* [C# 5.0 и .NET 4.5](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php)
* [WPF](https://professorweb.ru/my/WPF/base_WPF/level1/info_WPF.php)
* [Темы WPF](https://professorweb.ru/my/WPF/themes/DarkBlue_UI/info_themes.php)
* [Silverlight 5](https://professorweb.ru/my/Silverlight/base/level1/s_index.php)
* [Работа с БД](https://professorweb.ru/my/ADO_NET/base/level1/info_db.php)
* [LINQ](https://professorweb.ru/my/LINQ/base/level1/info_linq.php)
* [ASP.NET](https://professorweb.ru/my/ASP_NET/base/level1/aspnet_info.php)
* [Windows 8/10](https://professorweb.ru/my/windows8/rt/level1/info_w8.php)
* [Программы](https://professorweb.ru/my/programs/visual-studio/level1/info_programs.php)

**Пространство имен System.Net**

[C# и .NET](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php) --- [Сетевое программирование](https://professorweb.ru/my/csharp/web/level1/web_index.php) --- Пространство имен System.Net

Пространство имен **System.Net** содержит сетевые классы для поиска IP-адресов, сетевой аутентификации, разрешений, отправки и получения данных. Рассмотрим эти классы, рассортировав их по группам.

**Поиск имен**

Чтобы получить IP-адрес из DNS-имени хоста или получить имя хоста из IP-адреса, можно использовать **класс Dns**. **Класс DnsPermission** представляет разрешение, необходимое для поиска имени. **DnsPermissionAttribute** — это класс атрибута, позволяющий отмечать сборки, классы и методы, нуждающиеся в этих полномочиях.

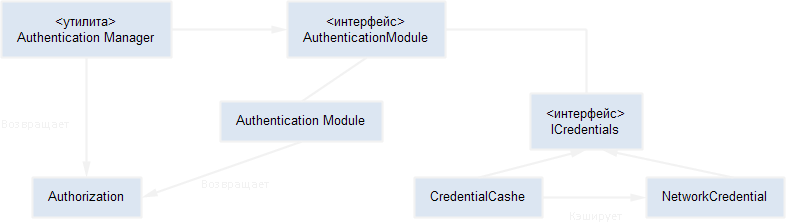
**IP-адреса**

IP-адреса обрабатываются в классе **IPAddress**. У одного хоста может быть несколько IP-адресов и алиасов. Вся эта информация содержится в классе **IPHostEntry**. Когда мы ищем имя, класс Dns возвращает объект типа IPHostEntry.

**Аутентификация и авторизация**

В классе **AuthenticationManager** есть статические методы, предназначенные для аутентификации клиентского пользователя. В этом классе-утилите используются модули, реализующие *интерфейс IAuthenticationModule*. Класс AuthenticationManager обращается к этим модулям, чтобы идентифицировать пользователя. Модули аутентификации получают информацию запроса и данные о личности пользователя с помощью *интерфейса ICredentials* и возвращают объект Authorization для авторизованных пользователей, которым разрешается использовать ресурс.

Приложение-клиент может передать данные о пользователе на сервер экземпляром класса **NetworkCredential**. Эти данные о пользователе могут быть занесены в кэш BCredentialCache:



**Запросы и ответы**

Абстрактные базовые классы, предназначенные для отправки запросов на сервер и получения ответов, называются *WebRequest* и *WebResponse*. В пространстве имен System.Net имеется несколько специальных реализаций этих классов для HTTP и доступа к файлам: HttpWebRequest, HttpWebResponse, FileWebRequest и FileWebResponse.

Классы HttpXXX также используют еще один класс из пространства имен System.Net: **класс HttpVersion** применяется для указания версии HTTP. У классов HttpWebRequest и HttpWebResponse есть *свойство ProtocolVersion*, определяющее версию HTTP - HttpVersion.Version10 или HttpVersion.Version11. Версия HTTP 1.0 использовалась на заре Интернета и продолжает использоваться некоторыми Web-серверами, текущая версия HTTP 1.1 включает некоторые дополнительные возможности, например, может поддерживать открытое соединение для нескольких запросов.

Разрешения, необходимые для классов запросов и ответов, определены в классе **WebPermission** и классе атрибута **WebPermissionAttribute**.

Класс компонентов WebClient облегчает использование WebRequest и WebResponse из Visual Studio .NET. Этот класс порожден от класса Component и поэтому может использоваться с функцией буксировки мышью из панели Toolbox. Однако по умолчанию он не сконфигурирован для использования Toolbox. С классом WebClient нетрудно копировать файлы с сервера и на сервер.

**Управление соединениями**

Классы **ServicePoint** и **ServicePointManager** играют важную роль для НТТР-соединений. Для ресурса экземпляр класса ServicePoint связан с URI и может обрабатывать несколько соединений. Класс-утилита *ServicePointManager* управляет объектами ServicePoint, создавая новые или отыскивая существующие объекты.

Пропускную способность приложения, которое одновременно запрашивает множество данных с сервера, можно повысить, увеличив число соединений на базе приложения. Чтобы ограничить выделение сетевых ресурсов, по умолчанию максимальное число соединений с одним и тем же сервером устанавливается равным двум.

**Записи cookies**

Cookies — это хранящиеся на стороне клиента наборы данных, которые используются сервером для запоминания некоторой информации между запросами. Когда для запросов данных с сервера используется такой Web-браузер, как Internet Explorer, он сам управляет приемом и сохранением записей cookies и посылает их обратно Web-cepвepy. При создании специального приложения, запрашивающего данные с Web-сервера, посылающего записи cookies, можно их считывать в объект класса **CookieCollection**, возвращаемый свойством Cookies объекта HttpWebResponse.

С помощью класса CookieContainer передаются записи cookie серверу. Сама запись cookie представлена в классе Cookie.

**Сокеты**

Вместо Web-классов используются классы сокетов, при этом мы, приобретая дополнительные возможности и гибкость, сталкиваемся с определенной сложностью. Большинство классов, которые используются в программировании сокетов, можно найти в пространстве имен **System.Net.Sockets**.

Программирование сокетов не только позволяет осуществлять связь, ориентированную на соединения, как в случае с HTTP, но также и реализовывать связь без установления соединений, которая используется при групповой рассылке или широковещательной передаче с UDP. Программирование сокетов — чрезвычайно гибкий механизм, позволяющий пользоваться самыми разными протоколами: GGP, ICMP, IGMP, IPX, SPX.

Обзор наиболее важных классов из пространства имен System.Net закончен, и теперь более детально рассмотрим некоторые из этих классов.

**Работа с URI**

[C# и .NET](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php) --- [Сетевое программирование](https://professorweb.ru/my/csharp/web/level1/web_index.php) --- Работа с URI

Каждый день мы используем ***универсальные идентификаторы ресурса (Uniform Resource Identifiers, URI)***, когда что-то ищем в WWW. URI нужны, чтобы идентифицировать и запросить новый вид ресурса. Используя URI, можно обращаться не только к Web-страницам, но и к FTP-серверу, Web-сервису и локальным файлам.

