter
- IsBadFileSystemCharacter
- IsExcludedCharacter
- EscapeString

變更

- 針對已知沒有查詢部分的 URI 配置 (file、ftp 和其他項目)，一律會逸出 '?' 字元，而且它不是 Query 部分的開頭。
- 針對隱含檔案 URI (形式為 `c:\directory\file@name.txt`)，除非要求完整取消逸出，或 LocalPath 是 `true`，否則一律會逸出片段字元 ('#')。
- 已移除 UNC 主機名稱支援;已採用用於呈現國際主機名稱的 IDN 規格。
- LocalPath 一律會傳回完整未逸出的字串。
- ToString 不會取消逸出已逸出的 '%'、'?' 或 '#' 字元。
- Equals 現在會在相等檢查時包含 Query 部分。
- 運算子 "==" 和 "!=" 會覆寫並連結至 Equals 方法。
- IsLoopback 現在會產生一致的結果。
- URI "`file:///path`" 不再轉譯成 `file://path`。
- "#" 現在會辨識為主機名稱結束字元。也就是 `http://contoso.com#fragment` 現在會轉換成 `http://contoso.com/#fragment`。
- 已修正合併基底 URI 與片段時的 Bug。
- 已修正 HostNameType 中的 Bug。

- 已修正 NNTP 剖析中的 Bug。
- HTTP:contoso.com 形式的 URI 現在會擲回剖析例外狀況。
- Framework 可正確處理 URI 中的 userinfo。
- 已修正 URI 路徑壓縮，因此中斷 URI 無法周遊檔案系統的根目錄。

另請參閱

- System.Uri

System.Uri 的國際資源識別項支援

2020/3/20 • • Edit Online

System.Uri 類別已擴充加上國際資源識別碼 (IRI) 和國際化網域名稱 (IDN) 支援。NET Framework 3.5、3.0 SP1 和 2.0 SP1 皆可使用這些增強功能。

IRI 和 IDN 支援

網址通常是使用統一資源識別碼 (URI) 來表示，而 URI 是由非常有限的字元組構成：

- 大寫和小寫的英文 ASCII 字母。
- 0 到 9 位數。
- 少數幾個其他 ASCII 符號。

網際網路工程任務推動小組 (IETF) 發行的 RFC 2396 和 RFC 3986 中有記載 URI 的規格。

隨著網際網路的成長，識別使用非英文語言資源的需求也持續成長。有利此需求且允許非 ASCII 字元 (Unicode/ISO 10646 字元集中的字元) 的識別碼，稱為國際資源識別碼 (IRI)。IETF 發行的 RFC 3987 中有記載 IRI 的規格。使用 IRI 可讓 URL 包含 Unicode 字元。

已擴充現有的 System.Uri 類別，根據 RFC 3987 提供 IRI 支援。目前的使用者除非特別啟用 IRI，否則看不出與 .NET Framework 2.0 的行為有任何不同之處。這可確保應用程式與舊版 .NET framework 相容。

應用程式可以指定是否使用網域名稱套用的國際化網域名稱 (IDN) 剖析功能，以及是否應套用 IRI 剖析規則。此作業可在 machine.config 或 app.config 檔案中完成。

啟用 IDN 會將網域名稱中所有的 Unicode 標籤轉換成對等的 Punycode。Punycode 名稱只包含 ASCII 字元，而且開頭一律為前置詞 xn--。這是為了支援網際網路上現有的 DNS 伺服器，因為大部分的 DNS 伺服器僅支援 ASCII 字元 (請參閱 RFC 3940)。

啟用 IRI 和 IDN 會影響 Uri.DnsSafeHost 屬性的值。啟用 IRI 和 IDN 也可以變更 Uri.Equals、Uri.OriginalString、Uri.GetComponents 和 IsWellFormedOriginalString 方法的行為。

System.GenericUriParser 類別也已經擴充，允許建立可自訂的剖析器，支援 IRI 和 IDN。指定 System.GenericUriParser 物件行為的方法，是將 System.GenericUriParserOptions 列舉中的可用值位元組合傳遞至 System.GenericUriParser 建構函式。GenericUriParserOptions.IriParsing 類型，指出支援 RFC 3987 指定的國際資源識別碼 (IRI) 剖析規則的剖析器。實際是否使用 IRI，取決於是否已啟用 IRI。

GenericUriParserOptions.Idn 類型，指出剖析器支援主機名稱的國際化網域名稱 (IDN) 剖析。實際是否使用 IDN，取決於是否已啟用 IDN。

啟用 URI 剖析會根據 RFC 3987 中最新的 IRI 規則來執行正規化和字元檢查。預設值專供要停用的 IRI 剖析使用，以便正規化和字元檢查根據 RFC 2396 和 RFC 3986 完成。

System.Uri 類別中的 IRI 和 IDN 處理也可以使用 System.Configuration.IriParsingElement 和 System.Configuration.IdnElement 組態設定類別來控制。System.Configuration.IriParsingElement 設定可啟用或停用 System.Uri 類別中的 IRI 處理。System.Configuration.IdnElement 設定可啟用或停用 Uri 類別中的 IDN 處理。System.Configuration.IriParsingElement 設定也可以間接控制 IDN。必須啟用 IRI 處理才能進行 IDN 處理。如果停用 IRI 處理，則 IDN 處理會設定為預設設定，此種情況下，.NET Framework 2.0 行為會用於相容性，但不使用 IDN 名稱。

建構第一個 System.Uri 類別時，System.Configuration.IriParsingElement 和 System.Configuration.IdnElement 組態類別的組態設定會讀取一次。該時間之後的組態設定變更會被忽略。

另請參閱

- System.Configuration.IdnElement
- System.Configuration.IriParsingElement
- System.Uri
- Uri.DnsSafeHost

3.5 版中的通訊端效能增強功能

2020/3/20 • • Edit Online

System.Net.Sockets.Socket 類別已經在版本 3.5 中增強，以供使用非同步網路 I/O 的應用程式使用，以達到最高的效能。一系列新類別已新增在 Socket 類別的一組增強功能當中，提供另一種非同步模式，可供專業化的高效能通訊端應用程式使用。這些增強功能專為需要高效能的網路伺服器應用程式而設計。應用程式可以獨佔方式，使用增強的非同步模式，或是只在其應用程式的目標熱區使用 (例如接收大量資料時)。

類別增強功能

這些增強功能的主要功能是在大量的非同步通訊端 I/O 期間，避免重複配置和同步處理物件。Socket 類別目前針對非同步通訊端 I/O 所實作的 Begin/End 設計模式，需要為每個非同步通訊端作業配置一個 System.IAsyncResult 物件。

在新的 Socket 類別增強功能中，應用程式所配置和維護的可重複使用 System.Net.Sockets.SocketAsyncEventArgs 類別物件會描述非同步通訊端作業。高效能通訊端應用程式最知道必須維持的重疊通訊端作業量。應用程式可以視需要建立許多 SocketAsyncEventArgs 物件。例如，如果伺服器應用程式需要隨時有 15 個通訊端接受作業，以支援內送的用戶端連線比率，它可以事先配置 15 個可重複使用的 SocketAsyncEventArgs 物件以用於此用途。

以這個類別執行非同步通訊端作業的模式，包含下列步驟：

1. 配置新 SocketAsyncEventArgs 內容物件，或從應用程式集區取得一個可用的內容物件。
2. 將內容物件上的屬性設為即將執行的作業 (例如回呼委派方法和資料緩衝處理)。
3. 呼叫適當的通訊端方法 (xxxAsync) 來啟始非同步作業。
4. 如果非同步通訊端方法 (xxxAsync) 在回呼中傳回 true，請查詢內容屬性以取得完成狀態。
5. 如果非同步通訊端方法 (xxxAsync) 在回呼中傳回 false，則作業以同步方式完成。可以查詢內容屬性以取得作業結果。
6. 重複使用內容進行另一個作業、將它放入集區，或是將它捨棄。

