Cursos de Programación Web .NET

Unidad 1 – Material adicional

“Redes”

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Índice

Trabajando en Red

Introducción

El Framework de Microsoft .NET nos brinda muchas facilidades en términos de trabajo en red que pueden ser integradas fácilmente en nuestras aplicaciones.
Con mínimos conocimientos de redes podemos crear un sistema de aplicaciones que se comunican en la red gracias a los componentes pre-ensamblados de .NET sin ser una tarea tan tediosa.

Conceptos generales de Redes

Introducción

Un entendimiento básico de redes o networking es importante para alguien que desarrolla software ya que casi todas las aplicaciones de hoy en día se comunican entre sí y la manera de comunicarse sigue los mismos viejos conceptos de redes de hace varias décadas. Los conceptos de **Web Services** o **Web Apis** dependen de la teoría clásica de redes.

La intención de este apartado es darte la teoría básica en términos de nomenclatura, protocolos y elementos de redes para que puedas no solo crear software en red sino también entender como funciona.

Términos básicos

Conexión: Se refiere al canal de transmisión de información entre dos puntos de la red. Una conexión se crea antes de la transferencia de datos y luego se destruye finalizando la misma.

Paquete: Un paquete es la unidad básica de transferencia en una red. Cuando nos comunicamos en la red la información se enviá por partes, cada parte se denomina paquete. Los paquetes tienen además información adicional como destino, horario y puntos por donde pasó.

Network Interface: Una interfaz de red se refiere a cualquier hardware (placa de red) o software (placa virtual) que permite conectarse, transmitir o recibir información. La misma tiene una dirección (IP) y una computadora puede tener varias interfaces, vendrían a ser como buzones. Una interfaz puede estar conectada a la LAN (Área local) o a la WAN (Internet).

Protocolos

Un protocolo es un estándar que define un lenguaje de comunicación entre dos aparatos electrónicos.

IP: La dirección IP es la dirección para llegar a nuestra computadora desde la LAN (IP Privada) o desde internet (IP Pública). Ejemplo: 192.168.1.105

Puerto: El puerto es una ubicación dentro de la máquina que se designa a una aplicación para poder enviar y recibir datos en él. Ejemplo: 8080

Firewall: El firewall es un programa que decide si el tráfico que se envía y recibe de un puerto y una IP es seguro.

Protocolo TCP

TCP significa Protocolo de Control de transmisión y es usado para establecer conexiones confiables.

El protocolo TCP encapsula los datos en paquetes que luego envía por medio del canal de red hacia el destino. Del otro lado el que recibe puede pedir reenviar paquetes si estos presentan errores.

Este protocolo es la elección principal de muchos de las tecnologías de Internet, básicamente sin este protocolo la informática como la conocemos sería muy distinta. Las páginas Web, los FTP, los e-mails todos están basados en este protocolo.

Protocolo UDP

La principal diferencia entre UDP y TDP es que el primero no verifica que los datos se hayan enviado. Al no requerir este acuse de recibo este protocolo es mucho mas rápido y puede ser muy útil para muchos usos como Voz sobre IP o video juegos en línea, que requieren que la velocidad de transmisión sea la mejor.

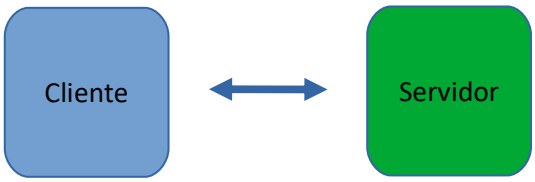
Protocolo HTTP

Este protocolo es el que que define los conceptos fundamentales para la Web y principalmente para los Web Services. HTTP define funciones para simplificar la comunicación de las páginas web como: GET, POST y DELETE.

Protocolo DNS

DNS es el protocolo para darle nombres amigables a las direcciones IP. Es el encargado de transformar www.google.com que ingresas en tu browser en la IP que va a ser utilizada en el protocolo HTTP.

Aplicaciones Cliente Servidor



Explicación

El modelo cliente servidor es una estructura distribuida en la cual las tareas se reparten entre los puntos que envían los pedidos (Clientes) y los puntos que procesan y proveen la información (Servidores)

Generalmente Clientes y Servidores están en distintas computadoras sobre la misma red, aunque se puede dar el caso de que ambos estén en la misma computadora.

Un cliente hace pedidos de procesamiento en el Servidor y este le devuelve los resultados, por lo tanto es el Cliente el que comienza la comunicación a los Servers que están (la mayor parte del tiempo) encendidos esperando a recibir pedidos.

Un gran ejemplo clásico es la **Web**, nosotros como usuarios de Internet Explorer o Chrome instalado en nuestra PC somos los clientes, del otro lado Google o Microsoft tiene un Servidor que espera que pongamos en la barra sus direcciones para empezar a enviarnos información en forma de páginas web. En el caso de la Web el browser y el server se entienden por medio del protocolo HTTP.