Вместо URI часто используется термин *унифицированный указатель ресурса (Uniform Resource Locator, URL)*. URI-общий термин, используемый для ссылок на ресурсы. URL - это URI, связанный с такими популярными схемами URI, как http, ftp и mailto. В технической документации термин URL больше не употребляется.

Еще один термин может быть вам уже известен - *унифицированное имя ресурса (Uniform Resource Name, URN)*. URN - это стандартизированный URI, используемый для указания ресурса независимо от его расположения в сети.

Проанализируем части URI, ссылающегося на страницу Web-сайта компании Global Knowledge:

*http://www.globalknowledge.net:80/training/generic.asp?pageid=1078&country=DACH*

* Первая часть URI называется **схемой (scheme)**. Схема определяет пространство имен URI и может сузить синтаксис следующего за схемой выражения. Многие схемы названы по соответствующим протоколам (как http, ftp), которые они используют, но это не является обязательным. В нашем примере идентификатором схемы является http. *Ограничитель схемы* (// в этом примере) отделяет схему от остальной части URL.
* После ограничителя схемы следует имя сервера или IP-адрес в десятичной записи с точками, например www.globalknowledge.net.
* За именем сервера или IP-адресом находится номер порта, определяющий соединение с конкретным приложением на сервере. Если номер порта не задан, используется номер порта, устанавливаемый для этого протокола по умолчанию (например, порт 80 для HTTP).
* *Путь* определяет страницу (и каталог) запрошенного ресурса. Он необязательно представляет физический файл на сервере, а может создаваться динамически. В данном случае путь имеет вид /training/generic.asp.
* От пути символом ? отделена последняя часть этого URI, называемая **запросом (query)**. В нашем примере запрос определен строкой pageid=1078&country=DACH. Строка запроса может состоять из нескольких компонентов, каждый из которых задает переменную и значение, объединенные символом &. Несколько компонентов запроса могут комбинироваться символом &. Так, в нашем примере первый компонент — pageid=1078 с переменной pageid и значением 1078, а второй компонент - country=DACH.
* Разделы внутри ресурса можно отождествить с фрагментами. **Фрагменты** используются для ссылок на разделы внутри HTML-страницы. В разработке Web-страниц фрагменты также называются закладками (bookmarks). Символ # отделяет идентификатор фрагмента от пути. В URL http;//www.microsoft.com/net/basics/glossary.asp#NETFramework фрагментом является строка #NETFramework.

Если символ # добавлен в строку запроса, то это уже не фрагмент. В URL может присутствовать строка запроса или фрагмент, но не то и другое одновременно.

В URI зарезервировано использование нескольких символов — они не могут входить в имена хостов или путь, поскольку представляют собой специальные символы-разделители. В URI зарезервированы следующие символы:

*; / ? : @ & = + $ ,*

**Класс Uri** из пространства имен System инкапсулирует универсальный идентификатор ресурсов. Он содержит свойства и методы для анализа, сравнения и комбинирования URI.

Можно создать объект Uri, передав конструктору строку URI:

Uri baseURI = new Uri("http://professorweb.ru");

Если уже есть базовый объект Uri, можно создать новый URI, комбинируя базовый URI с относительным URI:

Uri baseURI = new Uri("http://professorweb.ru");

Uri newURI = new Uri(baseURI, "my/csharp/web/level2/2\_2.php");

Если базовый URI уже содержит путь, он игнорируется. В качестве базы для нового URI берутся лишь схема, порт и имя сервера.

В классе Uri есть несколько статических полей только для чтения, позволяющих получить некоторые широко распространенные схемы:

**Uri.UriSchemeFile**

Файловая схема используется для доступа к файлам локально или на совместно используемых сетевых ресурсах, для которых могут применяться имена, соответствующие соглашению об универсальном назначении имен (*Universal Naming Convention, UNC*).

**Uri.UriSChemeFtp**

Протокол FTP со схемой ftp используется для получения файлов с ftp-сервера и, напротив, помещения файлов на ftp-сервер.

**Uri.UriSchemeGopher**

Протокол gopher был предшественником HTTP. Он предоставлял возможности иерархического просмотра текстовой информации о содержании, в чем превосходил FTP. Но скоро был заменен протоколом HTTP.

**Uri.UriSchemeHttp, Uri.UriSchemeHttps**

Эти две схемы хорошо известны: http и https. Схема https используется для защищенного обмена.

**Uri.UriSchemeMailto**

Схема mailto используется для отправки почтовых сообщений.

**Uri.UriSchemeNews, Uri.UriSchemeNntp**

Схемы news и nntp применяются в группах новостей, использующих протокол NNTP.

В классе Uri есть статические методы для проверки правильности схемы и имени хоста: Uri.CheckSchemeName() возвращает true, если имя схемы задано правильно, а метод **UriCheckHostName()** не только проверяет имя хоста, но и возвращает значение перечисления UriHostNameType, указывающее тип хоста.

В классе Uri имеется масса свойств с доступом только на чтение, позволяющих обращаться ко всем частям URI. В следующей таблице используем приведенный выше URI как пример, демонстрирующий применение свойств:

|  |  |
| --- | --- |
| *AbsoluteUri* | Это свойство показывает полный URI. Если указанный номер порта для протокола равен номеру порта по умолчанию, конструктор Uri автоматически его удаляет. Для нашего примера значение свойства AbsoluteUri выглядит так: *http://www.globalknowledge.net/t raining/generic.asp?pageid=1078&country=DACH*. Если конструктору класса Uri передается имя файла, свойство AbsoluteUri автоматически помещает перед именем файла схему file://. |
| *Scheme* | Схема — первая часть URI, и в данном случае это свойство возвращает значение http. |
| *Host* | Свойство Host показывает имя хоста из URI: www.globalknowledge.net |
| *Authority* | Если номер порта равен номеру, используемому протоколом по умолчанию, свойство Authority показывает ту же строку, что и свойство Host. Если используется другой номер порта, то свойство Authority также показывает номер порта. |
| *HostNameType* | Тип имени хоста зависит от используемого имени. В данном случае получается то же самое значение перечисления UriHostNameType, которое было рассмотрено выше. |
| *Port* | С помощью свойства Port получается номер порта — 80. |
| *AbsolutePath* | Абсолютный путь начинается после номера порта в URI и заканчивается перед строкой запроса. В этом случае он имеет значение /training/generic.asp. |
| *LocalPath* | Локальный путь дает значение /training/generic.asp. Как можно видеть, для запроса HTTP между AbsolutePath и LocalPath нет никакого различия. Различие появляется, если URI ссылается на совместно используемый сетевой ресурс. Для URI в виде file:\\server\share\directory\file.txt свойство LocalPath возвращает только имена directory и file, а свойство AbsolutePath включает имена server и share. |
| *Query* | Свойство Query показывает строку, следующую после пути: ?pageid=1078&country=DACH. |
| *PathAndQuery* | Свойство PathAndQuery дает комбинацию пути и строки запроса: /training/generic.asp?pageid=1078&country=DACH. |
| *Fragment* | Если после пути следует фрагмент, он возвращается в свойстве Fragment. За путем могут следовать только строка запроса или фрагмент. Фрагмент идентифицируется символом # |
| *Segments* | Свойство Segments возвращает массив строк, сформированный из пути. В данном случае у нас есть три сегмента: /, training/ и generic.asp. |
| *UserInfo* | Установленное в URI имя пользователя можно прочитать из свойства UserInfo. Передача имен пользователей распространена в протоколе FTP, и если указан не анонимный пользователь, например ftp://myuser@ftp.myserver.com, то свойство UserInfo вернет myuser. |

Кроме перечисленных, существует еще несколько свойств, возвращающих булевы значения, если URI представляет файл, путь UNC, адрес обратной связи или если для данного протокола используется номер порта по умолчанию. Это свойства IsFile, IsUnc, IsLoopback и IsDefaultPort соответственно.

