DOCUMENTACIÓN

TRATAMIENTO DE FICHEROS XML Y ENVIO DE INFORMACIÓN

El correcto funcionamiento para la trasmisión y recepción de ficheros XML se realiza en una serie de clases. Principalmente, está compuesto por dos: una encargada de instanciar el mensaje, obteniendo los atributos que lo forman, y otra encargada de gestionar los contenderos de mensajes.

1. Clase Mensaje

1.1 Variables globales de la clase

Lo primero que tenemos que hay que indicar acerca de está clase son las variables globales de esta, que harán posible la consulta rápida de los principales atributos de un mensaje. Podemos dividir las variables en los siguientes grupos:

- Documento XML:
  - **Document mensaje:** Se trata del documento XML con el que se trabajará para determinar la información que contiene.
  - **Boolean correcto:** Se utiliza como flag a la hora de validar el XML contra el XSD.

- Atributos del receptor y emisor:
  - **String receptorID:** Se trata del identificadore del receptor.
  - **String puertoReceptor:** Se trata del puerto del receptor.
  - **String receptorIP:** Se trata de la dirección IP del receptor.
  - **String emisorIP:** Se trata de la dirección IP del emisor.
  - **String emisorID:** Se trata del identificador del emisor.
  - **String puertoEmisor:** Se trata del puerto del emisor.

- Atributos del mensaje:
  - **String idMensaje:** Se trata del identificador del mensaje.
  - **String horaGeneracion:** Se trata de la hora de envío del mensaje.
  - **String tipoMensaje:** Indica el tipo de mensaje.
  - **String protocolo:** Indica el protocolo que se está utilizando.

- Cuerpo del mensaje: se trata de un atributo de tipo element que guarda el cuerpo del mensaje, se conserva de esta forma para posteriormente tratarlo.

1.2 Constructores de la clase

Encontramos dos tipos de constructor, uno para tratar los mensajes recibidos y otro para tratar los mensajes a enviar. Por lo tanto, para tratar un mensaje recibido instanciaremos la clase pasando el XML por parámetro, lo que lanzará el constructor encargado de recibir mensajes. Por lo contario, para realizar un envío instanciaremos la clase pasando por parámetro un HashMap indicando la jerarquía de la cabecera, un elemento body y el DOM.

Para una mejor compresión mostraremos a continuación una representación de lo que significaría cada constructor:

- `public Mensaje(String xml):` En primer lugar, crearemos el documento XML partir del string que recibimos como parámetro, para posteriormente tratar la cabeza del mensaje y obtener así los valores de las etiquetas. Después repetiremos el proceso para los datos del emisor y el receptor (Aún no tratamos el body).
- `public Mensaje(HashMap<String,Object> cabe, Element body, File dom):`

`(cabe = cabecera del mensaje, body = cuerpo del mensaje, dom=XLS para comprobar que es correcto el mensaje)`

  En este segundo constructor sacaremos todos los atributos del hashmap que hemos pasado por parámetro y los almacenamos en las variables globales que hemos declarado. Llamaremos al método que genera un XML con la cabeza y el body que le pasamos por parámetro, si el mensaje XML no es nulo, validaremos dicho XML con el dom pasado por parámetro.

1.3 Métodos de la clase

- `private Document creaXML(HashMap<String,Object> cab,Element body):`

  En este primer método llevaremos a cabo la creación de un XML a partir de los parámetros que le pasemos, para ello lo primero que hacemos es crear el elemento raiz, la cabecera y el body. En cada uno de ellos, iremos recorriendo los nodos en profundidad, para ello tendremos en cuenta que, si hashmap almacena en un elemento un string, significa que ese elemento no tiene más hijos, en cambio sí almacena un elemento como una key significa que tendrá más hijos.

- `private void addRamasElem(HashMap<String,Object> lista, Element e,Document document):`

En este método obtendremos los árboles de profundidad mayor que 2 para posteriormente generar el DOM.

- `private boolean validaXML(File dom):`

  Valida el XML a partir de un XSD donde declaramos la estructura.

Seguidamente tenemos todos los métodos getters y setters para asignar u obtener valores de las variables globales.

2. Clase GestorMensajes

Esta clase será fundamental para gestionar todo tipo de mensajes, tanto los recibidos como aquellos que son enviados.

