Las dos técnicas de invocación remota más prominentes para la comunicación en sistemas distribuidos:

• **El enfoque de llamada a procedimiento remoto (RPC)** extiende la abstracción de programación común de la llamada a procedimiento a entornos distribuidos, permitiendo que un proceso de llamada llame a un procedimiento en un nodo remoto como si fuera local.

• **La invocación de método remoto (RMI)** es similar a RPC pero para objetos distribuidos, con beneficios adicionales en términos de usar conceptos de programación orientada a objetos en sistemas distribuidos y también extender el concepto de referencia de objeto a los entornos distribuidos globales, y permitir el uso de referencias a objetos como parámetros en invocaciones remotas.

El ejemplo más antiguo y quizás el más conocido de un modelo más amigable para el programador fue la extensión del modelo de llamada a procedimiento convencional a sistemas distribuidos (la llamada a procedimiento remoto o modelo RPC), que permite a los programas clientes llamar a procedimientos de forma transparente en programas de servidor que se ejecutan en procesos separados y generalmente en diferentes computadoras del cliente.

En la década de 1990, el modelo de programación basado en objetos se amplió para permitir que los objetos en diferentes procesos se comuniquen entre sí mediante la invocación de métodos remotos (RMI). RMI es una extensión de la invocación de métodos locales que permite que un objeto que vive en un proceso invoque los métodos de un objeto que vive en otro proceso.

5.2 Protocolos de solicitud-respuesta

Esta forma de comunicación está diseñada para admitir los roles y los intercambios de mensajes en las interacciones típicas cliente-servidor. En el caso normal, la comunicación de solicitud-respuesta es síncrona porque el proceso del cliente se bloquea hasta que llega la respuesta del servidor. También puede ser confiable porque la respuesta del servidor es efectivamente un reconocimiento para el cliente. La comunicación asíncrona de solicitud-respuesta es una alternativa que puede ser útil en situaciones en las que los clientes pueden permitirse recuperar las respuestas más adelante. Los intercambios cliente-servidor se describen en los siguientes párrafos en términos de las operaciones de envío y recepción en la API de Java para datagramas UDP, aunque muchas implementaciones actuales usan flujos TCP. Un protocolo construido sobre datagramas evita sobrecargas innecesarias asociadas con el protocolo de flujo TCP. En particular:

• Los acuses de recibo son redundantes, ya que las solicitudes van seguidas de respuestas.

• Establecer una conexión implica dos pares adicionales de mensajes además del par requerido para una solicitud y una respuesta.

• El control de flujo es redundante para la mayoría de las invocaciones, que solo pasan pequeños argumentos y resultados.

El protocolo de solicitud-respuesta: El protocolo que describimos aquí se basa en un trío de primitivas de comunicación, doOperation, getRequest y sendReply.

Este protocolo de solicitud-respuesta coincide con las solicitudes de respuestas. Puede estar diseñado para proporcionar ciertas garantías de entrega. Si se usan datagramas UDP, las garantías de entrega deben ser proporcionadas por el protocolo de solicitud-respuesta, que puede usar el mensaje de respuesta del servidor como un acuse de recibo del mensaje de solicitud del cliente. Los clientes utilizan el método doOperation para invocar operaciones remotas. Sus argumentos especifican el servidor remoto y qué operación invocar, junto con la información adicional (argumentos) requerida por la operación. Su resultado es una matriz de bytes que contiene la respuesta. Se supone que el cliente que llama a doOperation reúne los argumentos en una matriz de bytes y desordena los resultados de la matriz de bytes que se devuelve. El primer argumento de doOperation es una instancia de la clase RemoteRef, que representa referencias para servidores remotos. Esta clase proporciona métodos para obtener la dirección de Internet y el puerto del servidor asociado. El método doOperation envía un mensaje de solicitud al servidor cuya dirección de Internet y puerto se especifican en la referencia remota dada como argumento. Después de enviar el mensaje de solicitud, doOperation invoca recibir para recibir un mensaje de respuesta, del cual extrae el resultado y lo devuelve al llamante. La persona que llama de doOperation se bloquea hasta que el servidor realiza la operación solicitada y transmite un mensaje de respuesta al proceso del cliente. getRequest es utilizado por un proceso de servidor para adquirir solicitudes de servicio.