После создания конструктором экземпляр класса Uri не может больше изменяться. Свойства класса Uri доступны только на чтение. Для динамического изменения URI можно использовать **класс UriBuilder**. Его свойства аналогичны свойствам класса Uri, но их значения можно как считывать, так и записывать. Благодаря доступу только на чтение класс Uri гораздо быстрее класса UriBuilder. С такой идеей вы, возможно, знакомы, если работали с классами String и [StringBuilder](https://professorweb.ru/my/csharp/charp_theory/level4/4_8.php).

В следующем примере передачей конструктору схемы, имени хоста, имени порта и пути создается экземпляр класса UriBuilder. Затем изменим путь, устанавливая свойство Path. Свойство Uri класса UriBuilder возвращает доступный только на чтение экземпляр класса Uri:

UriBuilder myURI = new UriBuilder("http", "professorweb.ru", 80,

"my/csharp/web/level2/2\_2.php", "#titlecode");

Uri uri1 = myURI.Uri;

**Класс WebClient**

[C# и .NET](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php) --- [Сетевое программирование](https://professorweb.ru/my/csharp/web/level1/web_index.php) --- Класс WebClient

Если необходимо только запросить файл с определенного URI (Uniform Resource Identifier — унифицированный идентификатор ресурса), то простейшим в использовании классом .NET, который подходит для этого, будет **System.Net.WebClient**. Этот исключительно высокоуровневый класс предназначен для выполнения базовых операций с помощью всего одной или двух команд. В настоящее время в .NET Framework поддерживаются URI, начинающиеся с идентификаторов *http:*, *https:* и *file:*.

Важно отметить, что термин URL (Uniform Resource Locator — универсальный локатор ресурсов) больше не используется в новых технических спецификациях, а вместо него отдается предпочтение URI. URI имеет приблизительно тот же смысл, что и URL, но немного более общий, потому что в URL не подразумевается обязательное применение одного из знакомых протоколов, таких как HTTP или FTP.

**Загрузка файлов**

Для загрузки файлов с использованием WebClient доступны два метода. Выбор метода зависит от того, как должно обрабатываться содержимое файла. Если необходимо просто сохранить файл на диске, следует применять **метод DownloadFile()**. Этот метод принимает два параметра: URI файла и местоположение (путь и имя файла) для сохранения запрошенных данных:

WebClient Client = new WebClient();

Client.DownloadFile("http://www.professorweb.ru/", "index.php");

Часто приложение должно обрабатывать данные, извлеченные с веб-сайта. Это обеспечивает **метод OpenRead()**, возвращающий ссылку на Stream, которую можно использовать для извлечения данных в память:

WebClient Client = new WebClient();

Stream str = Client.OpenRead("http://www.professorweb.ru/");

В следующем примере демонстрируется применение метода WebClient.OpenRead(). Содержимое загруженной страницы будет отображено в элементе управления TextBox. Для начала создайте новый проект как стандартное приложение WPF и добавьте элемент управления TextBox по имени txb. В начало файла к списку директив using потребуется добавить ссылки на пространства имен System.Net и System.IO. Затем добавьте обработчик клика по кнопке:

private void Button\_Click(object sender, RoutedEventArgs e)

{

WebClient client = new WebClient();

Stream stream = client.OpenRead("http://www.professorweb.ru");

StreamReader sr = new StreamReader(stream);

string newLine;

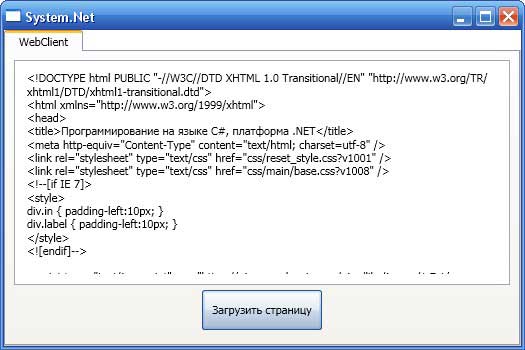
while ((newLine = sr.ReadLine()) != null)

txb.Text += newLine;

stream.Close();

}

В этом примере класс StreamReader из пространства имен System.IO подключается к сетевому потоку. Это позволяет получить данные из потока в виде текста, используя высокоуровневые методы вроде ReadLine(). На рис. показаны результаты запуска этого кода:



Класс WebClient также включает в себя **метод OpenWrite()**. Этот метод возвращает записываемый поток для отправки данных по определенному URI. При этом можно указать метод, который должен использоваться для отправки данных на хост, по умолчанию это POST.

**Выгрузка файлов**

В классе WebClient также имеются методы **UploadFile()** и **UploadData()**. Они используются, когда нужно отправить HTML-форму или выгрузить на сервер целый файл. UploadFile() выгружает файл в указанное местоположение по указанному имени локального файла, в то время как UploadData() выгружает двоичные данные, представленные в виде массива байт, по указанному URI (есть также *метод DownloadData()*, предназначенный для извлечения массива байтов из URI).

WebClient client = new WebClient();

client.UploadFile("http://professorweb.ru/NewFile.html", "samplefile.txt");

byte[] image;

client.UploadData("http://professorweb.ru/NewFile.jpg", image);

Хотя класс WebClient очень прост в использовании, он обладает весьма ограниченными возможностями. В частности, невозможно передавать регистрационные данные для аутентификации — одна из проблем, связанных с загрузкой данных, состоит в том, что лишь относительно немного сайтов принимают файлы без аутентификации.

В запросы можно добавлять информацию заголовка и проверять заголовки в ответе, но лишь в очень общем смысле — специфическая поддержка какого-то одного протокола отсутствует. Причина в том, что WebClient — класс общего назначения, предназначенный для работы с любым протоколом, позволяющем отправлять запросы и получать ответы (вроде HTTP и FTP). Он не может обработать никаких средств, специфичных для какого-то одного протокола, например, сокеты, которые специфичны для HTTP. Чтобы воспользоваться преимуществами этих средств, нужно работать с семейством классов, основанным на двух других классах из пространства имен System.Net, а именно — WebRequest и WebResponse.