2.1 Atributos de la clase

En primer lugar, implementaremos dos objetos "LinkedList" con el fin de almacenar todos los mensajes enviados y recibidos:

- `public LinkedList<Mensaje> contenedor_de_mensajes_a_enviar = new LinkedList<>();`

- `public LinkedList<Mensaje> contenedor_de_mensajes_recibidos = new LinkedList<>();`

Declaramos dos instancias de cada protocolo de comunicación, para recibir mensajes por ambos protocolos:

- `private RecibeTcp recibeTcp;`
- `private RecibeUdp recibeUdp;`

Para finalizar, declararemos un objeto "mutex" para coordinar los hilos, y los puertos de ambos protocolos UDP y TCP:

- `private Object mutex = new Object();`
- `public int Puerto_PropioTcp;`
- `public int Puerto_PropioUdp;`

2.2 Constructores de la clase

Encontramos un único constructor al que se le pasa por parámetro el número de puerto TCP y el número de puerto UDP. Dicho constructor llamará al constructor de la clase Thread dado que se trata de un hilo, después asignará los valores de los puertos a las variables de la clase.

- `public GestorMensajes(int puerto_PropioTcp, int puerto_PropioUdp);`

2.3 Métodos de la clase

En cuanto a los métodos que componen la clase encontramos los siguientes:

- `public void EnviarMensaje():` Este método se encarga de enviar los mensajes que se encuentran en la cola, si dicha cola está vacía se duerme y volverá a mirar más tarde. Lo primero que hace es comprobar si tiene elementos en la cola, si no tiene se duerme durante un segundo, si por lo contrario tiene obtenemos el mensaje del contenedor y obtenemos el formato XML de este. Por último, según el protocolo que se utilice lo enviaremos por TCP o UDP.

- `public void AñadirMensajeContenedor(Mensaje msg):` Este método se encarga de añadir mensajes al contenedor de mensajes a enviar. Para solucionar la problemática de que dos hilos accedan al mismo tiempo al contenedor se utiliza mutual exclusión.

- `public Mensaje CogerMensajeDelContenedor():` Este método se encarga de coger mensajes al contenedor de mensajes recibidos. Para solucionar la problemática de que dos hilos accedan al mismo tiempo al contenedor se utiliza mutual exclusión.

- `public boolean ComprobarContenedorDeMensajes():` Este método se encarga de comprobar si el contenedor de mensajes recibidos se encuentra vacío o no. Esto se realiza consultando con la función .IsEmpty(). Para solucionar la problemática de que dos hilos accedan al mismo tiempo al contenedor se utiliza mutual exclusión.

- `public void EnviaTcp(String msg, String host, String puerto):` Este método se encarga enviar los mensajes vía TCP, se le necesita pasar por parámetro el mensaje, la IP de la máquina y el puerto. Para poder enviarlo primero establece el socket para comunicarse, después crea un flujo de salida por el cuál envía el mensaje y lo ciera.

- `public void EnviaUdp(String msg, String host, String puerto):` Este método se encarga enviar los mensajes vía UDP, se le necesita pasar por parámetro el mensaje, la IP de la máquina y el puerto. Para poder enviarlo primero establece el datagram socket UDP para comunicarse, después convertimos el mensaje a un array de bytes, creamos el datagrama y lo enviamos, una vez enviado se cierra el socket.

- `public HashMap<String,Object> generaCab(String p_e,String id_e,String ip_e,String p_r,String id_r,String ip_r, String tipo,String protocolo, String id_men):` Este método se encarga de generar la cabecera con la correspondiente jerarquía en formato HashMap pasado por parámetro todos los atributos necesarios para completarla.

- `public void ProcesaMensaje(String xml):` Este método se encarga de generar una instanciación de la clase Mensaje, donde se almacenará los datos del XML. Una vez creada la instancia se añade a la cola de mensajes recibidos.

- `public String TransformaMensaje(Mensaje ma):` Este método se encarga de generar un XML a partir de una instancia de la clase Mensaje. Por lo tanto, haciendo uso de los getters de dicha clase se pueden obtener todos los elementos necesarios para su elaboración. Dicho método nos devolverá un String con el mensaje a enviar.