Cuando el servidor ha invocado la operación especificada, utiliza sendReply para enviar el mensaje de respuesta al cliente. Cuando el cliente recibe el mensaje de respuesta, doOperation original se desbloquea y la ejecución del programa del cliente continúa. La información a transmitir en un mensaje de solicitud o un mensaje de respuesta es:

messageType int (0=Request, 1= Reply)

requestId int

remoteReference RemoteRef

operationId int or Operation

arguments // array of bytes

El primer campo indica si el mensaje es una solicitud o un mensaje de respuesta. El segundo campo, requestId, contiene un identificador de mensaje. Un doOperation en el cliente genera un requestId para cada mensaje de solicitud, y el servidor copia estos ID en los mensajes de respuesta correspondientes. Esto permite que doOperation verifique que un mensaje de respuesta sea el resultado de la solicitud actual, no una llamada anterior

retrasada. El tercer campo es una referencia remota. El cuarto campo es un identificador para la operación a invocar.

Identificadores de mensajes: Cualquier esquema que implique la gestión de mensajes para proporcionar propiedades adicionales tales como la entrega confiable de mensajes o la comunicación de solicitud-respuesta requiere que cada mensaje tenga un identificador de mensaje único por el cual pueda ser referenciado. Un identificador de mensaje consta de dos partes:

1. un requestId, que es tomado de una secuencia creciente de enteros por el proceso de envío;

2. un identificador para el proceso del remitente, por ejemplo, su puerto y dirección de Internet.

La primera parte hace que el identificador sea único para el remitente, y la segunda parte lo hace único en el sistema distribuido. (La segunda parte se puede obtener de forma independiente, por ejemplo, si UDP está en uso, a partir del mensaje recibido).

Cuando el valor de requestId alcanza el valor máximo para un entero sin signo (por ejemplo, 2^32 - 1) se restablece a cero. La única restricción aquí es que la vida útil de un identificador de mensaje debe ser mucho menor que el tiempo necesario para agotar los valores en la secuencia de enteros. Modelo de falla del protocolo de solicitud-respuesta • Si las tres primitivas doOperation, getRequest y sendReply se implementan sobre datagramas UDP, entonces sufren las mismas fallas de comunicación. Es decir:

• Sufren fallas por omisión.

• No se garantiza que los mensajes se entreguen en orden de remitente.

Además, el protocolo puede sufrir fallas en los procesos. Asumimos que los procesos tienen fallas de bloqueo. Es decir, cuando se detienen, permanecen detenidos, no producen un comportamiento bizantino. Para permitir ocasiones en las que un servidor ha fallado o se ha eliminado un mensaje de solicitud o respuesta, doOperation utiliza un tiempo de espera cuando está esperando recibir el mensaje de respuesta del servidor. La acción tomada cuando se produce un tiempo de espera depende de las garantías de entrega que se ofrecen.

Tiempos de espera: Hay varias opciones sobre lo que puede hacer doOperation después de un tiempo de espera. La opción más simple es regresar inmediatamente de doOperation con una indicación al cliente de que la operación ha fallado. Este no es el enfoque habitual: el tiempo de espera puede deberse a que se perdió el mensaje de solicitud o respuesta y, en este último caso, la operación se habrá realizado. Para compensar la posibilidad de mensajes perdidos, doOperation envía el mensaje de solicitud repetidamente hasta que recibe una respuesta o está razonablemente seguro de que el retraso se debe a la falta de respuesta del servidor en lugar de a los mensajes perdidos. Finalmente, cuando regresa doOperation, indicará al cliente por una excepción que no se recibió ningún resultado. Descartar mensajes de solicitud duplicados • En los casos en que el mensaje de solicitud se

retransmite, el servidor puede recibirlo más de una vez. Por ejemplo, el servidor puede recibir el primer mensaje de solicitud, pero tarda más que el tiempo de espera del cliente para ejecutar el comando y devolver la respuesta. Esto puede llevar al servidor a ejecutar una operación más de una vez para la misma solicitud. Para evitar esto, el protocolo está diseñado para reconocer mensajes sucesivos (del mismo cliente) con el mismo identificador de solicitud y para filtrar duplicados. Si el servidor aún no ha enviado la respuesta, no necesita realizar ninguna acción especial: transmitirá la respuesta cuando haya terminado de ejecutar la operación.

Mensajes de respuesta perdidos: Si el servidor ya ha enviado la respuesta cuando recibe una solicitud duplicada, deberá ejecutar la operación nuevamente para obtener el resultado, a menos que haya almacenado el resultado de la ejecución original. Algunos servidores pueden ejecutar sus operaciones más de una vez y obtener los mismos resultados cada vez. Una operación idempotente es una operación que se puede realizar repetidamente con el mismo efecto que si se hubiera realizado exactamente una vez. Por ejemplo, una operación para agregar un elemento a un conjunto es una operación idempotente porque siempre tendrá el mismo efecto en el conjunto cada vez que se realiza, mientras que una operación para agregar un elemento a una secuencia no es una operación idempotente porque extiende la secuencia cada vez que se realiza. Un servidor cuyas operaciones son todas idempotentes no necesita tomar medidas especiales para evitar ejecutar sus operaciones más de una vez.