# Классы WebRequest и WebResponse

[C# и .NET](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php) --- [Сетевое программирование](https://professorweb.ru/my/csharp/web/level1/web_index.php) --- Классы WebRequest и WebResponse

Класс WebRequest представляет запрос информации для отправки по определенному URI. Идентификатор URI передается в качестве параметра методу Create(). Класс WebResponse представляет данные, извлекаемые из сервера. Вызов метода WebRequest.GetResponse() на самом деле приводит к отправке запроса веб-серверу и к созданию объекта WebResponse для просмотра возвращенных данных. Как и в случае объекта WebClient, можно получить поток для представления данных, но для этого должен использоваться метод WebResponse.GetResponseStream().

Сначала рассмотрим класс WebRequest:

|  |  |
| --- | --- |
| Методы и свойства класса WebRequest | |
| **Методы и свойства** | **Описание** |
| Create() и CreateDefault() | В классе WebRequest нет открытого конструктора. Вместо конструктора для создания экземпляров класса могут использоваться статические методы Create() и CreateDefault(). Эти методы в действительности создают не объект типа WebRequest, а новый объект класса, производного от WebRequest, такого как HttpWebRequest или FileWebRequest. |
| RegisterPrefix() | Используя метод RegisterPrefix(), можно зарегистрировать класс для обработки специфического протокола. Объекты этого класса будут создаваться методом WebRequest.Create(). Этот механизм называется "подключаемыми протоколами" (pluggable protocols). |
| GetRequestStream() | Метод GetRequestStream() возвращает объект потока, который может использоваться для отправки некоторых данных на сервер. |
| BeginGetRequestStream() и EndGetRequestStream() | Асинхронный доступ к потоку запроса выполняется методами BeginGetRequestStream() и EndGetRequestStream(). |
| GetResponse() | Метод GetResponse() возвращает объект WebResponse, который может использоваться для чтения данных, полученных от сервера. |
| BeginGetResponse() и EndGetResponse() | Как и для потока запроса, имеются асинхронные методы дпя получения потока ответа. |
| Abort() | Если метод BeginXX() начал асинхронную обработку, ее можно остановить методом Abort(). |
| RequestUri | RequestUri - свойство только для чтения, возвращающее URI, связанный с WebRequest. Этот URI может быть установлен в статическом методе Create() данного класса. |
| Method | Свойство Method используется, чтобы получить или установить метод для конкретного запроса. Объект HttpWebRequest поддерживает HTTP-методы GET, POST, HEAD и т. д. |
| Headers | В зависимости от используемого протокола серверу может передаваться и от сервера может получаться различная информация в заголовках. Информация заголовка протокола содержится в коллекции WebHeaderCollection, к которой можно обращаться через свойство Headers. |
| ContentType и ContentLength | Тип данных, отправленных серверу, определяется в свойстве ContentType. Могут быть разные типы данных такой длины, чтобы данные могли разместиться в массиве байтов. Тип содержания обычно определяет МIМЕ-тип данных: image/jpeg, image/gif, text/html или text/xml. |
| Credentials | Если серверу требуется аутентификация пользователя, удостоверения личности пользователя можно установить через свойство Credentials. |
| PreAuthenticate | Для протоколов, поддерживающих предварительную аутентификацию, в свойстве PreAuthenticate можно установить значение true. По умолчанию Web-браузер сначала пытается обратиться к странице Web-сайта без аутентификации. Если Web-сайту требуется аутентификация, сервер отвечает, что для неидентифицированных пользователей доступ отклонен. Следующий запрос, выполняемый клиентом, содержит информацию аутентификации. Этого дополнительного цикла обмена можно избежать, если установить в свойстве PreAuthenticate значение true. |
| Proxy | В свойстве Proxy можно установить Web-прокси, который используется для этого запроса. |
| ConnectionGroupName | В свойстве ConnectionGroupName можно определить пул соединений, который должен использоваться с этим объектом WebRequest. |
| Timeout | Свойство Timeout определяет время в миллисекундах, которое необходимо для ответа от сервера. По умолчанию установлено значение 100 000 млс. Если в течение этого времени сервер не отвечает, порождается исключение WebException. |

Класс WebResponse используется для чтения данных от сервера. Объект этого класса возвращается методом GetResponse(), как видно при рассмотрении класса WebRequest.

|  |  |
| --- | --- |
| Методы и свойства класса WebResponse | |
| **Методы и свойства** | **Описание** |
| GetResponseStream() | Метод GetResponseStream() возвращает объект потока, который используется для чтения ответа от сервера. |
| Close() | Если объект ответа больше не нужен, его следует закрыть методом Close(). |
| ResponseUri | С помощью свойства ResponseUri мы можем считать URI, связанный с объектом ответа. Он может совпадать с URI объекта WebRequest, но может и отличаться, если сервер переадресовал запрос к другому ресурсу. |
| Headers | Свойство Headers возвращает коллекцию WebHeaderCollection, которая включает специфичную для протокола информацию о заголовках, возвращаемых от сервера. |

Давайте рассмотрим небольшой пример использования этих классов в WPF-приложении:

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="auto"/>

<RowDefinition Height="auto"/>

<RowDefinition/>

<RowDefinition Height="auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<StackPanel Orientation="Horizontal" Margin="8">

<TextBlock VerticalAlignment="Center">URL-адрес</TextBlock>

<TextBox x:Name="txb\_url" Margin="14,0" Width="250" VerticalContentAlignment="Center" Text="http://professorweb.ru/"/>

<Button Click="request\_Click" Padding="5" Content="Получить информацию"/>

</StackPanel>

<TextBlock Margin="8" Text="Исходный код страницы: " Grid.Row="1"/>

<TextBox x:Name="txb\_sourceCode" Grid.Row="2" Padding="5" Margin="8" TextWrapping="Wrap" VerticalScrollBarVisibility="Visible"/>

<GridSplitter Grid.Row="3" HorizontalAlignment="Stretch" Height="3" Margin="8,0" Background="#aaa"/>

<TextBox x:Name="txb\_serverInfo" Grid.Row="4" Padding="5" Margin="8" TextWrapping="Wrap" VerticalScrollBarVisibility="Visible"/>

</Grid>

private void request\_Click(object sender, RoutedEventArgs e)

{

// Создать объект запроса

WebRequest request = WebRequest.Create(txb\_url.Text);

// Получить ответ с сервера

WebResponse response = request.GetResponse();

// Получаем поток данных из ответа

using (StreamReader stream = new StreamReader(response.GetResponseStream()))

{

// Выводим исходный код страницы

string line;

while ((line = stream.ReadLine()) != null)

txb\_sourceCode.Text += line + "\n";

}

// Получаем некоторые данные о сервере

string messageServer = "Целевой URL: \t" + request.RequestUri + "\nМетод запроса: \t" + request.Method +

"\nТип полученных данных: \t" + response.ContentType + "\nДлина ответа: \t" + response.ContentLength + "\nЗаголовки";