新非同步通訊端作業內容物件的存留期，取決於應用程式程式碼中的參考和非同步 I/O 參考。應用程式不需要在送出作為其中一個非同步通訊端作業方法的參數之後，保留對非同步通訊端作業內容物件的參考。在完成回呼傳回之前，它會一直被參考。不過讓應用程式保留內容物件的參考有好處，它可以重複用於未來的非同步通訊端作業。

另請參閱

- System.Net.Sockets.Socket
- System.Net.Sockets.SendPacketsElement
- System.Net.Sockets.SocketAsyncEventArgs
- System.Net.Sockets.SocketAsyncOperation
- 網路程式設計範例
- 通訊端程式碼範例

對等名稱解析通訊協定

2020/3/20 • • Edit Online

在對等環境中，對等使用特定名稱解析系統，才能從名稱或其他類型的識別碼解析彼此的網路位置 (位址、通訊協定和連接埠)。過去，網域名稱系統 (DNS) 內本來就是暫時性連線以及其他缺點，而讓對等名稱解析變得複雜。

Microsoft® Windows® 對等網路平台會利用對等名稱解析通訊協定 (PNRP) 解決此問題，此通訊協定最先是針對 Windows XP 所開發，之後於 Windows Vista™ 中升級，是一項安全、可調整的動態名稱登錄與名稱解析通訊協定。PNRP 的運作方式與傳統的名稱解析系統十分不同，為應用程式開發人員開拓令人雀躍的嶄新視野。

使用 PNRP，可以將對等名稱套用至電腦，或是電腦上的個別應用程式或服務。對等名稱解析包含位址、連接埠，以及可能的延伸承載。此系統的優點包含容錯、無瓶頸，以及絕不會傳回過時位址的名稱解析；這樣讓通訊協定成為尋找行動使用者的最佳解決方案。

就安全性而言，可以將對等名稱發行為安全的 (受保護) 或不安全的 (未受保護)。PNRP 使用公開金鑰加密來保護安全對等名稱防止詐騙；您可以使用 PNRP 來命名電腦和服務。

對等名稱解析通訊協定會示範下列屬性：

- 分散式且幾乎完全無伺服器。只有啟動程序處理序才需要伺服器。
- 沒有第三方的安全名稱發行。與 DNS 名稱發行不同，PNRP 名稱發行是瞬間發生的，沒有財務成本。
- PNRP 會即時更新，以防止解析過時位址。
- 透過 PNRP 的名稱解析同時允許服務的名稱解析，以擴充到不只限於電腦。

System.Net.PeerToPeer 命名空間

- PNRP 功能是由 .NET Framework 3.5 版內的 System.Net.PeerToPeer 命名空間所定義。它提供的一組類型可用來向可用的 PNRP 服務註冊和解析對等名稱。
- (PNRP 和自訂對等解析程式可以使用 System.ServiceModel.PeerResolvers 命名空間中所提供的類型進行建立和具現化)。
- 用來向可用 PNRP 服務註冊和解析名稱的基本類型如下：
- Cloud：定義描述可用 PNRP 雲端的資訊，包含其範圍。
- PeerName：定義可用來註冊並後續解析雲端內對等的對等名稱。
- PeerNameRecord：定義 PNRP 雲端中包含對等註冊資訊的記錄，而對等包含可連絡對等的網路端點。
- PeerNameRegistration：定義對等名稱的註冊處理序，包含啟動和停止對等名稱註冊的方法。
- PeerNameResolver：定義將對等名稱解析成其網路端點的處理序，同時包含解析的同步和非同步方法。

另請參閱

- System.ServiceModel.PeerResolvers
- System.Net.PeerToPeer
- 網路程式設計範例

對等名稱和 PNRP 識別碼

2020/3/20 • • Edit Online

對等名稱代表通訊端點，可以是一部電腦、一位使用者、一個群組、一項服務，或是與可解析成 IPv6 位址之對等建立關聯的任何項目。對等名稱解析通訊協定 (PNRP) 採用靜態上唯一的對等名稱來建立 PNRP 識別碼，以用來識別雲端成員。

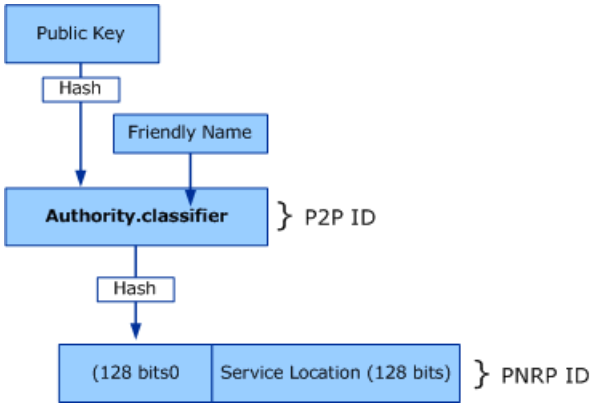
對等名稱

對等名稱可以註冊為不安全或安全的名稱。不安全的名稱是指單純為文字字串的名稱，容易受到詐騙，因為任何人都可以註冊重複的不安全名稱。不安全的名稱最好用於私人或是受保護的網路中。安全的名稱是使用憑證和數位簽章進行保護。只有原始發行者才能證明安全名稱的擁有權。

雲端與範圍的組合提供參與 PNRP 活動的對等相當安全的環境。不過，使用安全的對等名稱不保證網路應用程式的整體安全性。應用程式的安全性是與實作相依。

安全的對等名稱只能由其擁有者註冊，並以公開金鑰加密進行保護。安全的對等名稱是為由具有對應私密金鑰的對等實體所擁有。擁有權可以透過使用私密金鑰簽署的認證對等位址 (CPA) 進行證明。惡意使用者沒有對應的私密金鑰，就無法偽造對等名稱的擁有權。

PNRP 識別碼



PNRP 識別碼由下列各項所組成：

- 高序位 128 位元 (稱為對等 (P2P) 識別碼) 是指派給端點的對等名稱雜湊。對等名稱的格式如下：*Authority.Classifier*。安全名稱的 *Authority* 是以十六進位字元表示，是對等名稱公開金鑰的安全雜湊演算法 1 (SHA1) 雜湊。不安全名稱的 *Authority* 則為單一字元 "0"。*Classifier* 是識別應用程式的字串。對等名稱分類器的長度不能超過 149 個字元 (包含 `null` 結束字元)。
- 低序位 128 位元用於表示服務位置所產生的數字，用來識別相同雲端中同一個 P2P 識別碼的不同執行個體。

這個 P2P 識別碼與服務位置的組合可讓單一電腦登錄多個 PNRP 識別碼。

另請參閱

- PeerName
- System.Net.PeerToPeer

對等名稱發佈和解析

2020/3/20 • • Edit Online

發行對等名稱

為了發行新的 PNRP 識別碼，對等會執行下列動作：

- 將 PNRP 發行訊息傳送至相鄰的快取 (在快取的最下層中註冊 PNRP 識別碼的對等)，讓其快取成為種子。
- 在雲端中隨機選擇非相鄰的節點，並將其專屬 P2P 識別碼的 PNRP 名稱解析要求傳送至這些節點。產生的端點判斷處理序會以發行對等的 PNRP 識別碼，作為雲端中隨機節點的快取種子。

如果 PNRP 第 2 版僅解析其他 P2P 識別碼，則其節點不會發行 PNRP 識別碼。HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PeerNet\PNRP\IPV6-Global\SearchOnly=1 登錄值 (REG_DWORD 類型) 指定對等節點只能使用 PNRP 進行名稱解析，而不進行名稱發行。此登錄值也可以透過群組原則進行設定。

解析對等名稱

在 PNRP 網路或雲端中尋找其他對等，是包含兩個階段的處理序：

1. 端點判斷

2. PNRP 識別碼解析

在端點判斷階段中，嘗試解析另一部電腦上服務之 PNRP 識別碼的對等會判斷該遠端對等的 IPv6 位址。遠端對等是發行電腦或服務之 PNRP 識別碼的對等，或與之建立關聯的對等。

