Sistemas distribuidos y de tiempo real

2 .- Sockets

¿Qué son los sockets?

- Sockets = Conectores
- Objeto software mediante el que un cliente puede conectarse a un servidor estableciendo así un canal de comunicación.
- Parámetros que se suelen incluir
  - Dirección de la máquina a la que se quiere conectar
  - Puerto por el que se quiere conectar
  - Protocolo
- Se puede establecer con ellos una comunicación peer-to-peer
- Componente básico de comunicación
  - Interproceso (2 o más procesos en la misma o distinta máquina)
  - Intersistema (2 o más sistemas)

Comunicaciones en .NET

- Biblioteca de clases .NET facilita la programación de aplicaiones para la red y concretamente Internet
- Clases que se utilizan están en dos ámbitos
  - System.Net : Interfaz de programación para protocolos de red
  - System.Net.Sockets: Implementación administrada de la interfaz de Windows Sockets (WinSock).

  Comunicaciones
- Estos dos espacios de nombres proporcionan:
  1. Clases para trabajar con direcciones
  2. Clases para trabajar con sockets
  3. Clases que representan conexiones
  4. Clases que trabajan con el modelo request/response o petición/respuesta Web

1.Clases para trabajar con direcciones

- Todas pertenecen al espacio de nombres System.Net
- EndPoint: Identifica una direccion de red pero es una clase abstracta. Existen dos clases derivadas para usar su funcionalidad
  - IPEndPoint: Representa un punto final de red (IP + puerto)
  - IPAddress: Proporciona es una dirección IP
- Dns: Proporciona funcionalidad de resolución de nombres de dominio de una manera sencilla
- DnsPermission: Controla los derechos de acceso a los DNS's en la red
- DnsPermissionAtributte: Especifica los permisos para solicitar información a DNS's

2. Clases para trabajar con sockets (I)

System.Net.Sockets

- Socket: Implementa la interfaz de sockets definida por la universidad de Berkeley
- SocketException: Excepción que se inicia cuando se produce un error de socket
- NetworkStream: Proporciona la secuencia de datos subyacente para el acceso a la red
- MulticastOption: Contiene los valores de IPAddress usados para unirse y separarse de un grupo de multidifusión.
- LingerOption: Especifica si un objeto Socket seguirá estando conectado después de llamar al método Close, así como la duración de la conexión si hay datos pendientes que enviar

2. Clases para trabajar con sockets (II)

System.Net

- SocketAddress: Almacena información serializada procedente de clases derivadas de **EndPoint** (IPEndPoint, IPAddress)
- SocketPermission: Controla los derechos para realizar o aceptar conexiones en una dirección de transporte. Las instancias de **SocketPermission** controlan el permiso que aceptan conexiones o inician conexiones de **Socket**. Un permiso de **Socket** garantiza el acceso basado en el nombre del host o de la dirección IP, un número de puerto y un protocolo de transporte.
- SocketPermissionAttribute: Especifica acciones de seguridad para controlar conexiones **Socket**.

3. Clases que representan conexiones

- Lo que hacen es encapsular sockets
- Pertenecen al espacio de nombres System.Net.Sockets
- TcpClient: Proporciona conexiones cliente para servicios de red TCP. La clase **TcpClient** proporciona métodos sencillos para conectar, enviar y recibir secuencias de datos a través de una red en modo de bloqueo síncrono. Para que se pueda conectar debería haber un TCPListener al otro lado.
- TcpListener: Escucha las conexiones de los clientes de red TCP. La clase **TcpListener** proporciona métodos sencillos para escuchar y aceptar solicitudes de conexión entrantes en modo de bloqueo sincrónico.
- UdpClient: Proporciona servicios de red mediante datagramas (UDP)

4. Clases para trabajar con el modelo request/response Web (I)

- Pertenecen al espacio de nombres System.Net
- WebRequest: Realiza una solicitud a un URI (Uniform Resource Identifier, tambien se usa URL). Clase abstracta
  - FileWebRequest: Proporciona una implementación del sistema de archivos de la clase WebRequest.
  - HttpWebRequest: Proporciona una implementación específica de la clase WebRequest
- WebResponse: Proporciona una respuesta desde un URI. Clase abstracta
  - FileWebResponse: Proporciona una implementación específica de HTTP de la clase WebResponse.
  - HttpWebRessponse: Proporciona una implementación del sistema de archivos de la clase WebResponse.

4. Clases para trabajar con el modelo request/response Web (II)

- Cookie: Proporciona un conjunto de propiedades y métodos que se utilizan para administrar cookies. No se puede heredar esta clase.
- CookieCollection: Proporciona un contenedor de colección para instancias de la clase Cookie.
- CookieContainer: Proporciona un contenedor para una colección de objetos CookieCollection.
- CookieException: Excepción que se inicia cuando se produce un error al agregar un objeto Cookie a un objeto CookieContainer.

