4.3 Representación externa de datos y empaquetado.

La información que almacena un programa en ejecución es almacenada a través de estructuras de datos, sin embargo para comunicarse a través de un mensaje es necesario que estas sean convertidas a información pura, en otras palabras en bits para su reestructuración al llegar el mensaje a otro proceso. Debido a las diversas implementaciones de las estructuras de datos y los mismos datos, se deben utilizar técnicas para hacer que la información de un proceso a otro sea realmente la misma (Pueda ser utilizada de la misma forma). Estas son:

1. Se hace un acuerdo de utilizar un tipo de formato externo para transmisión y otro interno para recepción, si los sistemas fueran del mismo tipo, deberían mandar la información de forma nativa (Sin usar un formato externo).
2. El emisor avisa el tipo de formato que usará para que el receptor lo convierta a uno que pueda recibir solo de ser necesario.

Representación externa de datos: Estándar acordado para la representación de estructuras de datos y valores primitivos.

Empaquetado: Tomar una coleccion de items y ensamblarlos para ser enviados a través de un mensaje.

Desempaquetado: Uso de los datos del mensaje para ser convertidos en su forma original de estructura y valores primitivos.

Existen dos alternativas para la representación externa de datos y el empaquetado:

1. Representación mediante CORBA: Utilizando métodos en CORBA.
2. Serialización de objetos: Comúnmente usado en Java pero también existe para otros lenguajes de programación.

Ambas opciones son implementadas a través de un middleware sin participación del programador de la aplicación. El rendimiento y la fiabilidad son tomados en cuenta al hacer el empaquetado y desempaquetado. También se pueden hacer implementacion tan básicas como la comunicación usando ASCII.

Empaquetado en CORBA

Las operaciones de empaquetado se pueden generar automáticamente a partir de las especificaciones de los tipos de datos de los elementos que tienen que ser transmitidos. Los tipos de las estructuras de datos y de los elementos de datos básicos están descritos en CORBA IDL, que proporciona una notación para describir los tipos de los argumentos y los resultados de los métodos RMI.

4.3.2 Serialización de Objetos en Java

Declarar que una clase implementa la interfaz Serializable tiene el efecto de permitir que sus instancias sean serializables.

Serialización se refiere a la actividad de aplanar un objeto o un conjunto de objetos para obtener una forma lineal adecuada para ser almacenada en disco o para ser transmitida en un mensaje. La deserialización consiste en restablecer el estado de un objeto desde su estado lineal. Debe incluirse en la forma lineal alguna información sobre la clase de cada objeto, que se compone del nombre de la clase y un número de versión.

El número de versión está pensado para cambiar y reflejar cambios y modificaciones importantes en la clase. La serialización comprueba que se tiene la versión correcta de la clase.

Cuando un objeto es serializado, todos los objetos a los que referencia son serializados con él para asegurarse de que cuando el objeto sea reconstruido, todas sus referencias pueden ser rellenadas en el destino. Las referencias se serializan como apuntadores. Debe existir una correspondencia 1 - 1 entre las referencias y los apuntadores.

Para serializar un objeto, se escribe la información de su clase, los tipos y los nombres de los campos. Si los campos pertenecen a otra clase, se repite el proceso recursivamente. Cada clase recibe un apuntador, y ninguna clase es escribe más de una vez en el flujo de bytes.

La serialización y deserialización de los argumentos y los resultados de una invocación remota son llevados a cabo generalmente de forma automática por el middleware, sin ninguna participación del programador. Aquellas variables que no deberían ser serializadas se escriben como transient, como referencias a recursos locales (archivos o sockets).

El uso de la reflexión
La reflexión es la habilidad de preguntar sobre las propiedades de una clase. Hace posible que se creen las clases a partir de su nombre, y crea un constructor para una clase dada con unos argumentos dados. La serialización y deserialización es genérica: no hay necesidad de generar funciones de empaquetado especiales para cada tipo de objeto. Con la reflexión, la serialización de objetos encuentra el nombre de la clase del objeto, los nombres, tipos y valores de sus variables de instancia. Para la deserialización se utiliza el nombre de la clase en la forma serializada para crear un nuevo constructor con los tipos de argumentos requeridos, con el cual se crea una nueva instancia del objeto con aquellos valores leídos de la forma serializada.