確認已將遠端端點註冊至 PNRP 雲端之後，PNRP 識別碼解析階段中發出要求的對等會將要求傳送至所需服務之 PNRP 識別碼的那個對等端點。端點會傳送回覆，確認服務的 PNRP 識別碼、註解，以及最多 4 KB 的額外資訊，以供發出要求的對等未來進行通訊時使用。例如，如果所需的端點是遊戲伺服器，則額外的對等名稱記錄資料可能會包含遊戲、參與遊戲等級和目前玩家人數的相關資訊。

在端點判斷階段中，PNRP 會使用互動的處理序尋找發佈 PNRP 識別碼的節點，其中執行解析的節點會負責連絡接近目標 PNRP 識別碼的後續節點。

為了在 PNRP 中執行名稱解析，對等節點會檢查本身的快取項目中是否有符合目標 PNRP 識別碼的項目。如果找到相符項目，對等會將 PNRP 要求訊息傳送至對等，並等候回應。如果找不到 PNRP 識別碼的項目，則對等會將 PNRP 要求訊息傳送至對應到最接近目標 PNRP 識別碼之 PNRP 識別碼的對等。收到 PNRP 要求訊息的節點會檢查自己的快取，並執行下列動作：

- 如果找到 PNRP 識別碼，則收到要求的對等會直接回覆發出要求的對等。
- 如果找不到 PNRP 識別碼，而快取中有接近目標 PNRP 識別碼的 PNRP 識別碼，則收到要求的對等會將回應傳送至發出要求的對等，其中包含代表擁有較接近目標 PNRP 識別碼之 PNRP 識別碼的對等 IPv6 位址。使用回應中的 IP 位址時，發出要求的節點會將另一個 PNRP 要求訊息傳送至 IPv6 位址來進行回應，或檢查其快取。
- 如果找不到 PNRP 識別碼，而且快取中沒有接近目標 PNRP 識別碼的 PNRP 識別碼，則收到要求的對等會將指出此情況的回應傳送給發出要求的對等。發出要求的對等接著會選擇下一個最接近的 PNRP 識別碼。

發出要求的對等會繼續重複執行此處理序，直到最後找到登錄 PNRP 識別碼的節點。

在 System.Net.PeerToPeer 命名空間內，包含端點以及在其中進行通訊之 PNRP 雲端或網格的 PeerName 記錄間有多對多關聯性。如果有重複或過時項目，或多個具有相同對等名稱的節點，則 PNRP 節點可以使用

PeerNameResolver 類別來取得目前資訊。PeerNameResolver 方法使用單一對等名稱，來簡化一個對等到多個對等名稱記錄以及相同的一個對等到許多雲端的觀點。這類似於使用關聯式資料表聯結所執行的查詢。成功完成時，解析程式物件會傳回所指定對等名稱的 PeerNameRecordCollection。例如，對等名稱發生在集合的所有對等名稱記錄中，並依雲端進行排序。這些是對等名稱的執行個體，而 PNRP 應用程式可以要求其支援資料。

另請參閱

- System.Net.PeerToPeer

PNRP 雲端

2020/3/20 • • Edit Online

PNRP「雲端」代表一組節點，可以透過網路彼此進行通訊。「雲端」這個詞相當於「對等網格」和「點對點圖形」。

節點之間的通訊絕對不應該跨越不同的雲端。Cloud 執行個體可透過其區分大小寫的名稱唯一進行識別。單一對等或節點可能已連線至多個雲端。

雲端極為緊密地繫結至網路介面。在有兩張網路卡連結至不同子網路的多重主目錄電腦上，將會傳回三個雲端：一個介面的一個連結本機位址有一個，以及單一全域範圍雲端。

PNRP 會使用三個雲端「範圍」，而範圍是一組可找到彼此的電腦：

- 全域雲端對應至全域 IPv6 位址範圍和全域位址，並代表整個 IPv6 網際網路上的所有電腦。單一全域雲端只有一個。
- 連結-本機雲端對應至連結-本機 IPv6 位址範圍與連結-本機位址。連結-本機雲端用於特定連結，而且通常與本機連接的子網路相同。可以有多個連結-本機雲端。

第三個雲端是網站特定雲端，並對應至網站 IPv6 位址範圍和網站-本機位址。此雲端已過時，不過 PNRP 中仍然支援此雲端。

雲端

PNRP 雲端是由 Cloud 類別的執行個體所表示。使用對等的雲端群組是由可列舉 CloudCollection 類別的執行個體所表示。呼叫靜態 GetAvailableClouds 方法，即可取得目前對等已知的 PNRP 雲端集合。

個別雲端具有唯一名稱，並以 256 個字元的 Unicode 字串呈現。這些名稱以及上述範圍是用來建構 Cloud 類別的唯一執行個體。這些執行個體可以序列化並重新建構以供持續使用。

建立或取得雲端執行個體之後，可以向它註冊對等名稱，以建立已知對等的網格。

另請參閱

- Cloud
- 對等名稱解析通訊協定

PNRP 快取

2020/3/20 • • Edit Online

對等名稱解析通訊協定 (PNRP) 快取是以演算法方式選取對等上所維護之對等端點的本機集合。

PNRP 快取初始化

若要在對等節點啟動時初始化 PNRP 快取或對等名稱記錄集合，節點可以使用下列方法：

- 節點關閉時存在的持續快取項目是從硬碟儲存體載入。
- 如果應用程式使用 P2P 共同作業基礎結構，則可以在該節點的連絡人管理員中取得共同作業資訊。

使用多層快取調整對等名稱解析

為了盡量縮小 PNRP 快取的大小，對等節點會使用多層快取，而每一層都包含最大項目數。快取中的每一層都代表 PNRP 識別碼空間的十分之一 (2^{256})。快取中的最下面一層包含在本機上註冊的 PNRP 識別碼，以及其他數字接近的 PNRP 識別碼。一層快取填滿最多 20 個項目之後，就會建立新的較低一層。快取中的最大層數是按照 log10 的順序而定 (雲端中 PNRP 識別碼的總數)。例如，對於含有 1 億個 PNRP 識別碼的全域雲端來說，快取中不會超過 8 (=log10(100,000,000)) 層，這與解析名稱時用來解析 PNRP 識別碼的躍點數目相似。此機制允許使用可針對其解析任意 PNRP 識別碼的分散式雜湊表，方法是將 PNRP 要求訊息轉送至下一個最接近的對等，直到找到包含對應 CPA 的對等為止。

為確保解析能夠完成，每次節點將項目新增至其快取的最下層時，都會將該項目的複本傳送到快取最後一層內的所有節點。

快取項目會隨著時間進行重新整理。過時的快取項目則會從快取中移除。因此 PNRP 識別碼的分散式雜湊表會以作用中的端點為主，而不像 DNS 中，位址記錄和 DNS 通訊協定無法保證與位址建立關聯的節點在網路上也是作用中的。

其他 PNRP 快取

另一個持續性資料存放區是本機快取。除了 PNRP 活動所需的其他物件之外，它還可以包含與 PNRP 雲端或共同作業工作階段建立關聯的記錄，而 PNRP 雲端或共同作業工作階段已在雲端的所有成員之間安全地進行發行和同步處理。這個複寫的存放區代表群組資料的檢視，而且應該適用於所有群組成員。技術上來說，這些物件本身不是記錄，而是以本機快取為目的的應用程式、目前狀態和物件資料。使用 PNRP 雲端，可確保將物件傳播到共同作業工作階段或 PNRP 雲端中的所有節點。雲端成員之間的記錄複寫會使用 SSL 來提供加密和資料完整性。

對等加入雲端時，不會自動收到來自所連結之主對等的本機快取資料；它們必須訂閱主對等，才能收到應用程式、目前狀態和物件資料的更新。初始同步處理之後，對等會定期重新同步處理其複寫的存放區，確保所有群組成員都一致具有相同的檢視。共同作業工作階段或是共同作業工作階段內的應用程式可能也會執行相同的函式。