Historial: Para los servidores que requieren la retransmisión de respuestas sin volver a ejecutar las operaciones, se puede utilizar un historial. El término "historial" se utiliza para referirse a una estructura que contiene un registro de mensajes (de respuesta) que se han transmitido. Una entrada en un historial contiene un identificador de solicitud, un mensaje y un identificador del cliente al que se envió. Su propósito es permitir que el servidor retransmita mensajes de respuesta cuando los procesos del cliente los soliciten. Un problema asociado con el uso de un historial es su costo de memoria. Un historial se volverá muy grande a menos que el servidor pueda decir cuándo los mensajes ya no serán necesarios para la retransmisión. Como los clientes solo pueden hacer una solicitud a la vez, el servidor puede interpretar cada solicitud como un reconocimiento de su respuesta anterior. Por lo tanto, el historial debe contener solo el último mensaje de respuesta enviado a cada cliente. Sin embargo, el volumen de mensajes de respuesta en el historial de un servidor aún puede ser un problema cuando tiene una gran cantidad de clientes. Esto se ve agravado por el hecho de que, cuando finaliza un proceso del cliente, no reconoce la última respuesta que recibió; por lo tanto, los mensajes en el historial normalmente se descartan después de un período de tiempo limitado. Estilos de protocolos de intercambio

Tres protocolos, que producen comportamientos diferentes en presencia de fallas de comunicación, se utilizan para implementar varios tipos de comportamiento de solicitud. Fueron identificados originalmente por Spector [1982]:

- El protocolo de solicitud (R);

- El protocolo de solicitud-respuesta (RR);

• El protocolo de solicitud-respuesta-confirmación de respuesta (RRA).

En el protocolo R, el cliente envía un único mensaje de solicitud al servidor. El protocolo R se puede usar cuando no hay ningún valor que devolver de la operación remota y el cliente no requiere confirmación de que la operación se haya ejecutado. El cliente puede continuar inmediatamente después de enviar el mensaje de solicitud, ya que no es necesario esperar un mensaje de respuesta. Este protocolo se implementa sobre datagramas UDP y, por lo tanto, sufre los mismos fallos de comunicación. El protocolo RR es útil para la mayoría de los intercambios cliente-servidor porque se basa en el protocolo de solicitud-respuesta. No se requieren mensajes de confirmación especiales, ya que el mensaje de respuesta de un servidor se considera como una confirmación del mensaje de solicitud del cliente. Del mismo modo, una llamada posterior de un cliente puede considerarse como un reconocimiento del mensaje de respuesta de un servidor. Como hemos visto, la comunicación las fallas debidas a la pérdida de datagramas UDP pueden enmascararse mediante la retransmisión de solicitudes con filtrado duplicado y el almacenamiento de respuestas en un historial de retransmisión.

El protocolo RRA se basa en el intercambio de tres mensajes: solicitud-respuesta, confirmación de respuesta. El mensaje de confirmación de confirmación contiene el requestId del mensaje de respuesta que se confirma. Esto permitirá que el servidor descarte entradas de su historial. La llegada de un requestId en un mensaje de acuse de recibo se interpretará como el acuse de recibo de todos los mensajes de respuesta con requestIds inferiores, por lo que la pérdida de un mensaje de acuse de recibo es inofensiva. Aunque el intercambio implica un mensaje adicional, no es necesario que bloquee al cliente, ya que el acuse de recibo puede transmitirse después de que se haya dado la respuesta al cliente. Sin embargo, utiliza recursos de procesamiento y de red. El ejercicio 5.10 sugiere una optimización del protocolo RRA.

Uso de flujos TCP para implementar el protocolo de solicitud-respuesta

A menudo es difícil decidir un tamaño apropiado para el búfer en el que recibir datagramas. En el protocolo de solicitud-respuesta, esto se aplica a las memorias intermedias utilizadas por el servidor para recibir mensajes de solicitud y por el cliente para recibir respuestas. La longitud limitada de los datagramas (generalmente 8 kilobytes) puede no considerarse adecuada para su uso en sistemas RMI o RPC transparentes, ya que los argumentos o resultados de los procedimientos pueden ser de cualquier tamaño. El deseo de evitar la implementación de protocolos de paquetes múltiples es una de las razones para elegir implementar protocolos de solicitud-respuesta a través de flujos TCP, lo que permite transmitir argumentos y resultados de cualquier tamaño. En particular, la serialización de objetos Java es un protocolo de flujo que permite enviar argumentos y resultados a través de flujos entre el cliente y el servidor, lo que hace posible que las colecciones de objetos de cualquier tamaño se transmitan de manera confiable. Si se utiliza el protocolo TCP, garantiza que los mensajes de solicitud y respuesta se entreguen de manera confiable, por lo que no es necesario que el protocolo de solicitud-respuesta se encargue de la retransmisión de mensajes y el filtrado de duplicados o con historiales.