4.3.3 Referencias a objetos remotos

Una referencia a un objeto remoto es un identificador para un objeto remoto que es válida a lo largo y ancho del sistema distribuido. En el mensaje de invocación se incluya una referencia a un objeto remoto que especifica cual es el objeto invocado.

Las referencias a objetos remotos deben ser únicas entre todos los procesos en las computadoras de un sistema distribuido. Cualquier intento de invocar objetos borrados debería producir un error en lugar de permitir el acceso a un objeto diferente.

Para que una referencia a un objeto remoto sea única, se puede concatenar la dirección IP del computador y el número de puerto, junto con el instante de tiempo de su creación y un número de objeto local.

En RMI, los objetos remotos viven en el proceso que los crea y sobreviven mientras el proceso continúe en ejecución. Los mensajes de invocación se envían a la dirección IP de

la referencia remota, y dentro del computador al proceso identificado por el número de puerto.

4.4 Comunicación Cliente - Servidor

La comunicación petición - respuesta es síncrona, ya que el proceso cliente se bloquea hasta que llega la respuesta del servidor. La respuesta del servidor es un acuse de recibo para el cliente. La comunicación cliente - servidor asíncrona es una alternativa que puede ser útil cuando los clientes puedan recuperar las respuestas más tarde.

El protocolo petición . respuesta, construido sobre datagramas, evita las sobrecargas asociadas con el protocolo TCP.

El protocolo de petición-respuesta. El siguiente protocolo está basado en un trio de primitivas de comunicación: hazOperacion, damePeticion, enviaRespuesta. Este protocolo de petición-respuesta, diseñado especialmente, hace corresponder a cada petición una respuesta. Podría estar diseñado para obtener ciertas garantías de entrega. Si se utilizan datagramas UDP, las garantías de entrega deben venir dadas por el protocolo petición-respuesta, donde el mensaje de respuesta del servidor sirve como acuse de recibo del mensaje de petición del cliente. La primitiva hazOperacion se utiliza en los clientes para invocar operaciones remotas. Su resultado es una respuesta RMI. El primer argumento de hazOperacion es una instancia de la clase RemoteObjectRef, que representa las referencias de los objetos remotos. Esta primitiva envía un mensaje de petición al servidor cuya dirección y puerto se especifican en la referencia de objeto remoto dada como argumento, después de hazOperacion se invoca el método recibe para conseguir la respuesta y lo devuelve a su invocador.

La primitiva damePeticion se usa en el servidor para hacerse con las peticiones de servicio, Cuando el servidor ha invocado el método sobre el objeto especificado, utiliza el método enviarRespuesta para mandar la respuesta al cliente.

La información transmitida en un mensaje de petición se acomoda de la siguiente manera:

El primer campo indica si es petición o respuesta, el segundo tiene el identificador de mensaje, el tercero es la referencia al objeto remoto empaquetada, el cuarto campo es un identificador del método a invocar.

Identificadores de mensaje: lo utiliza cualquier esquema de gestión de mensajes para proporcionar propiedades adicionales. Un identificador de mensajes tiene 2 partes:

1. Un identificador de petición

2. Un identificador para el proceso emisor

La primera parte hace que el identificador sea único para el emisor y la segunda lo hace único para el sistema distribuido

Modelo de fallos del protocolo petición-respuesta. Si las tres primitivas previamente mencionadas se implementan en datagramas UDP, adolecerán de los mismos fallos de comunicación que cualquier otro ejemplo de aplicación UDP. Esto es:

- Sufrirán fallos de omisión
- No se garantiza que los mensajes lleguen en el orden de emisión

Tiempos de espera límite: cuando hazOperacion supera un timeout, la opción más simple es devolver inmediatamente el control indicando al cliente que fracasó. Para compensar la posibilidad de la perdida de los mensajes hazOperacion manda el mensaje de petición de forma repetida hasta que obtiene una respuesta o está seguro que el retraso se debe a una falla del servidor.