開始雲端的共同作業工作階段之後，應用程式可以註冊對等，並開始使用雲端範圍所定義的安全性來發行其資訊。對等加入雲端時，會將雲端的安全性機制套用至對等，以指定它所參與的範圍。接著可以在雲端範圍內安全地發行其記錄。請注意，雲端範圍可能與共同作業應用程式範圍不同。

對等可以註冊要接收來自其他對等的物件。更新物件時，會通知共同作業應用程式，並將新的物件傳遞給應用程式的所有訂閱者。例如，群組聊天應用程式中的對等可以註冊要接收應用程式資訊，這樣會將所有交談記錄當成應用程式資料傳送給它。這可讓它監視雲端內的聊天活動。

另請參閱

- System.Net.PeerToPeer

應用程式開發中的 PNRP

2020/3/20 • • Edit Online

在 Windows Vista 中，網路應用程式可以透過簡化的 PNRP 應用程式設計介面 (API)，存取名稱發行和解析功能。

實作對等名稱解析通訊協定

有了簡化的 PNRP API 之後，不會明確指定雲端來註冊名稱和位址；PNRP 元件會自動判斷要加入的適當雲端，以及雲端內要發行的位址。

針對 Windows Vista 中高度簡化的 PNRP 名稱解析，PNRP 名稱現在已整合至 getaddrinfo() Windows 通訊端函式。若要使用 PNRP 將名稱解析為 IPv6 位址，應用程式可以使用 getaddrinfo() 函式解析完整網域名稱 (FQDN) name.prnp.net，其中 name 為要解析的對等名稱。pnrp.net 網域是 Windows Vista 中專為 PNRP 名稱解析保留的網域。

在 PeerToPeer 應用程式之間傳遞的訊息仍然是透過基礎架構進行處理，例如 PeerChannel 和 WCF 大型資料與資料流。

另請參閱

- System.Net.PeerToPeer

對等共同作業

2020/3/20 • • Edit Online

對等網路使用功能相當強大的電腦 (個人電腦)，而這些電腦存在於網際網路邊緣，並且不只是用於進行用戶端運算工作。現代個人電腦 (PC) 具有極快速的處理器、大量記憶體和大型硬碟，但在執行電子郵件和網頁瀏覽這類常見運算工作時並未完全利用到它們。現代電腦可以輕鬆地作為許多類型之應用程式的用戶端和伺服器 (對等)。

對等共同作業基礎結構是簡化的 Microsoft Windows 對等基礎結構實作，其利用 Windows Vista 和更新版本的平台中的「近端分享」服務。雖然它也可以服務網際網路端點或連絡人，但最適合用於「近端分享」服務在其上運作之子網路內具對等功能的應用程式。它會合併 Live Messenger 和其他 Live 感知應用程式所使用的常見連絡人管理員，以判斷連絡端點、可用性和目前狀態。

共同作業應用程式

典型對等共同作業的應用包含下列步驟：

- 對等可決定有興趣裝載共同作業工作階段之對等的身分識別
- 以某種方式傳送主持工作階段的要求，而且主對等同意管理共同作業活動。
- 主對等會邀請子網路上的連絡人 (包含要求者) 加入工作階段。
- 所有想要共同作業的對等都可能會將主對等新增至其連絡人管理員。
- 大部分的對等都會將邀請回應 (接受還是拒絕) 及時傳回給主對等。
- 所有要共同作業的對等都會訂閱主對等。
- 對等執行其初始共同作業活動的同時，主對等可能會將遠端對等新增至其連絡人管理員。它也會處理所有邀請回應，判斷接受者、拒絕者和未回答者。它可能會取消未回答者的邀請，或執行某個其他活動。
- 目前，主對等可以啟動與所有受邀對等的共同作業工作階段，或註冊具有共同作業基礎結構的應用程式。P2P 應用程式使用「對等共同作業基礎結構」和 System.Net.PeerToPeer.Collaboration 命名空間來協調遊戲、電子佈告欄、會議和無伺服器目前狀態應用程式的通訊。

對等網路安全性

在 Active Directory 網域中，網域控制站使用 Kerberos 提供驗證服務。在無伺服器對等環境中，對等必須提供自己的驗證。針對對等網路，任何節點都可以作為 CA，因此不需要每個對等之受信任根存放區中的根憑證。使用格式為 X.509 憑證的自我簽署憑證來提供驗證。這些憑證是由每個對等所建立，而每個對等都會產生公開金鑰/私密金鑰配對，以及使用私密金鑰所簽署的憑證。自我簽署憑證用來進行驗證，以及提供對等實體的相關資訊。與 X.509 驗證類似，對等網路驗證依賴追蹤回信任公開金鑰的憑證鏈。

另請參閱

- System.Net.PeerToPeer.Collaboration
- 關於 System.Net.PeerToPeer.Collaboration 命名空間

關於 System.Net.PeerToPeer.Collaboration 命名空間

2020/3/20 • • Edit Online

System.Net.PeerToPeer.Collaboration 命名空間提供使用對等共同作業基礎結構，可用來實作對等共同作業活動的類別和 API。

類別

用於對等共同作業活動之實作的主要類別如下：

- ContactManager，可用來儲存對等連絡人。
- 要在其中進行共同作業的 PeerApplication，例如遊戲、交談用戶端或會議解決方案。
- 將在活動中共同作業的同儕節點。這些同儕節點可以 PeerContact、PeerNearMe 或 PeerEndPoint 物件表示。
- 靜態 PeerCollaboration 類別本身，其指定哪些應用程式可用以及哪些同儕節點會參與它們。

Invite 方法可用來邀請同儕節點加入共同作業工作階段。呼叫端同儕節點可以訂閱另一個同儕節點的事件，以指出附屬於共同作業工作階段的應用程式、物件或目前狀態資訊的更新。目前狀態類別指定 Peer 是否可用於共同作業，而 PeerScope 類別用來指定同儕節點允許的參與數目：Internet (全域)、NearMe (子網路) 或 None。

共同作業工作階段包含四個步驟：

- 探索。探索或發佈應用程式、同儕節點和目前狀態資訊。例如，找出本機子網路上已安裝相同遊戲的其他人。
- 邀請。傳送和接受遠端同儕節點的安全邀請，以啟動或加入 PeerCollaboration 工作階段。
- 管理連絡人。將探索到的同儕節點作為連絡人加入 ContactManager。
- 通訊。建立通訊時，請使用 System.Net API、System.Net.PeerToPeer API 或 Windows Communication Foundation 對等通道類別來進行多方通訊。

例如，主機同儕節點會啟動共同作業工作階段，並利用 CreateContact 方法，將遠端同儕節點和其中一個本機同儕節點加入主機同儕節點的連絡人管理員。接著，這三個使用者將參與自己的私用共同作業工作階段。

P2P 應用程式通常是：共同作業筆記記錄或白板的電話會議、無伺服器交談應用程式、互動式廣告以及線上遊戲工作階段。

另請參閱

- System.Net.PeerToPeer.Collaboration

對等網路案例

2020/3/20 • • Edit Online

對等網路可啟用或增強下列案例：

即時通訊 (RTC)

- 無伺服器即時通訊

  目前已有 RTC。現在，電腦使用者可以與其對等聊天，以及進行語音或視訊對話。不過，許多現有程式和其通訊協定都是依賴運作的伺服器。如果您要參與臨機操作無線網路或是隔離網路的一部分，則無法使用這些 RTC 功能。對等技術允許將 RTC 技術擴充到這些額外網路環境。

- 即時配對和玩遊戲

  與 RTC 類似，目前已有即時玩遊戲方式。有許多 Web 遊戲網站透過網際網路服務遊戲社群。它們提供的功能可尋找興趣類似的玩家並一起玩遊戲。問題是遊戲網站只存在於網際網路上，而且是專為想要挑戰世界最佳玩家的熱衷玩家所存在。這些網站會追蹤並提供可協助處理此處理序的統計資料。不過，這些網站不允許玩家在各種網路環境的朋友之間設定特定遊戲。對等網路可以提供這項功能。