Adicionalmente el mecanismo de control de flujo permite pasar grandes argumentos y resultados sin tomar medidas especiales para evitar abrumar al receptor. Por lo tanto, el protocolo TCP se elige para los protocolos de solicitud-respuesta porque puede simplificar su implementación. Si se envían sucesivas solicitudes y respuestas entre el mismo par cliente-servidor a través de la misma secuencia, la sobrecarga de la conexión no necesita aplicarse a cada invocación remota. Además, la sobrecarga debida a los mensajes de confirmación de TCP se reduce cuando sigue un mensaje de respuesta poco después de un mensaje de solicitud. Sin embargo, si la aplicación no requiere todas las facilidades ofrecidas por TCP, se puede implementar un protocolo más eficiente y especialmente diseñado sobre UDP. Por ejemplo, Sun NFS no requiere soporte para mensajes de tamaño ilimitado, ya que transmite bloques de archivos de tamaño fijo entre el cliente y el servidor. Además de eso, sus operaciones están diseñadas para ser idempotentes, por lo que no importa si las operaciones se ejecutan más de una vez para retransmitir mensajes de respuesta perdidos, lo que hace innecesario mantener un historial.

HTTP: un ejemplo de un protocolo de solicitud-respuesta • El Protocolo de transferencia de hipertexto (HTTP) utilizado por los clientes del navegador web para realizar solicitudes a los servidores web y recibir respuestas de ellos. Los servidores web administran los recursos implementados de diferentes maneras:

• Como datos, por ejemplo, el texto de una página HTML, una imagen o la clase de un applet;

• Como programa, por ejemplo, servlets [java.sun.com III] o programas PHP o Python que se ejecutan en el servidor web.

Las solicitudes de los clientes especifican una URL que incluye el nombre de host DNS de un servidor web y un número de puerto opcional en el servidor web, así como el identificador de un recurso en ese servidor. HTTP es un protocolo que especifica los mensajes involucrados en un intercambio de solicitud-respuesta, los métodos, argumentos y resultados, y las reglas para representarlos (ordenarlos) en los mensajes. Admite un conjunto fijo de métodos (GET, PUT, POST, etc.) que son aplicables a todos los recursos del servidor. Es diferente a los protocolos descritos anteriormente, donde cada servicio tiene su propio conjunto de operaciones. Además de invocar métodos en recursos web, el protocolo permite la negociación de contenido y la autenticación con estilo de contraseña: Negociación de contenido: las solicitudes de los clientes pueden incluir información sobre qué representaciones de datos pueden aceptar (por ejemplo, idioma o tipo de medio), lo que permite servidor para elegir la representación más adecuada para el usuario.

Autenticación: las credenciales y los desafíos se utilizan para admitir la autenticación con estilo de contraseña. En el primer intento de acceder a un área protegida por contraseña, la respuesta del servidor contiene un desafío aplicable al recurso.

Cuando un cliente recibe un desafío, hace que el usuario escriba un nombre y una contraseña y envía las credenciales asociadas con solicitudes posteriores.

HTTP se implementa sobre TCP. En la versión original del protocolo, cada interacción cliente-servidor consistía en los siguientes pasos:

• El cliente solicita y el servidor acepta una conexión en el puerto predeterminado del servidor o en un puerto especificado en la URL.

• El cliente envía un mensaje de solicitud al servidor.

• El servidor envía un mensaje de respuesta al cliente.

• La conexión está cerrada.