// Получаем заголовки, используем LINQ

WebHeaderCollection whc = response.Headers;

var headers = Enumerable.Range(0, whc.Count)

.Select(p =>

{

return new

{

Key = whc.GetKey(p),

Names = whc.GetValues(p)

};

});

foreach (var item in headers)

{

messageServer += "\n " + item.Key + ":";

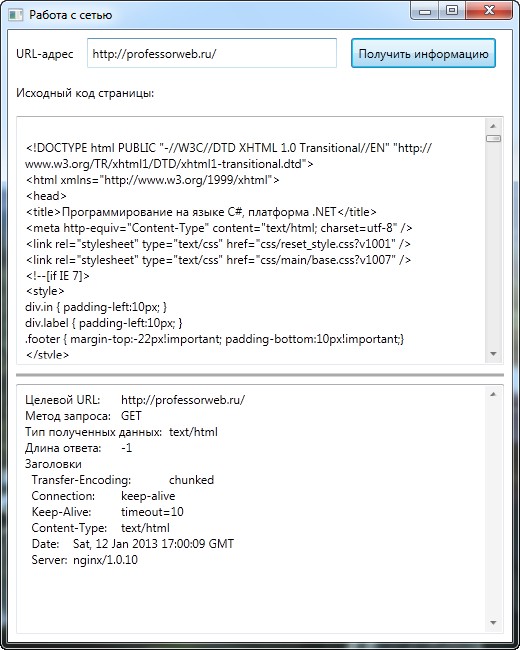
foreach (var n in item.Names)

messageServer += "\t" + n;

}

txb\_serverInfo.Text = messageServer;

}



## Подключаемые протоколы

WebRequest — это абстрактный класс, поэтому метод WebRequest.Create() не может создать объект типа WebRequest — вместо этого создается объект класса, производного от WebRequest. При передаче HTTP-запроса методу WebRequest.Create() создается объект **HttpWebRequest**. При передаче запроса со схемой файла создается объект **FileWebRequest**.

Как показано далее, схемы http, https и file предопределены в конфигурационном файле .NET, файле machine.config. Конфигурационный файл можно найти в каталоге <windows>\Microsoft.NET\Framework\<version>\CONFIG.

Набор протоколов, используемых классом WebRequest, можно расширить программно или добавив элемент в конфигурационный файл. Для поддержки нового протокола, отличного от схем http, https и file, нужно создать новый класс, производный от WebRequest, например FtpWebRequest для протокола FTP. Этот класс должен переопределить методы и свойства базового класса и в них реализовать специфичное для протокола поведение. Кроме того, требуется определить класс-инициатор (factory class), создающий объекты класса FtpWebRequest. Такой класс-инициатор, используемый классом WebRequest, должен реализовать интерфейс IWebRequestCreate. Назовем этот класс FtpWebRequestCreator. Экземпляр этого класса должен быть зарегистрирован для схемы ftp с помощью класса WebRequest:

WebRequest.RegisterPrefix("ftp", new FtpWebRequestCreator);

Если теперь схема ftp используется с методом WebRequest.Create(), создается и возвращается в программу новый экземпляр класса FtpWebRequest. Теперь объект request можно использовать для копирования файлов с FTP-cepвера и на FTP-сервер. Здесь мы не собираемся заниматься реализацией класса FtpWebRequestCreator, но вы можете это сделать самостоятельно. Для программирования FTP-клиента требуется использовать классы сокетов с соединением TCP.

## FileWebRequest и FileWebResponse

Чтение и запись локальных файлов или файлов, находящихся на совместно используемых устройствах, не очень отличаются от чтения и записи файлов, расположенных на Web-серверах. Чтобы считывать и записывать файлы, используем классы FileWebRequest и FileWebResponse. Однако многие методы и свойства, определенные в базовых классах WebRequest и WebResponse, не используются в производных классах, и в документации MSDN они лишь перечисляются, как "зарезервированные для использования в будущем".

Для демонстрации возможного использования классов FileWebRequest и FileWebResponse создается простое приложение WPF, в котором имя открываемого файла можно ввести в текстовом поле, после чего файл открывается и отображается в многострочном текстовом поле. В открытый файл можно будет записать новый текст:

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="auto"/>

<RowDefinition Height="auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="auto"/>

<ColumnDefinition/>

<ColumnDefinition Width="auto"/>

</Grid.ColumnDefinitions>

<TextBlock Text="Открыть файл" VerticalAlignment="Center" Margin="8"/>

<TextBox x:Name="txb\_fileuri" Margin="14,8" VerticalContentAlignment="Center" Grid.Column="1"/>

<Button Content="Открыть" Click="openFile\_Click" Padding="5" MinWidth="120" Grid.Column="2" Margin="0,8,8,8"/>

<TextBlock Text="Записать в файл" VerticalAlignment="Center" Grid.Row="1" Margin="8"/>

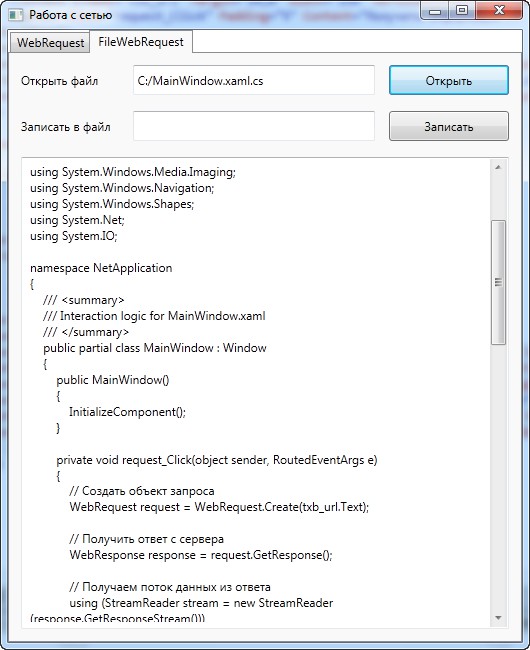
<TextBox x:Name="txb\_writefile" Margin="14,8" VerticalContentAlignment="Center" Grid.Row="1" Grid.Column="1"/>

<Button Content="Записать" Click="writeFile\_Click" Padding="5" MinWidth="120" Grid.Row="1" Grid.Column="2" Margin="0,8,8,8"/>

<TextBox x:Name="txb\_fileContent" TextWrapping="Wrap" Margin="8" Padding="5" Grid.ColumnSpan="3" Grid.Row="2"

VerticalScrollBarVisibility="Visible"/>

</Grid>



### Чтение из файлов

Обработчик щелчка по кнопке "Открыть" открывает файл и записывает содержание файла в многострочное текстовое поле. Передадим имя файла методу WebRequest.Create(). Ставить схему file:// перед именем файла необязательно. Класс WebRequest создает объект Uri и использует его свойство AbsolutePath. Как указывалось ранее, класс Uri автоматически предпосылает имени файла корректную схему. Поэтому передача имени файла классу WebRequest создаст объект FileWebRequest , и требуется лишь привести его тип. Метод GetResponse() возвращает объект FileWebResponse, который сразу же используется для создания методом GetResponseStream() объекта Stream. Обычными методами класса StreamReader считывается поток. Данные всего файла считываем в строку и передаем ее в свойство Text многострочного текстового поля txb\_fileContent:

private void openFile\_Click(object sender, RoutedEventArgs e)