共同作業

- 解決目標的專案工作區

  共用工作區應用程式允許建立臨機操作工作群組，然後允許工作群組擁有者將使用允許群組解決問題的工具和內容填入共用工作區中。這可能包含留言板、生產力工具和檔案。

- 與其他人共用檔案

  專案工作區共用的子集是共用檔案的能力。雖然這項功能只存在於目前的 Windows 版本中，但是可透過對等網路予以增強，以簡單易懂的方式提供檔案內容。允許輕鬆存取網際網路邊緣或臨機操作運算環境中的大量內容，可提高網路運算的價值。

- 共用體驗

  無線連線變得更為普遍之後，對等網路可讓您在一組對等中連線而且可以共用您所發生的體驗 (例如日落、搖滾音樂會或郵輪假期)。

內容發佈

- 簡訊

  對等網路可以允許以檔案或訊息形式將文字型資訊散佈給一大群使用者。範例是新聞清單。

- 音訊和視訊

  對等網路也可以允許將音訊或視訊資訊散佈給一大群使用者 (例如大型音樂會或公司會議)。若要現在散發內容，您必須設定高容量伺服器來收集負載，以及將負載散發至數百位或數千位使用者。使用對等網路時，只有少數幾個對等會實際從集中式伺服器取得其內容。這些對等會用這項資訊塞滿將它傳送給其他人的一些人，依此類推。散發內容的負載會散發到雲端中的對等。想要接收內容的對等會尋找最接近的散發對等，並從中取得內容。

- 產品更新的散發

  對等網路也可以提供有效的機制來散發軟體，例如產品更新 (安全性更新和 Service Pack)。連線至軟體散發

伺服器的對等可以取得產品更新，並將它傳播給其群組的其他成員。

分散式處理

- 工作的分割和散發

  一個大型運算工作可以先分成最適合對等之運算資源的數個較小的個別運算工作。對等可以執行大型運算工作的分割。然後，對等網路可以將個別工作散發給群組中的不同對等。每個對等都會執行其運算工作，並將其結果回報給集中式累積點。

- 電腦資源的彙總

  利用對等網路進行分散式處理的另一種方式是在每個對等上執行程式，而程式是在閒置處理器時間期間執行並且是中央伺服器所協調之較大運算工作的一部分。透過彙總多部電腦的處理器，對等網路可以將一組對等電腦轉換成進行大型運算工作的大型平行處理器。

另請參閱

- System.Net.PeerToPeer.Collaboration

3.5 SP1 版中 HttpWebRequest 之 NTLM 驗證的變更

2020/3/20 • • Edit Online

已在 .NET Framework 版本 3.5 SP1 和更新版本中進行安全性變更，這些變更會影響 HttpWebRequest、HttpListener、NegotiateStream 以及 System.Net 命名空間中的相關類別處理整合式 Windows 驗證的方式。這些變更可能會影響使用這些類別提出 Web 要求並接收回應的應用程式，而且其中使用根據 NTLM 的整合式 Windows 驗證。這項變更可能會影響設定成使用整合式 Windows 驗證的網頁伺服器和用戶端應用程式。

概觀

整合式 Windows 驗證的設計可讓某些認證回應成為通用，這表示可以重複使用或轉寄它們。如果不需要此特定設計功能，則驗證通訊協定應該執行目標特定資訊，以及通道特定資訊。服務隨後可以提供延伸保護，確保認證回應包含服務特定資訊 (例如服務主體名稱 (SPN))。在認證交換時利用此資訊，服務就能進一步免於惡意使用可能未正確取得的認證回應。

System.Net 和 System.Net.Security 命名空間中的多個元件都會代表呼叫端應用程式執行整合式 Windows 驗證。本節描述在使用整合式 Windows 驗證時新增擴充保護的 System.Net 元件變更。

變更

與整合式 Windows 驗證搭配使用的 NTLM 驗證程序包含目的電腦所發出並傳回給用戶端電腦的挑戰。除非連線是迴路連線 (例如，IPv4 位址 127.0.0.1)，否則電腦在收到它自己產生的挑戰時，驗證會失敗。

存取在內部網頁伺服器上執行的服務時，通常會使用與 `http://contoso/service` 或 `https://contoso/service` 類似的 URL 存取服務。"contoso" 名稱通常不是服務部署所在電腦的電腦名稱。使用 Active Directory、DNS、NetBIOS、本機電腦的 hosts 檔案 (例如，通常是 WINDOWS\system32\drivers\etc\hosts) 或本機電腦的 lmhosts 檔案 (例如，通常是 WINDOWS\system32\drivers\etc\lmhosts) 的 System.Net 和相關命名空間支援，以將名稱解析為位址。已解析名稱 "contoso"，以將傳送至 "contoso" 的要求傳送至適當的伺服器電腦。

針對大型部署進行設定時，也經常會將單一虛擬伺服器名稱授與用戶端應用程式和終端使用者絕不會使用之基礎電腦名稱的部署。例如，您可能會呼叫 `www.contoso.com` 伺服器，但在內部網路上只需要使用 "contoso"。此名稱稱為用戶端 Web 要求中的主機標頭。根據 HTTP 通訊協定所指定，主機要求標頭欄位可指定所要求資源的網際網路主機和連接埠號碼。這項資訊取自使用者或參考資源所提供的原始 URI (通常是 HTTP URL)。在 .NET Framework 版本 4 上，用戶端也可以使用 Host 屬性來設定這項資訊。

AuthenticationManager 類別控制 WebRequest 衍生類別和 WebClient 類別所使用的 Managed 驗證元件 (「模組」)。AuthenticationManager 類別提供屬性來公開以 URI 字串編製索引的 AuthenticationManager.CustomTargetNameDictionary 物件，讓應用程式提供要在驗證期間使用的自訂 SPN 字串。

如果未設定 CustomTargetNameDictionary 屬性，版本 3.5 SP1 現在預設成指定 NTLM (NT LAN Manager) 驗證交換時 SPN 之要求 URL 中所使用的主機名稱。要求 URL 中所使用的主機名稱可能不同於用戶端要求的 System.Net.HttpRequestHeader 中所指定的主機標頭。要求 URL 中所使用的主機名稱可能不同於伺服器的實際主機名稱、伺服器的電腦名稱、電腦的 IP 位址或迴路位址。在這些情況下，Windows 會讓驗證要求失敗。若要解決這個問題，我們需要通知 Windows：用戶端要求的要求 URL 中所使用的主機名稱 (例如，"contoso") 實際上是本機電腦的替代名稱。

有數種可行的方法可讓伺服器應用程式解決這項變更。建議的方法是將要求 URL 中所使用的主機名稱對應至伺服器登錄中的 `BackConnectionHostNames` 機碼。`BackConnectionHostNames` 登錄機碼通常用來將主機名稱對應至迴路位址。步驟如下所列。

若要指定對應至迴路位址且可連線至本機電腦上網站的主機名稱，請遵循下列步驟：

1. 依序按一下 [開始] 和 [執行]，鍵入 regedit.exe，然後按一下 [確定]。
2. 在登錄編輯程式中，找到並按一下下列登錄機碼：

   ```
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\MSV1_0
   ```

3. 以滑鼠右鍵按一下 MSV1_0，並指向 [新增]，然後按一下 [多字串值]。
4. 輸入 `BackConnectionHostNames`，然後按 ENTER。
5. 以滑鼠右鍵按一下 `BackConnectionHostNames`，然後按一下 [修改]。
6. 在 [數值資料] 方塊中，鍵入主機名稱或本機電腦上網站的主機名稱 (要求 URL 中所使用的主機名稱)，然後按一下 [確定]。
7. 結束登錄編輯程式，然後重新啟動 IISAdmin 服務並執行 IISReset。

較不安全的因應措施是停用迴圈檢查，如 https://support.microsoft.com/kb/896861 中所述。這會停用反映攻擊的保護。因此，最好只將這組替代名稱限制為預期電腦實際使用的替代名稱。