En el próximo apartado vamos a realizar un Cliente Servidor utilizando el Protocolo TCP

Desarrollo de un Chat usando TCP

Para crear nuestra aplicación Cliente Servidor vamos a utilizar componentes .NET que nos van a facilitar el trabajo utilizando el protocolo TCP

TCP Listener

Este componente se utiliza para escuchar conexiones TCP en nuestra aplicación .NET

TCP Client

Este componente se utiliza para conectarse a servidores TCP en nuestra aplicación .NET

Implementación

TCP Server

```
TcpListener server = null;

      Int32 port = 16000;
IPAddress localAddr = IPAddress.Parse("127.0.0.1");

server = new TcpListener(localAddr, port);


        server.Start();

Byte[] bytes = new Byte[256];
String data = null;

while (true)
        {
  Console.Write("Esperando que un cliente se conecte... ");
            TcpClient client = server.AcceptTcpClient();
  Console.WriteLine("Cliente conectado!");
  data = null;
            NetworkStream stream = client.GetStream();

  int i;
  while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
            {
                data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);
    Console.WriteLine("Recibido: {0}", data);
                data = data.ToUpper();
    byte[] msg = System.Text.Encoding.ASCII.GetBytes(data);
                stream.Write(msg, 0, msg.Length);
    Console.WriteLine("Enviado: {0}", data);
            }
            client.Close();
        }

server.Stop();
```

Tenemos varias funciones clave:

- **TcpListener(Dirección, Puerto):**
  Esta función es clave, es el constructor de nuestro server, nos dice en qué interfaz de red escuchar (LAN, WAN o Loopback) y a su vez el puerto en el cual voy a escuchar a los clientes. Esta información

es crucial porque luego los clientes van a necesitar estos dos datos para llegar a mi servidor. Luego de esta función usamos Start() para comenzar el server.

- **AcceptTcpClient():**
  Esta función es bloqueante, quiere decir que va a detener mi programa en ese punto. Además de bloquear el programa va aquedar esperando que un cliente se conecte en mi puerto e IP. Una vez que esto sucede la función devuelve un objeto **TcpCLient** que representa a mi cliente con el cual puedo interactuar.

- **TcpCLient.GetStream() & stream.Read() & stream.Write():**
  Stream representa el canal de datos con mi cliente y estas tres funciones me permiten obtenerlo y operar sobre él, tanto como enviar o recibir datos de él.

TCP Client

```
static void Connect(String server, String message)
      {
  try
          {
              Int32 port = 16000;
    TcpClient client = new TcpClient(server, port);
              Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

               NetworkStream stream = client.GetStream();
              stream.Write(data, 0, data.Length);

    Console.WriteLine("Enviado: {0}", message);
    data = new Byte[256];
              String responseData = String.Empty;

              Int32 bytes = stream.Read(data, 0, data.Length);
              responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes);
    Console.WriteLine("Recibido: {0}", responseData);

              stream.Close();
              client.Close();
          }
  catch (ArgumentNullException e)
          {
    Console.WriteLine("ArgumentNullException: {0}", e);
          }
  catch (SocketException e)
          {
    Console.WriteLine("SocketException: {0}", e);
          }

  Console.WriteLine("\n Presiona Enter para continuar..");
          Console.Read();
      }
```

Funciones claves del lado cliente:

- **TcpCLient(Dirección servidor, Puerto):** Esta es la función de conexión al Server con la dirección y el puerto del mismo. Una vez realizada la conexión devuelve un objecto `TcpClient` válido para operar contra el server.
- **TcpCLient.GetStream() & stream.Read() & stream.Write():** La dinámica es muy parecida al lado Server pero desde el punto de vista del cliente. `Read()` va a leer los `Write()` del cliente y viceversa.

Conversión de Bytes a String y String a Byte

El Código ASCII

Para hacer posible el envió de `Strings` por medio del `Stream` de datos necesitamos codificarlos en el modelo **ASCII**. El Modelo **ASCII** básicamente es una codificación para caracteres en el cual cada uno tiene una representación en byte. **Por ejemplo**: A = 65, B = 66 (por eso es que al hacer ALT + 65 nos sale una A)

Decimal	Hex	Char	Decimal	Hex	Char
64	40	@	96	60	`
65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q
82	52	R	114	72	r
83	53	S	115	73	s
84	54	T	116	74	t
85	55	U	117	75	u
86	56	V	118	76	v
87	57	W	119	77	w
88	58	X	120	78	x
89	59	Y	121	79	y
90	5A	Z	122	7A	z
91	5B	[	123	7B	{
92	5C	\	124	7C	|
93	5D	]	125	7D	}
94	5E	^	126	7E	~
95	5F	_	127	7F	[DEL]

Convertir de String a ASCII para poder enviar datos

```
Byte[] data = System.Text.Encoding.ASCII.GetBytes("Mensaje a enviar");
stream.Write(data, 0, data.Length);
```

Usando la función **GetBytes** me permite hacer una conversión de un String a Código ASCII listo para enviar por la red

Convertir de ASCII a String para poder leer lo recibido

```
Byte[] data = new Byte[256]; Int32 NumeroDeBytes = stream.Read(data, 0, data.Length);
responseData = System.Text.Encoding.ASCII.GetString(data, 0, NumeroDeBytes) ;
```

La función **GetString** me permite hacer una conversión de ASCII a String para poder ser leido y usado como cualquier otro string.

Diagrama