{

string filename = txb\_fileuri.Text;

FileWebRequest request =

(FileWebRequest)WebRequest.Create(filename);

using (StreamReader sr = new StreamReader(request.GetResponse().GetResponseStream()))

{

txb\_fileContent.Text = sr.ReadToEnd();

}

}

### Запись в файлы

Для записи данных обратно в файл реализуем обработчик щелчка по кнопке "Записать". Как и раньше, создадим объект WebRequest, передавая имя файла. Теперь вместо StreamReader используем StreamWriter. Кроме этого есть еще одно существенное изменение в коде. Чтобы сделать поток "записываемым", следует установить в свойстве Method значение "PUT". По умолчанию это свойство имеет значение "GET", указывая, что поток можно только считывать:

private void writeFile\_Click(object sender, RoutedEventArgs e)

{

WebRequest request = WebRequest.Create(txb\_fileuri.Text);

request.Method = "PUT";

using (StreamWriter sw = new StreamWriter(request.GetRequestStream()))

{

sw.Write(txb\_writefile.Text);

}

}

## Асинхронные запросы страниц

Дополнительным средством класса WebRequest является способность запрашивать страницы асинхронно. Это средство существенно, потому что между отправкой запроса на хост и получением ответа может существовать ощутимая задержка. Такие методы, как WebClient.DownloadData() и WebRequest.GetResponse(), не вернут управления, пока не будет готов ответ сервера. Вряд ли захочется "замораживать" приложение на длительный период, и потому в таких сценариях лучше применять методы BeginGetResponse() и EndGetResponse().

Метод BeginGetResponse() работает асинхронно и возвращает управление практически мгновенно. "За кулисами" исполняющая среда асинхронно управляет фоновым потоком, чтобы получить ответ от сервера. Вместо возврата объекта WebResponse, метод BeginGetResponse() возвращает объект, реализующий интерфейс **IAsyncResult**. С этим интерфейсом можете продолжить работу или подождать, пока не станет доступным ответ, и затем вызвать EndGetResponse() для сбора результатов.

Можно также передать делегат обратного вызова в метод BeginGetResponse(). Целью делегата обратного вызова должен быть метод, возвращающий void и принимающий ссылку IAsyncResult в качестве параметра. Когда рабочий поток завершает получение результата, исполняющая среда вызывает делегат обратного вызова, чтобы проинформировать о завершении работы. Как показано в следующем коде, вызов EndGetResponse() в методе обратного вызова позволяет извлечь объект WebResponse (модифицируем первый пример):

private void request\_Click(object sender, RoutedEventArgs e)

{

// Создать объект запроса

WebRequest request = WebRequest.Create(txb\_url.Text);

request.BeginGetResponse(new AsyncCallback(OnResponse), request);

}

protected void OnResponse(IAsyncResult ar)

{

WebRequest request = (WebRequest)ar.AsyncState;

WebResponse response = request.EndGetResponse(ar);

// Читаем ответ

...

}

Обратите внимание, что для извлечения исходного объекта WebRequest методу BeginGetResponse() можно передать этот объект во втором параметре. Второй параметр является ссылкой на объект, и он известен как параметр состояния. Во время выполнения метода обратного вызова тот же объект состояния можно извлечь с использованием свойства AsyncState интерфейса IAsyncResult.

**Классы .NET для IP-адресов**

[C# и .NET](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php) --- [Сетевое программирование](https://professorweb.ru/my/csharp/web/level1/web_index.php) --- Классы .NET для IP-адресов

В Интернете серверы и клиенты идентифицируются по IP-адресу либо имени хоста (также известному, как DNS-имя). В общем случае, имя хоста — это дружественное для человека имя, которое вводится в поле адреса веб-браузера, такое как www.professorweb.ru или www.microsoft.com. IP-адрес — это идентификатор, используемый компьютерами для опознания друг друга. IP-адреса представляют собой идентификаторы, применяемые для обеспечения достижения запросами и ответами соответствующих машин. Один компьютер может даже иметь несколько IP-адресов.

В наше время IP-адреса обычно представлены 32-битным значением. Вот пример 32-битного IP-адреса: 192.168.1.100. Этот формат IP-адреса определен в протоколе *Internet Protocol version 4*.

Поскольку количество компьютеров и прочих устройств, претендующих на место в Интернете, неуклонно растет, была разработана более новая система адресации — *Internet Protocol version 6*. Протокол IPv6 предоставляет 64-битные IP-адреса. IPv6 потенциально может представить до 3e+28 уникальных адресов. Среда .NET Framework позволяет приложениям работать как с IPv4, так и с IPv6.

Чтобы имена хостов работали, сначала должен быть отправлен сетевой запрос на трансляцию имени хоста в IP-адрес; эту задачу решает один или более DNS-серверов.

DNS-сервер хранит таблицу, отображающую имена хостов на IP-адреса для всех известных ему компьютеров, а также IP-адреса других DNS-серверов, где можно искать имена хостов, которые ему неизвестны. Локальный компьютер должен всегда знать, по крайней мере, один DNS-сервер. Сетевые администраторы конфигурируют эту информацию при настройке компьютера.

Прежде чем отправлять запрос, компьютер сначала спрашивает у DNS-сервера IP-адрес, соответствующий введенному имени хоста. Получив корректный IP-адрес, компьютер может отправить ему запрос через сеть. Все это нормально работает "за кулисами", пока пользователь путешествует по Интернету.

В .NET Framework предлагается множество классов, которые помогают в процессе поиска IP-адресов и нахождении информации о компьютерах-хостах.

**Класс IPAddress**

Класс IPAddress представляет IP-адрес. Сам адрес доступен в виде свойства **GetAddressBytes** и может быть преобразован в десятичный формат с разделителями-точками с помощью метода ToString(). В этом классе также реализован статический метод **Parse()**, который эффективно выполняет преобразование, обратное ToString() — из десятичного формата с разделителями-точками в IPAddress:

IPAddress ip = IPAddress.Parse("234.58.78.9");

byte[] adress = ip.GetAddressBytes();

string ipString = ip.ToString();

В классе IPAddress есть несколько открытых, доступных только для чтения полей, которые возвращают предопределенные IP-адреса:

**IPAddress.None**

Возвращает адрес, который означает, что ни один сетевой интерфейс не должен использоваться. Это поле используется классом Socket, чтобы указать серверу не ожидать активности клиента.

**IPAddress.Loopback**

Возвращает предопределенный адрес обратной связи 127.0.0.1. Этот адрес используется не для соединения с сетью, а для локальных операций на одной машине.

**IPAddress.Broadcast**

Возвращает широковещательный IP-адрес. В широковещательных сообщениях можно посылать данные всем компьютерам в локальной сети.