Eliminación de mensajes de petición duplicados: cuando un mensaje de petición es retransmitido puede llevar al servidor a ejecutar una operación más de una vez, para evitarlo se diseñó al protocolo para reconocer mensajes sucesivos (del mismo cliente) con el mismo identificador de petición y eliminarlos.

Pérdida de mensajes respuesta: Si el servidor ya envió la respuesta y se recibe duplicada la petición, necesita volver a ejecutar la petición, a menos que haya almacenado el resultado original de la ejecución. Una operación idempotente es una operación que puede ser llevada a cabo repetidamente con el mismo efecto que si hubiera sido ejecutada una sola vez. Un servidor cuyas operaciones sean todas idempotentes no tendrán que tomar medidas especiales para evitar que se ejecuten más de una vez.

Historial: permite almacenar las respuestas enviadas a cada cliente sin tener que ejecutar las operaciones, contiene la respuesta, el identificador del mensaje y el identificador del cliente, el problema de esto es su costo de almacenamiento, el servidor debe decir cuando ya no es necesario almacenar un mensaje

Protocolo de intercambio de RPC. En la implementación de los distintos tipos de RPC se utilizan 3 protocolos:

- El protocolo petición
- El protocolo petición-respuesta
- El protocolo petición-respuesta-confirmación de la respuesta (RRA)

El protocolo petición puede utilizarse cuando el cliente no necesita el resultado ni confirmación de la respuesta. El protocolo petición-respuesta, no se utilizan mensaje especiales de acuse ya que respuesta del servidor sirve como confirmación.

El protocolo RRA está basado en el intercambio de estos 3 mensajes: el mensaje de reconocimiento de la respuesta contiene el id Petición del mensaje de respuesta que reconoce, esto hace que el servidor pueda descartar las entradas de su historial. Aunque el intercambio implica un mensaje adicional, no necesita bloquear al cliente, ya que el

reconocimiento puede transmitirse después de que la respuesta haya sido entregada al cliente.

Utilización de streams TCP para implementar el protocolo petición-respuesta: Una de las razones por la cual elegir la implementación de los protocolos petición-respuesta sobre streams TCP es el deseo de evitar la implementación de protocolos multipaquete, permitiendo la transmisión de argumentos y resultados de cualquier tamaño. Si se utiliza el protocolo TCP, se está asegurando que los mensajes de petición y de respuesta serán entregados de manera fiable, de modo que no será necesario un protocolo petición-respuesta para tratar los mensajes de retransmisión y filtrar los duplicados, y los historiales.

HTTP (HyperText Transfer Protocol), un ejemplo de protocolo petición-respuesta

Las peticiones de los clientes especifican un URL que incluye el nombre DNS del host del servidor web y un número de puerto *opcional,* además del identificador de un recurso en el servicio. Soporta un conjunto fijo de métodos (GET, PUT, POST, etc.) que son aplicables a todos los recursos. En este protocolo, cada objeto tiene sus propios métodos.

- *Negociación del contenido:* las peticiones de los clientes pueden incluir información sobre qué tipo de representación de datos pueden aceptar (por ejemplo lenguaje o tipo de medio), haciendo posible que el servidor pueda elegir la representación más apropiada para el usuario.
- *Autenticación:* se utilizan credenciales y desafíos para conseguir una autenticación del estilo *clave de acceso.*

Los recursos considerados como datos se proporcionan en forma de estructuras de tipo MIME tanto en los argumentos como en los resultados. ***Multipurpose Internet Mail Extension (Extensiones de Correo Electrónico Multipropósito, MIME)*** es un estándar para enviar mensajes de correo electrónico compuestos por varias partes conteniendo a la vez, por ejemplo, texto, imágenes y sonido.

Métodos HTTP

Cada petición de un cliente especifica el nombre de un método que habrá de ser aplicado al recurso en el servidor y el URL de dicho recurso. El mensaje de respuesta indica el estado de la petición. Los métodos considerados son los siguientes:

- ***GET:*** Pide el recurso cuyo URL se da como argumento. Se pueden añadir argumentos al URL; por ejemplo, un GET se puede utilizar para enviar el contenido de un formulario a un programa ***cgi*** que los tomará como entrada.