另請參閱

- AuthenticationManager.CustomTargetNameDictionary
- System.Net.HttpRequestHeader
- HttpWebRequest.Host

具有擴充保護的整合式 Windows 驗證

2020/3/20 • • Edit Online

已建立的增強功能會影響 System.Net 中的 HttpWebRequest、HttpListener、SmtpClient、SslStream、NegotiateStream 和相關類別以及相關命名空間處理整合式 Windows 驗證的方式。為加強安全性，擴充保護已新增支援。

這些變更可能會影響使用這些類別提出 Web 要求並接收回應的應用程式，它們使用整合式 Windows 驗證。這項變更也會影響設定成使用整合式 Windows 驗證的網頁伺服器和用戶端應用程式。

這些變更也會影響使用這些類別提出其他類型要求並接收回應的應用程式，它們使用整合式 Windows 驗證。

支援擴充保護的變更僅供 Windows 7 和 Windows Server 2008 R2 的應用程式使用。舊版 Windows 無法使用擴充保護的功能。

概觀

整合式 Windows 驗證的設計可讓某些認證的挑戰回應成為通用的，這表示可以重複使用或轉寄它們。挑戰回應至少應該使用目標特定資訊建構，最好也能使用某些通道特定資訊。服務隨後可以提供擴充保護，確保認證的挑戰回應包含服務特定資訊，例如服務主體名稱 (SPN)。在認證交換時利用此資訊，服務就能進一步免於惡意使用可能未正確使用的認證挑戰回應。

擴充保護旨在增強驗證通訊協定，以強化其降低驗證轉送攻擊的功能。它會圍繞著通道和服務繫結資訊的概念打轉。

整體目標如下：

1. 如已更新用戶端支援擴充保護，應用程式應該向所有支援的驗證通訊協定提供通道繫結和服務繫結資訊。有要繫結的通道 (TLS) 時，只能提供通道繫結資訊。應該一律提供服務繫結資訊。
2. 已正確設定的更新伺服器可能會在當它出現於用戶端驗證權杖時，驗證通道和服務繫結資訊，並在通道繫結不相符時，拒絕驗證嘗試。根據部署案例，伺服器可能會驗證通道繫結、服務繫結或兩者都驗證。
3. 更新的伺服器能夠接受或拒絕下層用戶端要求，這些要求不包含以原則為基礎的通道繫結資訊。

擴充保護所使用的資訊，是由下列兩個部分的其中之一或全部所組成：

1. 通道繫結權杖或 CBT。
2. 使用服務主體名稱或 SPN 格式的服務繫結資訊。

服務繫結資訊會指出用戶端意圖驗證特定的服務端點。用戶端與伺服器之間的通訊使用下列屬性：

- 以純文字格式執行用戶端驗證的伺服器必須能夠取得 SPN 值。
- SPN 的值是公用的。
- SPN 在傳輸中必須以密碼編譯方式保護，令攔截式攻擊無法插入、移除或修改其值。

CBT 是外部安全通道 (例如 TLS) 的屬性，透過內部、用戶端驗證型的通道用來繫結 (bind) 至交談。CBT 必須具有下列屬性 (也由 IETF RFC 5056 定義)：

- 有外部通道存在時，CBT 的值必須是能識別外部通道或伺服器端點的屬性，獨立抵達交談的用戶端和伺服器端。
- 用戶端傳送的 CBT 值絕不能是會受攻擊者影響的項目。

- 不保證 CBT 值的保密性。但這不表示，當帶有 CBT 的通訊協定可能加密它時，伺服器執行驗證以外的任何其他驗證一律可以檢查服務繫結的值以及通道繫結資訊。
- CBT 在傳輸中必須受到密碼編譯式的完整性保護，令攻擊者無法插入、移除或修改其值。

透過用戶端將 SPN 和 CBT 以防竄改方式傳送到伺服器，即完成通道繫結。伺服器驗證通道繫結資訊是否與其原則一致，拒絕它本身不認為是預期目標的驗證嘗試。如此一來，兩個通道就會以密碼編譯方式繫結在一起。

為保留現有用戶端和應用程式的相容性，您可設定伺服器允許尚不支援擴充保護的用戶端嘗試驗證。相較於「完全強化」組態，這稱為「部分強化」組態。

System.Net 和 System.Net.Security 命名空間中的多個元件都會代表呼叫端應用程式執行整合式 Windows 驗證。本節描述在使用整合式 Windows 驗證時新增擴充保護的 System.Net 元件變更。

Windows 7 目前支援的擴充保護。提供一種機制，讓應用程式可以判斷作業系統是否支援擴充保護。

支援擴充保護的變更

與整合式 Windows 驗證搭配使用的驗證程序，視所用的驗證通訊協定而定，通常包含目的地電腦所發出並傳回給用戶端電腦的挑戰。擴充的保護會在此驗證程序中新增功能

System.Security.Authentication.ExtendedProtection 命名空間支援應用程式使用擴充保護進行驗證。這個命名空間中的 ChannelBinding 類別代表通道繫結。命名空間的 ExtendedProtectionPolicy 類別代表伺服器用於驗證傳入用戶端連接的擴充保護原則。其他類別成員是搭配擴充保護使用。

就伺服器應用程式而言，這些類別包括下列：

ExtendedProtectionPolicy，具有下列項目：

- OSSupportsExtendedProtection 屬性，指出作業系統是否支援使用擴充保護的整合式 Windows 驗證。
- PolicyEnforcement 值，指出應該在何時強制執行延伸的保護原則。
- ProtectionScenario 值，指出部署案例。這會影響檢查擴充保護的方式。
- 選擇性的 ServiceNameCollection，其包含的自訂 SPN 清單，是用來比對用戶端提供為預期驗證目標的 SPN。
- 選擇性的 ChannelBinding，包含的自訂通道繫結可用於驗證。此案例不是常見案例

System.Security.Authentication.ExtendedProtection.Configuration 命名空間支援應用程式使用擴充保護的驗證組態。

已進行一些功能變更，以支援現有 System.Net 命名空間中的擴充保護。這些變更包括下列幾項：

- 新的 TransportContext 類別新增至表示傳輸內容的 System.Net 命名空間。
- HttpWebRequest 類別中的新 EndGetRequestStream 和 GetRequestStream 多載方法，可讓您擷取 TransportContext 以支援用戶端應用程式的擴充保護。
- 新增 HttpListener 和 HttpListenerRequest 類別以支援伺服器應用程式。

已進行一項功能變更，以支援現有 System.Net.Mail 命名空間中的 SMTP 用戶端應用程式的擴充保護：

- SmtpClient 類別中的 TargetName 屬性，代表使用 SMTP 用戶端應用程式的擴充保護時，用於驗證的 SPN。

已進行一些功能變更，以支援現有 System.Net.Security 命名空間中的擴充保護。這些變更包括下列幾項：

- NegotiateStream 類別中的新 BeginAuthenticateAsClient 和 AuthenticateAsClient 多載方法，允許傳送 CBT 以支援用戶端應用程式的擴充保護。
- NegotiateStream 類別中的新 BeginAuthenticateAsServer 和 AuthenticateAsServer 多載方法，允許傳送

ExtendedProtectionPolicy 以支援伺服器應用程式的擴充保護。

- SslStream 類別的新 TransportContext 屬性，支援用戶端和伺服器應用程式的擴充保護。

已新增 SmtpNetworkElement 屬性，支援 System.Net.Security 命名空間中 SMTP 用戶端擴充保護的組態。

用戶端應用程式的擴充保護

大部分用戶端應用程式的擴充保護支援都會自動發生。只要 Windows 基礎版本支援擴充保護，HttpWebRequest 和 SmtpClient 類別就支援擴充保護。HttpWebRequest 執行個體傳送從 Uri 建構的 SPN。根據預設，SmtpClient 執行個體會傳送從 SMTP 郵件伺服器主機名稱建構的 SPN。