Clase IPAddress

- Contiene la dirección IP de un computador cuando está en Internet
- Al constructor de la clase IPAddress hay pasarle la dirección como matriz de bytes o un long. Pero es más cómodo utilizar el método Parse( ) para crear la dirección
  - IPAddress direccion1 = IPAddress.Parse(“156.35.151.107”);
  - IPAddress direccion2 = direccion2.Parse(“156.35.151.107”);
  - IPAddress direccion3 = new IPAddress(0x6B97239C);
- El último constructor toma el valor del long en orden inverso, es decir:

  0x6B97239C → IPAddress → “156.35.151.107”

Clase IPEndPoint

- Es similar a la clase IPAddress pero además de la dirección IP contiene el puerto del computador donde se quiere establecer la comunicación, determinándose así el punto final de conexión
- Propiedades
  - Address: Instancia de la clase IPAddress que contiene la dirección IP del punto final**.**
  - AddressFamily: Especifica la familia de direcciones que utiliza. Es una enumeración.
    - Para una dirección IPv4 debe ser **InterNetwork**
    - Para una dirección IPv6 debe ser **InterNetworkV6**
  - Port: Es el número de puerto del extremo final de conexión
- IPEndPoint puntoFinal = new IPEndPoint(0x6B97239C,1256)
- IPAddress direccion = IPAddress.Parse(“156.35.151.107”);

  IPEndPoint puntoFinal = new IPEndPoint(direccion,1256)

Clase Socket (I)

- Esta clase proporciona gran número de métodos para comunicarse entre procesos a través de la red.
- Si se utiliza protocolo TCP se utilizarán los métodos para el servidor:
  - Bind(): Para enlazar un socket a una dirección y un puerto
  - Listen(): Para escuchar peticiones de conexión
  - Accept(): Para procesar las solicitdes de conexión entrantes y devolver un socket de datos
  - Receive(): Para recibir datos del cliente
  - Send(): Para enviar datos al cliente
- Si se utiliza protocolo TCP se utilizarán los métodos para el cliente:
  - Connect(): Para conectarse al servidor
  - Receive(): Para recibir datos del servidor
  - Send(): Para enviar datos al servidor

Clase socket (II)

- Ambos, servidor y cliente utilizarán los siguientes métodos
- Terminar las conexiones
  - Shutdown(): Para deshabilitar el socket al terminar la conexión (1º)
  - Close(): Para liberar todos los recursos asociados al socket (después)
- Configuración de los sockets
  - SetSocketOption: Establecer opciones de configuración del socket
  - GetSocketOption: Obtener las opciones de configuración del socket
- Comprobar el estado de los sockets
  - Select(): Determina el estado de uno o varios sockets
  - Poll(): Determina el estado de un socket
- Propiedades que dan información útil
  - AddressFamily, Available, Blocking, Connected, Handle, LocalEndPoint ProtocolType, RemoteEndPoint, SocketType, SupportIPv4, SupportIPv6

Clase Socket (III)

- Send (Admite 4 sobrecargas)
  - public int Send(byte[]);
  - public int Send(byte[], SocketFlags);
  - public int Send(byte[], int, SocketFlags);
  - public int Send(byte[], int, int, SocketFlags);
- Receive (Admite 4 sobrecargas)
  - public int Receive(byte[]);
  - public int Receive(byte[], SocketFlags);
  - public int Receive(byte[], int, SocketFlags);
  - public int Receive(byte[], int, int, SocketFlags);

SocketFlags

- Es una enumeración que tiene los siguientes valores:
  - DontRoute: Enviar sin utilizar tablas de enrutamiento
  - OutOfBand: Procesar los datos fuera de linea
  - Partial: Enviar o recibir parcialmente el mensaje
  - Peek: Leer el mensaje entrante
  - Hay mas pero… mirar en la ayuda

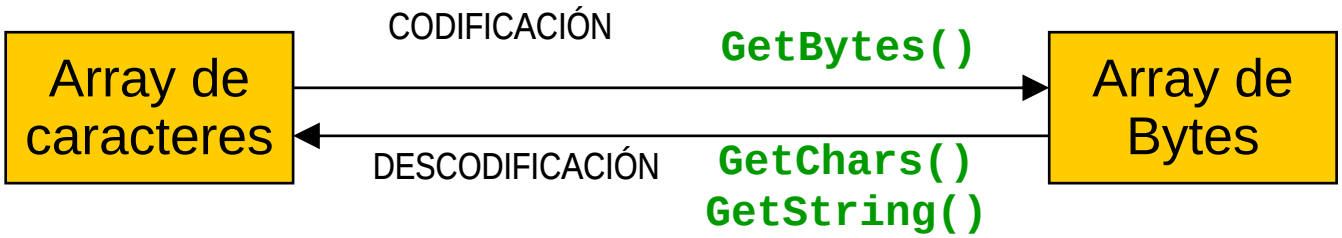
Clase Encoding (I)

- En System.Text se proporcionan clases para
  - Codificar arrays de caracteres o cadenas de caracteres en arrays de bytes, y viceversa
  - Descodificar arrays de bytes en arrays de caracteres o cadenas de caracteres
- Clase fundamental es Encoding (clase abstracta). Hay 4 implementaciones:
  - ASCIIEncoding
  - UnicodeEncoding
  - UTF7Encoding
  - UTF8Encoding
- UTF7, UTF8, UTF16 …..