- ***HEAD:*** Esta petición es idéntica a GET, sólo que no devuelve datos. Sin embargo, devuelve toda la información sobre los datos, como el tiempo de la última modificación, su tipo o tamaño.

- ***POST:*** Especifica el URL de un recurso (por ejemplo un programa) que puede tratar los datos aportados con la petición. Este método está diseñado para:

    - Proporcionar un bloque de datos (por ejemplo los obtenidos en un formulario) a un proceso de gestión de datos como un servlet o un programa *cgi.*

    - Enviar un mensaje a un tablón de anuncios, lista de correo o grupo de noticias.
    - Modificar una base de datos con una operación de añadir registro.

- ***PUT:*** Indica que los datos aportados en la petición deben ser almacenados con la URL aportada como su identificador, ya sea como una modificación de datos existentes o como la creación de un recurso nuevo

- ***DELETE:*** El servidor borrará el recurso identificado por el URL El servidor no siempre permitirá esta función, en cuyo caso se devolverá una indicación de fallo.

- ***OPTIONS:*** El servidor proporciona al cliente una lista de métodos aplicables a un URL (por ejemplo, GET,HEAD,PUT) y sus requisitos especiales.

- ***TRACE:*** El servidor envía de vuelta el mensaje de petición. Se utiliza en procesos de depuración.

Contenido del mensaje: Cada mensaje HTTP *request* especifica el nombre de un método, el URL de un recurso, la versión del protocolo, algunas cabeceras y un cuerpo de mensaje opcional. Los campos de la cabecera contienen modificadores de la petición e información sobre el cliente, como las condiciones sobre la última fecha de modificación del recurso o los tipos de contenido aceptables.

Metodo	URL	version HTTP	Cabeceras	Cuerpo del mensaje
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/1.1		
Mensaje HTTP Request				

Un mensaje *Reply* especifica la versión del protocolo, un código de estado y su razón, algunas cabeceras y un cuerpo de mesaje opcional. El código de estado y la razón proporcionan un informe sobre el éxito o cualquier otra situación asociada al cumplimiento de la petición.

Versión HTTP	Código de estado	razón	Cabeceras	Cuerpo del mensaje
GET	200	OK		
Mensaje HTTP Reply				

El cuerpo del mensaje en los mensajes de petición o en los de respuesta contienen los datos asociados con la URL especificada en la petición. El cuerpo del mensaje tiene sus propias cabeceras especificando información sobre los datos, como la longitud, el tipo MIME, el conjunto de caracteres, la codificación del contenido y la último fecha de modificación.

Preguntas

¿Qué información debe incluirse en la forma lineal al serializar un objeto?

A. El nombre de la clase y un número de versión
B. Los nombres y los tipos de sus campos, recursivamente
C. El nombre de la clase del objeto, los nombres, tipos y valores de sus variables de instancia
D. Apuntadores de los objetos a los cuales hace referencia

¿Como se pueden mandar estructuras de datos y datos primitivos en un mensaje?

A. Enviando sólo a sistemas del mismo tipo de empaquetado y formato externo
B. A través de un acuerdo mutuo de formato externo y empaquetado
C. Utilizando cadenas de caracteres y enteros.
D. A través de bytes sin ningún tipo de formato especial.

¿En cuál protocolo el cliente puede seguir enviando inmediatamente después de haber enviado el mensaje de petición?

A. Protocolo petición
B. Protocolo petición-respuesta-confirmación de la respuesta
C. Protocolo petición-respuesta
D. En todos

¿Cuál es una de las razones por las cuales se utiliza streams TCP para implementar el protocolo petición-respuesta?

A. No usar implementación de protocolos multipaquete

B. Manejar Historiales de peticiones

C. Restringir el tamaño de los mensajes

D. Ninguna de las anteriores

En el protocolo HTTP, ¿Cuál es el método para eliminar un recurso del servidor?

A. DELETE
B. PUT
C. OPTIONS
D. TRACE