針對自訂驗證，用戶端應用程式可以使用 HttpWebRequest 類別的 HttpWebRequest.EndGetRequestStream(IAsyncResult, TransportContext) 或 HttpWebRequest.GetRequestStream(TransportContext) 方法，允許擷取 TransportContext 和使用 GetChannelBinding 方法的 CBT。

HttpWebRequest 執行個體傳送至指定服務用於整合式 Windows 驗證的 SPN，可藉由設定 CustomTargetNameDictionary 屬性予以覆寫。

TargetName 屬性可用來設定自訂的 SPN，用於 SMTP 連線的整合式 Windows 驗證。

伺服器應用程式的擴充保護

HttpListener 會在執行 HTTP 驗證時，自動提供驗證服務繫結的機制。

最安全的案例是啟用 HTTPS:// 前置詞的擴充保護。在本例中，將 HttpListener.ExtendedProtectionPolicy 設定成將 PolicyEnforcement 設成 WhenSupported 或 Always 的 ExtendedProtectionPolicy，以及 ProtectionScenario 設成 TransportSelected。值 WhenSupported 會將 HttpListener 置於部分強化模式中，而 Always 則對應到完全強化模式。

在此組態中，當透過外部安全通道向伺服器提出要求時，會查詢外部通道是否有通道繫結。此通道繫結會傳遞給驗證 SSPI 呼叫，確認驗證 Blob 中的通道繫結是否相符。有三個可能的結果：

1. 伺服器的基礎作業系統不支援擴充的保護。要求不會向應用程式公開，且未經授權的 (401) 回應會傳回至用戶端。記錄到 HttpListener 追蹤來源的訊息會指定失敗的原因。
2. SSPI 呼叫失敗，表示可能是用戶端指定的通道繫結不符合從外部通道擷取的預期值，或針對 Always 在伺服器上設定擴充保護原則時，用戶端無法提供通道繫結。這兩種情況下，要求都不會向應用程式公開，且未經授權的 (401) 回應會傳回至用戶端。記錄到 HttpListener 追蹤來源的訊息會指定失敗的原因。
3. 用戶端指定了正確的通道繫結，或允許連接但未指定通道繫結，因為伺服器上使用 WhenSupported 設定擴充保護原則。要求會傳回應用程式處理。不會自動執行服務名稱檢查。應用程式可選擇執行它自己使用 ServiceName 屬性的服務名稱驗證，但在這些情況下會重複。

如果應用程式進行自己的 SSPI 呼叫，執行以在 HTTP 要求主體內來回傳遞的 Blob 為基礎的驗證，而且想要支援通道繫結，它需要從使用 HttpListener 的外部安全通道擷取預期的通道繫結，才能將它傳遞給原生的 Win32 AcceptSecurityContext 函式。若要這樣做，請使用 TransportContext 屬性並呼叫 GetChannelBinding 方法來擷取 CBT。僅支援端點繫結。如指定任何其他 Endpoint，就會擲回 NotSupportedException。如果基礎作業系統支援通道GetChannelBinding綁定，該方法將返回一個ChannelBindingSafeHandle指向通道綁定的包裝指標，該指標適用于作為 `pInput` 在參數中傳遞的 SecBuffer 結構的 pvBuffer 成員傳遞到AcceptSecurityCoNtext函數。Size 屬性包含通道繫結的長度，以位元組為單位。如果基礎作業系統不支援通道繫結，此函式會傳回 `null`。

另一個可能的案例是，在不使用 Proxy 的情況下，啟用 HTTP:// 前置詞的擴充保護。在本例中，將 HttpListener.ExtendedProtectionPolicy 設定成將 PolicyEnforcement 設成 WhenSupported 或 Always 的 ExtendedProtectionPolicy，以及 ProtectionScenario 設成 TransportSelected。值 WhenSupported 會將 HttpListener 置於部分強化模式中，而 Always 則對應到完全強化模式。

會根據已向 HttpListener 登錄的前置詞建立所允許服務名稱的預設清單。此預設清單可透過 DefaultServiceNames 屬性檢查。如果這份清單不完整，應用程式可以在建構函式中針對會使用的 ExtendedProtectionPolicy 類別指定自訂的服務名稱集合，而不是預設的服務名稱清單。

在此組態中，向沒有外部安全通道的伺服器提出要求時，通常會在沒有通道繫結檢查的情況下繼續驗證。如果驗證成功，會查詢內容是否有用戶端提供的服務名稱，用以比對驗證可接受的服務名稱清單。有四個可能的結果：

1. 伺服器的基礎作業系統不支援擴充的保護。要求不會向應用程式公開，且未經授權的 (401) 回應會傳回至用戶端。記錄到 HttpListener 追蹤來源的訊息會指定失敗的原因。

2. 用戶端的基礎作業系統不支援擴充的保護。在 WhenSupported 組態中，該驗證嘗試會成功，且要求會傳回給應用程式。在 Always 組態中，驗證嘗試會失敗。要求不會向應用程式公開，且未經授權的 (401) 回應會傳回至用戶端。記錄到 HttpListener 追蹤來源的訊息會指定失敗的原因。

3. 用戶端基礎作業系統支援擴充的保護，但應用程式未指定服務繫結。要求不會向應用程式公開，且未經授權的 (401) 回應會傳回至用戶端。記錄到 HttpListener 追蹤來源的訊息會指定失敗的原因。

4. 用戶端指定了服務繫結。比較服務繫結和允許的服務繫結清單。如果相符，則要求會傳回至應用程式。否則，要求不會向應用程式公開，且未經授權的 (401) 回應會自動傳回至用戶端。記錄到 HttpListener 追蹤來源的訊息會指定失敗的原因。

如果使用可接受服務名稱允許清單的這個簡單方法不足，應用程式可以查詢 ServiceName 屬性，提供它自己的服務名稱驗證。在上述的 1 和 2 案例中，屬性會傳回 `null`。在案例 3 中，它會傳回空字串。在案例 4 中，會傳回用戶端指定的服務名稱。

伺服器應用程式也可以使用這些擴充的保護功能，搭配其他類型的要求進行驗證，以及在使用受信任的 Proxy 時進行驗證。

另請參閱

- System.Security.Authentication.ExtendedProtection
- System.Security.Authentication.ExtendedProtection.Configuration

使用 IPv6 和 Teredo 的 NAT 周遊

2020/3/20 • • Edit Online

已進行支援網路位址轉譯 (NAT) 周遊的增強功能。這些變更設計成與 IPv6 和 Teredo 搭配使用，但也適用於其他 IP 通道技術。這些增強功能會影響 System.Net 和相關命名空間中的類別。

這些變更可能會影響計劃使用 IP 通道技術的用戶端和伺服器應用程式。

支援 NAT 周遊的變更只適用於使用 .NET Framework 第 4 版的應用程式。舊版 .NET Framework 無法使用這些功能。

概觀

網際網路通訊協定第 4 版 (IPv4) 已將 IPv4 位址的長度定義為 32 位元。因此，IPv4 大約支援 40 億個唯一 IP 位址 (2^32)。因為網際網路上的電腦和網路裝置數目已在 1990 年代擴充，所以 IPv4 位址空間限制變得明顯。

數個用來擴充 IPv4 存留期的其中一個技術是部署 NAT，以允許單一唯一公用 IP 位址代表大量私人 IP 位址 (私人內部網路)。受 NAT 裝置保護的私人 IP 位址會共用單一公用 IPv4 位址。NAT 裝置可能是專用硬體裝置 (例如，便宜的無線存取點和路由器) 或執行服務來提供 NAT 的電腦。此公用 IP 位址的裝置或服務會在公用網際網路與私人內部網路之間轉譯 IP 位址封包。

此配置適用於在私人內部網路上執行的用戶端應用程式，而這些用戶端應用程式會將要求傳送給網際網路上的其他 IP 位址 (通常是伺服器)。NAT 裝置或伺服器可以保留用戶端要求的對應；因此，它在傳回回應時會知道傳送回應的位置。但針對在受 NAT 裝置保護且想要提供服務、接聽封包以及回應之私人內部網路中執行的應用程式，此配置會造成問題。這是專用於對等應用程式的情況。