**IPAddress.Any**

У компьютера может быть несколько сетевых плат с несколькими IP-адресами. Сокет использует IPAddress.Any, чтобы ожидать действия на любом из этих сетевых интерфейсов.

**Класс IPHostEntry**

Класс IPHostEntry инкапсулирует информацию об определенном компьютере-хосте. Этот класс делает имя хоста доступным через свойство HostName (которое возвращает строку), а свойство AddressList возвращает массив объектов IPAddress.

**Класс Dns**

Класс Dns способен взаимодействовать с DNS-сервером по умолчанию для извлечения IP-адреса. Он имеет два важных статических метода — **GetHostEntry()**, который использует DNS-сервер для получения деталей хоста по заданному его имени, и **GetHostByAddress()**, который также возвращает детали хоста, но на этот раз используя IP-адрес. Оба метода возвращают объект IPHostEntry.

Класс Dns отличается от классов IPAddress и IPHostEntry тем, что обладает способностью действительно взаимодействовать с серверами для получения информации. В отличие от него IPAddress и IPHostEntry — это скорее просто структуры данных с удобными свойствами, обеспечивающими доступ к лежащим в основе данным.

Давайте рассмотрим пример WPF-приложения, использующего возможности класса Dns:

public MainWindow()

{

InitializeComponent();

string hostname = "www.google.com", message = "IP адреса для домена " + hostname + "\n";

IPHostEntry entry = Dns.GetHostEntry(hostname);

foreach (IPAddress a in entry.AddressList)

message += " --> "+ a.ToString() + "\n";

message += "\nАльтернативное имя домена: ";

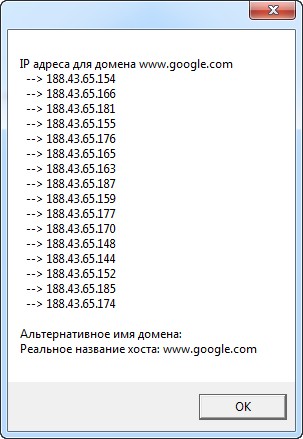
foreach (string aliasName in entry.Aliases)

message += aliasName + "\n";

message += "\nРеальное название хоста: " + entry.HostName;

MessageBox.Show(message);

}



**Web-прокси**

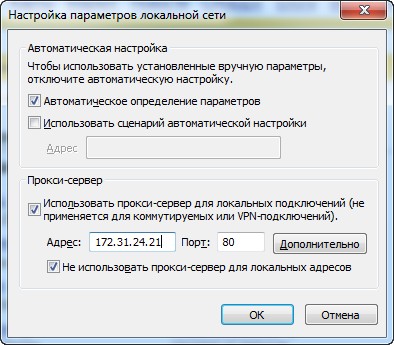
[C# и .NET](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php) --- [Сетевое программирование](https://professorweb.ru/my/csharp/web/level1/web_index.php) --- Web-прокси

В локальной сети можно использовать прокси-сервер, чтобы направить интернет-доступ к конкретным серверам. Прокси-сервер может сократить число передач и сетевых соединений из Интернета и повысить благодаря кэшированию ресурсов производительность локальных клиентов.

Прокси-сервер выполняет активное и пассивное кэширование:

* При пассивном кэшировании Web-pecypcы сохраняются в кэше прокси-сервера, как только клиент запрашивает ресурс. Если второй клиент запрашивает тот же самый ресурс, получать его снова от Web-сервера в Интернете не нужно, поскольку Web-прокси может ответить непосредственно из кэша, созданного при первом запросе.
* С помощью активного кэширования системный администратор может сконфигурировать конкретные Web-серверы и каталоги, которые должны кэшироваться автоматически в соответствии со специальным расписанием, например в ночные часы. Этим способом пропускная способность, необходимая для Интернета, днем может быть сокращена, чтобы увеличить производительность для часто используемых страниц.

Прокси-сервер, настроенный по умолчанию, устанавливается из Internet Options в Control Panel. Кроме того, к средствам конфигурирования можно также получить доступ из Internet Explorer (Tools --> Internet Options --> Connections --> LAN Settings):



В данном случае Web-прокси-сервер имеет IP-адрес 172.31.24.21 и слушает порт 80. Этот прокси-сервер не должен использоваться для Web-серверов в интрасети. Через кнопку "Дополнительно" можно сконфигурировать разные прокси-серверы для разных протоколов (HTTP, HTTPS или FTP) и выбрать конкретные Web-сайты, к которым прокси-сервер не должен обращаться.

**Класс WebProxy**

Класс WebProxy используется для определения прокси-сервера. Свойства этого класса аналогичны настройкам, которые были рассмотрены вместе с конфигурированием прокси-сервера:

|  |  |
| --- | --- |
| Класс WebProxy | |
| **Свойства WebProxy** | **Описание** |
| Address | Свойство Address имеет тип Uri и определяет URI прокси-сервера, IP-адрес или имя и номер порта. |
| BypassList | В свойстве BypassList можно получать и устанавливать в массиве строк URI, которые не должны использовать прокси-сервер. |
| BypassArrayList | BypassArrayList — это свойство только для чтения, возвращающее объект типа ArrayList, представляющий URI, которые устанавливаются в свойстве BypassList. |
| BypassProxyOnLocal | BypassProxyOnLocal — это логическое свойство, указывающее, должны ли с прокси-сервером использоваться локальные адреса. |
| Credentials | Если прокси-сервер требует аутентификации пользователя, в свойстве Credentials можно передать удостоверение личности пользователя. |

Вместо того, чтобы использовать установленный по умолчанию Web-прокси для всех запросов, можно выделить другой прокси для конкретных запросов. Для выбора другого прокси нужно лишь установить свойство Proxy класса WebRequest:

WebProxy wp = new WebProxy("92.168.1.100", true);

wp.Credentials = new NetworkCredential("user1", "user1Password");

WebRequest wrq = WebRequest.Create("http://www.example.com");

wrq.Proxy = wp;

WebResponse wrs = wrq.GetResponse () ;

# Аутентификация и разрешения

[C# и .NET](https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php) --- [Сетевое программирование](https://professorweb.ru/my/csharp/web/level1/web_index.php) --- Аутентификация и разрешения

Если Web-cepвep требует аутентификации пользователя, можно создать удостоверение личности пользователя и передать его Web-запросу. При этом полезны следующие интерфейсы и классы: ICredentials, NetworkCredential и CredentialCache.

Для аутентификации пользователя создадим объект типа NetworkCredential. Этот класс обеспечивает информацию с целью удостоверения личности пользователя для базовой аутентификации, аутентификации на основе дайджестов, NTLM и Kerberos.