UTF : Unicode Transformation Format

Clase Encoding (II)

- Objetivo de Encoding es codificar y decodificar



- Tiene tres métodos implementados en las clases hijas
  - GetBytes
    - public abstract int GetBytes(char[ ] chars, int charIndex, int charCount, byte[ ] bytes, int byteIndex, bool flush);
  - GetChars
    - public virtual char[] GetChars(byte[]);
    - public virtual char[] GetChars(byte[], int, int);
    - public abstract int GetChars(byte[], int, int, char[], int);
  - GetString
    - public virtual string GetString(byte[ ]);
    - public virtual string GetString(byte[ ], int, int);

Clase TCPClient (I)

- Para desarrollar aplicaciones sencillas, que solo realizan transferencias de datos síncronas (con bloqueo) hay clases que proveen una interfaz de programación más sencilla para las comunicaciones con sockets
- La clase representa a un cliente en una red TCP/IP, que funciona en modo bloqueante
- Proporciona métodos para conectarse y enviar / recibir datos
- Propiedades (algunas, el resto en la ayuda)
  - Active: Obtiene o establece un valor que indica si se ha realizado una conexión
  - Client: Obtiene o establece el objeto del socket subyacente
  - ReceiveBufferSize: Obtiene o establece el tamaño del buffer de recepción….

Clase TCPClient (II)

- Constructor
  - public TCPClient();
  - public TCPClient(AddressFamily);
  - public TCPClient(IPEndPoint);
  - public TCPClient(string,int);
- Métodos
  - Connect(): Conecta al cliente a un host mediante TCP (3 sobrecargas)
  - GetStream(): Devuelve la **NetworkStream** usada para enviar y recibir datos
  - Close():Cierra la conexión TCP y libera todos los recursos asociados a TCPClient

Clase TCPListener (I)

- Representa a un servidor en una red TCP /IP que funciona en modo bloqueante
- Proporciona métodos para escuchar y aceptar conexiones entrantes
- Propiedades
  - Active: Indica si objeto de la clase TCPListener está escuchando
  - LocalEndPoint: Permite obtener la dirección y el puerto usados para escuchar las solicitudes de conexión de los clientes
  - Server: Obtiene el objeto socket que sirve las peticiones

Clase TCPListener (II)

- Constructor
  - TCPListener(int) <= Obsoleto, se especificaba solo el puerto
  - TCPListener(IPAddress,int)
  - TCPListener(IPEndPoint)
- Métodos
  - Start: Inicializa el socket subyacente, lo enlaza a un extremo local y escucha los intentos de conexión entrantes que se colocan en la cola del socket.
  - AcceptTCPClient:Toma una petición de conexión de la cola de conexiones entrantes y devuelve un TcpClient que se puede utilizar para enviar y recibir datos.Es ideal para realizar una E/S bloqueante sencilla
  - AcceptSocket:Toma una petición de conexión de la cola de conexiones entrantes y devuelve un socket que se puede utilizar para enviar y recibir datos.
  - Stop:Cierra el agente de escucha y se pierden todas las solicitudes de conexión no aceptadas de la cola

Clase NetworkStream (I)

- La clase NetworkStream proporciona métodos para enviar y recibir datos a través de sockets de tipo secuencia (Stream) que funcionan en modo de bloqueo
- Constructor (4 sobrecargas)
  - NetworkStream(Socket)
  - NetworkStream(Socket,Boolean)
  - NetworkStream(Socket,FileAccess)
  - NetworkStream(Socket,FileAccess,Boolean)
- Generalmente, no se utiliza un constructor para crear una nueva NetworkStream, sino que se usa el método GetStream( ) de TcpClient para obtener una NetworkStream para el socket subyacente

Clase NetworkStream (II)

- Propiedades
  - CanRead: Si es true se pueden leer datos de la secuencia
  - CanWrite: Si es true se pueden escribir datos de la secuencia
  - DataAvailable: Si es true hay datos disponibles para leer de la secuencia
- Métodos
  - Read: Lee datos de la NetworkStream
    - public override int Read([In,Out] byte[ ] buffer,int offset,int size);
  - Write: Escribe datos en la NetworkStream
    - public override void Write(byte[ ] buffer,int offset,int size);
  - Close: Cierra la NetworkStream

Array de bytes que contiene los datos que se van a leerescribir en la NetworkStream

Número de bytes que se van a leer/escribir en la NetworkStream

Elemento del buffer donde se comienza a tomar datos para leerlos / escribirlos