IPv6 通訊協定已將 IPv4 位址的長度定義為 128 位元。因此，IPv6 支援 3.2 x 10^38 唯一位址 (2^128) 的大型 IP 位址空間。有了此大小的位址空間之後，每個連線至網際網路的裝置就可能獲指定唯一位址。但會發生問題。全世界大部分仍然都只使用 IPv4。特別的是，小型公司、組織和住家所使用的許多現有路由器和無線存取點都不支援 IPv6。而且，有些服務這些客戶的網際網路服務提供者不支援或尚未設定 IPv6 支援。

已開發數個 IPv6 轉換技術，對 IPv4 封包中的 IPv6 位址進行通道處理。這些技術所包含的 6to4、ISATAP 和 Teredo 通道在 IPv6 主機必須周遊 IP4 網路以連線至其他 IPv6 網路時，提供單點傳播 IPv6 流量的位址指派以及主機對主機自動通道。IPv6 封包是以通道方式傳送為 IPv4 封包。將會使用數個通道技術來允許透過 NAT 裝置進行 IPv6 位址的 NAT 周遊。

Teredo 是其中一種 IPv6 轉換技術，具有與 IPv4 網路的 IPv6 連線。網際網路工程任務推動小組 (IETF) 發行的 RFC 4380 中已記載 Teredo。Windows XP SP2 和更新版本所支援的虛擬 Teredo 配接器可以提供 2001:0::/32 範圍中的公用 IPv6 位址。這個 IPv6 位址可以用來接聽來自網際網路的連入連線，並且可以提供給具 IPv6 功能且想要連線至接聽端服務的用戶端。這讓應用程式不用擔心如何處理受 NAT 裝置保護的電腦，因為應用程式只能使用其 IPv6 Teredo 位址與之連線。

支援 NAT 周遊和 Teredo 的增強功能

在 System.Net、System.Net.NetworkInformation 和 System.Net.Sockets 命名空間中新增增強功能，以支援使用 IPv6 和 Teredo 進行 NAT 周遊。

System.Net.NetworkInformation.IPGlobalProperties 類別中已新增數個方法，來取得主機上的單點傳播 IP 位址清單。BeginGetUnicastAddresses 方法會開始非同步要求，以擷取本機電腦上的穩定單點傳播 IP 位址表格。EndGetUnicastAddresses 方法會結束暫止非同步要求，以擷取本機電腦上的穩定單點傳播 IP 位址表格。GetUnicastAddresses 方法是同步要求，以擷取本機電腦上的穩定單點傳播 IP 位址表格，並視需要等到位址表格穩定為止。

IPAddress.IsIPv6Teredo 屬性可以用來判斷 IPAddress 是否為 IPv6 Teredo 位址。

搭配使用這些新 IPGlobalProperties 類別方法與 IsIPv6Teredo 屬性，可讓應用程式輕鬆地找到 Teredo 位址。如果應用程式將這項資訊與遠端應用程式進行通訊，則通常只需要知道本機 Teredo 位址。例如，對等應用程式可能會將其所有 IPv6 位址傳送給配對伺服器，而配對伺服器接著可以將它們轉送給其他對等來啟用直接通訊。

應用程式通常應該設定其接聽服務接聽 IPAddress.IPv6Any，而不是本機 Teredo 位址。因此，如果遠端用戶端或對等具有接聽服務主機的直接 IPv6 路由，則用戶端或對等可以使用 IPv6 直接連線，而不需要使用 Teredo 對封包進行通道處理。

針對 TCP 應用程式，System.Net.Sockets.TcpListener 類別具有 AllowNatTraversal 方法來啟用 NAT 周遊。針對 UDP 應用程式，System.Net.Sockets.UdpClient 類別具有 AllowNatTraversal 方法來啟用 NAT 周遊。

針對使用 System.Net.Sockets.Socket 和相關類別的應用程式，可以搭配使用 GetSocketOption 和 SetSocketOption 方法與 SocketOptionName.IPProtectionLevel 通訊端選項來查詢、啟用或停用 NAT 周遊。

另請參閱

- IPAddress.IsIPv6Teredo
- IPGlobalProperties.BeginGetUnicastAddresses
- IPGlobalProperties.EndGetUnicastAddresses
- IPGlobalProperties.GetUnicastAddresses
- System.Net.Sockets.IPProtectionLevel
- Socket.SetIPProtectionLevel
- TcpListener.AllowNatTraversal
- UdpClient.AllowNatTraversal

Windows 市集應用程式的網路隔離

2020/3/20 • • Edit Online

中的 類 和System.Net.Http.Headers命名空間中可用於開發 Windows 應用商店應用或桌面應用。System.Net System.Net.Http 在 Windows 應用商店應用中使用時，這些命名空間中的類受網路隔離的影響，網路隔離是 Windows 8 使用的應用程式安全模型的一部分。必須在 Windows 市集應用程式的應用程式資訊清單中啟用適當的網路功能，讓系統允許網路存取。

網路隔離的檢查清單

使用此檢查清單，確定已設定 Windows 市集應用程式的網路隔離。

1. 決定應用程式所需的網路存取要求方向。這可以是輸出用戶端起始的要求或輸入未經要求的要求，或是這兩種網路要求類型的組合。
2. 判斷該應用程式將與其通訊的網路資源類型。應用程式可能需要與家用或工作場所網路上受信任的資源進行通訊。應用程式可能需要與網際網路上的資源進行通訊。應用程式可能需要存取這兩種類型的網路資源。
3. 在應用程式資訊清單中設定最低必要網路隔離功能。
4. 部署和執行應用程式，以使用針對疑難排解所提供的網路隔離工具對其進行測試。

有關如何配置網路功能和用於解決網路隔離故障的隔離工具的更多詳細資訊，請參閱 Windows 8.x 應用商店開發人員文檔中如何配置網路隔離功能。

另請參閱

- 連線至 Web 服務
- 網路隔離的方針和檢查清單
- 快速入門：使用 HttpClient 進行連線
- 如何使用 HttpClient 處理常式
- 如何保護 HttpClient 連線
- HttpClient 範例

網路程式設計範例

2020/3/20 • • Edit Online

本節包含可下載網路程式設計範例的描述和連結，而這些範例使用 System.Net、System.Net.Cache、System.Net.Configuration、System.Net.Mail、System.Net.Mime、System.Net.NetworkInformation、System.Net.Security、System.Net.Sockets 和相關命名空間中的類別。

> **NOTE**
> 所有範例均可從 .NET Framework SDK 2.0 版範例下載中取得，此連結是針對 .NET Framework 2.0 所發行，且可能會過時。

本節內容

下載進度指示器技術範例
示範如何顯示檔案下載進度。

FTP 用戶端技術範例
示範如何在 FTP 伺服器中上傳和下載檔案。

HttpListener 技術範例
示範如何處理應用程式中的 HTTP 要求。

HttpListener ASPX 主機應用程式示例演示如何使用System.Net.HttpListener類的功能創建將調用路由到託管ASP.NET應用程式的 HTTP 伺服器。

Mailer 技術範例
示範如何從用戶端應用程式傳送電子郵件訊息。

NetStat 工具技術範例
示範 NCLNetStat 網路資訊工具。

網路資訊技術範例
示範如何監視和顯示網路資訊。

Ping 用戶端技術範例
示範可 Ping 遠端主機的用戶端應用程式。

WebClient 技術範例
示範如何執行一般作業，例如上傳或下載檔案或資料。

安全資料流範例
示範如何使用安全資料流以在用戶端與伺服器之間進行通訊。

IPv6 通訊端範例
示範如何在啟用 IPv6 時使用通訊端。

FTP 總管技術範例
示範如何列出 FTP 伺服器的內容。

參考

System.Net

System.Net.NetworkInformation

另請參閱

- .NET 框架中的網路程式設計
- 網路程式設計「如何」主題