Конструктору класса **NetworkCredential** можно передать имя пользователя, пароль и дополнительно домен, разрешающий доступ пользователя:

NetworkCredential credential =

new NetworkCredential("Alex", "12345", "alex.professorweb.ru");

Для авторизации пользователя эту информацию удостоверения личности можно установить в свойстве Credentials класса WebRequest:

WebRequest request = WebRequest.Create("http://professorweb.ru/");

request.Credentials = credential;

Если нужна разная информация удостоверений личности для разных URI, можно использовать класс **CredentialCache**, как показано в следующем коде. С таким кэшем также определяется тип аутентификации для конкретного соединения. Здесь используется базовая аутентификация для Web-сайта www.unsecure.com и аутентификация на основе дайджестов для Web-сайта www.moresecure.com, для которого через сеть посылается не пароль, а хеш-код:

CredentialCache ccache = new CredentialCache();

ccache.Add(new Uri("http://www.unsecure.com"), "Basic",

new NetworkCredential("user", "password"));

ccache.Add(new Uri("http://www.moresecure.com"), "Digest",

new NetworkCredential("user", "password", "domain"));

## Разрешения

Всякий раз когда используются сетевые классы, требуются разрешения. Для вопросов сетевого обмена рассмотрим три типа разрешения.

Разрешение DnsPermission требуется для поиска имени DNS с помощью класса Dns. WebPermission используется классами из пространства имен System.Net, которые отправляют данные в Интернет и получают данные с помощью URI. SocketРеrmission используется для приема данных на локальном сокете или соединения с хостом через транспортный протокол.

Приложения, установленные локально в системе, пользуются полным доверием, поэтому все разрешения доступны по умолчанию. Приложения платформы. .NET также могут запускаться с общего сетевого ресурса, а сборки могут загружаться из Интернета — в этих ситуациях многие разрешения недоступны по умолчанию. Следовательно, для этих приложений надо конфигурировать параметры настройки безопасности. Сначала обсудим программные аспекты безопасности, после чего рассмотрим, как можно конфигурировать разрешения.

**DnsPermission**

Когда используется класс Dns для поиска IP-адреса, требуется разрешение DnsPermission. Для него различаем только значения "признать" и "отвергнуть". Запросы DNS или абсолютно не ограничиваются, или не разрешаются совсем.

**WebPermission**

WebPermission требуется для таких классов, как WebRequest и WebResponse, чтобы отправлять данные в Интернет и получать их из Интернета.

В этом случае различаются разрешения "согласиться" (Accept) и "соединиться" (Connect). Разрешение Accept нужно для URI, используемых внутри классов и методов. Клиентским приложениям, использующим URI для соединения с сервером, требуется полномочие Connect. У класса WebPermission также есть список URI, с которыми можно соединиться, и список URI, с которыми можно согласиться.

**SocketPermission**

Разрешения SocketРеrmission нужны для классов сокетов из пространства имен System.Net.Sockets. Это разрешение — самое гибкое из трех классов сетевых разрешений.

Для серверных приложений, ожидающих запросы на соединения от клиентов, в конструктор передается значение перечисления NetworkAccess.Accept; клиентские приложения, соединяющиеся с серверами, используют значение NetworkAccess.Connect. Можем ограничить соединение конкретными хостами и номерами портов и определить используемый транспортный протокол.

### Использование атрибутов разрешения

Если требуемое разрешение недоступно, программа завершается с исключением SecurityException, как только вызван привилегированный метод. До порождения исключения пользователь мог какое-то время работать с приложением, и может потерять данные, если не обработать исключение аккуратно. Хороший способ, позволяющий этого избежать, состоит в том, чтобы отметить сборку необходимыми разрешениями.

Если для получения данных из Интернета используется класс WebRequest, требуется разрешение WebPermission. Можно отметить классы и методы, требующие разрешения, атрибутом WebPermission (реализованным в классе WebPermissionAttribute) следующим образом:

[WebPermission(System.Security.Permissions.SecurityAction.Demand,

ConnectPattern="www.professorweb.ru")]

class PermissionDemo

{

...

В данном случае исключение SecurityException возникает, как только создается экземпляр класса PermissionDemo. Если хотим, чтобы эта проверка выполнялась при старте программы, можно применить атрибут WebPermission ко всей сборке:

[assembly: WebPermission(System.Security.Permissions.SecurityAction.Demand,

ConnectPattern = "www.professorweb.ru")]

Если этот атрибут применен к сборке, при старте программы среда выполнения проверяет, имеет ли программа необходимое разрешение. Если она не обладает требуемым разрешением, выполнение прекращается немедленно, прежде чем пользователь ввел (или потерял) информацию.

Со всеми атрибутами разрешения конструктору передается значение перечисления **SecurityAction**. Здесь рассмотрим только самые важные значения перечисления:

|  |  |
| --- | --- |
| Перечисление SecurityAction | |
| **Значения перечисления** | **Описание** |
| Demand и Deny | Значения перечисления SecurityAction.Demand и SecurityAction.Deny можно использовать с классами и методами. Значением Demand указывается, что классу или методу требуется разрешение, значение Deny определяет, что это разрешение не нужно. |
| RequestMinimum, RequestOptional и RequestRefuse | Значением перечисления Requestxxx можно пользоваться только в контексте сборки, его нельзя указывать с классами и методами. RequestMinimum определяет, что это разрешение обязательно для использования программы.   Значением RequestOptional сообщается, что программа может выполнить некоторую полезную работу без этого разрешения. В этом случае надо постепенно обработать исключение SecurityException.   Значение RequestRefuse определяет, что это разрешение не нужно. Это значение используется в тех случаях, когда возможно неправильное применение разрешения, например вызов сборок, для которых у нас нет исходных текстов и которым мы не доверяем полностью. |

С атрибутом WebPermission можно в дополнение к SecurttyAction установить следующие свойства:

**Accept и AcceptPattern**

В свойстве Accept определяется URI ресурса для использования с классом, методом или сборкой, к которым применен этот атрибут. В свойстве AcceptPattern указывается регулярное выражение, чтобы разрешить или запретить доступ к URI.

**Connect и ConnectPattern**

Два свойства ConnectXX аналогичны свойствам AcceptXX, но отличаются тем, что применяются для строки соединения с URI.

В классе SocketPermissionAttribute определены следующие дополнительные свойства:

**Access**

В этом свойстве определяется допустимый метод доступа к сети. Допустимы только два: Accept и Connect. Значение Accept используется для серверного приложения, слушающего и принимающего соединения клиентов, а значение Connect предназначено для клиента, соединяющегося с сервером.

**Host**

В свойстве Host с использованием синтаксиса DNS или IP-адреса устанавливается имя хоста, к которому относится разрешение.

**Port**

Это строковое свойство, задающее номер порта, для которого требуется разрешение. Свойство может использоваться для ограничения приложений-клиентов некоторыми конкретными серверами. Свойство имеет тип string, поскольку разные протоколы необязательно определяют номер порта как целое число.

**Transport**

С помощью свойства Transport ограничиваются сетевые соединения конкретным транспортным протоколом. Возможные значения: All, ConnectionLess, CormectionOriented, Tcp и Udp. Значение ConnectionLess позволяет использовать все протоколы, не требующие устанавливать соединения, например UDP; значение ConnectionOriented позволяет использовать протоколы, ориентированные на установление соединения, например TCP.