Sin embargo, establecer y cerrar una conexión para cada intercambio de solicitud de respuesta es costoso, sobrecarga el servidor y provoca que se envíen demasiados mensajes a través de la red. Teniendo en cuenta que los navegadores generalmente hacen múltiples solicitudes al mismo servidor. El cliente o el servidor pueden cerrar una conexión persistente en cualquier momento enviando una indicación al otro participante. Los servidores cerrarán una conexión persistente cuando haya estado inactiva durante un período de tiempo. Es posible que un cliente reciba un mensaje del servidor que indique que la conexión está cerrada mientras está enviando otra solicitud o solicitudes. En tales casos, el navegador reenviará las solicitudes sin la participación del usuario, siempre que las operaciones involucradas sean idempotentes. Por ejemplo, el método GET descrito a continuación es idempotente. Cuando se trata de operaciones no idempotentes, el navegador debe consultar al usuario sobre qué hacer a continuación. Las solicitudes y respuestas se agrupan en mensajes como cadenas de texto ASCII, pero los recursos pueden representarse como secuencias de bytes y pueden comprimirse. El uso de texto en la representación de datos externos ha simplificado el uso de HTTP para los programadores de aplicaciones que trabajan directamente con el protocolo. En este contexto, una representación textual no agrega mucho a la longitud de los mensajes. Los recursos de datos se suministran como estructuras tipo MIME en argumentos y resultados.

Las Extensiones multipropósito de correo de Internet (MIME), especificadas en RFC 2045 [Freed y Borenstein 1996], son un estándar para enviar datos de múltiples partes que contienen, por ejemplo, texto, imágenes y sonido en mensajes de correo electrónico. Los datos tienen como prefijo su tipo MIME para que el destinatario sepa cómo manejarlos. Un tipo MIME especifica un tipo y un subtipo, por ejemplo, text / plain, text / html, image / gif o image / jpeg. Los clientes también pueden especificar los tipos MIME que están dispuestos a aceptar.

Métodos HTTP • Cada solicitud de cliente especifica el nombre de un método que se aplicará a un recurso en el servidor y la URL de ese recurso. La respuesta informa sobre el estado de la solicitud. Las solicitudes y respuestas también pueden contener datos de recursos, el contenido de un formulario o la salida de un recurso de programa ejecutado en el servidor web. Los métodos incluyen lo siguiente:

GET: solicita el recurso cuya URL se proporciona como argumento. Si la URL se refiere a datos, el servidor web responde devolviendo los datos identificados por esa URL. Si la URL se refiere a un programa, el servidor web ejecuta el programa y devuelve su salida al

cliente. Se pueden agregar argumentos a la URL; por ejemplo, GET se puede usar para enviar el contenido de un formulario a un programa como argumento. La operación GET se puede condicionar a la fecha en que se modificó por última vez un recurso. GET también se puede configurar para obtener partes de los datos. Con GET, toda la información para la solicitud se proporciona en la URL.

HEAD: esta solicitud es idéntica a GET, pero no devuelve ningún dato. Sin embargo, devuelve toda la información sobre los datos, como la hora de la última modificación, su tipo o su tamaño.

POST: especifica la URL de un recurso (por ejemplo, un programa) que puede manejar los datos suministrados en el cuerpo de la solicitud. El procesamiento realizado en los datos depende de la función del programa especificado en la URL. Este método se utiliza cuando la acción puede cambiar los datos en el servidor. Está diseñado para tratar con:

• proporcionar un bloque de datos a un proceso de manejo de datos, como un servlet, por ejemplo, enviar un formulario web para comprar algo de un sitio web;

• publicar un mensaje en una lista de correo o actualizar detalles de los miembros de la lista;

• extender una base de datos con una operación de agregar.

PUT: solicita que los datos suministrados en la solicitud se almacenen con la URL dada como su identificador, ya sea como una modificación de un recurso existente o como un nuevo recurso.

BORRAR: El servidor elimina el recurso identificado por la URL dada. Es posible que los servidores no siempre permitan esta operación, en cuyo caso la respuesta indica un error.

OPCIONES: El servidor proporciona al cliente una lista de métodos que permite aplicar a la URL dada (por ejemplo, GET, HEAD, PUT) y sus requisitos especiales.

RASTREO: el servidor devuelve el mensaje de solicitud. Se utiliza con fines de diagnóstico. Las operaciones PUT y DELETE son idempotentes, pero POST no es necesariamente así porque puede cambiar el estado de un recurso. Los otros son operaciones seguras en el sentido de que no cambian nada. Las solicitudes descritas anteriormente pueden ser interceptadas por un servidor proxy. Los servidores proxy pueden almacenar en caché las respuestas a GET y HEAD.

Preguntas

¿Es una técnica de invocación remota que utiliza objetos distribuidos y permitir el uso de referencias a objetos como parámetros en invocaciones remotas?

A. RMI

B. RPC

C. Protocolo petición-respuesta

D. Cobra

¿En un caso normal, la comunicación de solicitud-respuesta es?

A. Síncrona

B. Asíncrona

C. No bloqueante

D. Ninguna de las anteriores

¿Cuál es una de las 3 primitivas de comunicación principales del protocolo solicitud-respuesta

A. sendReply

B. getResponse

C. operationDo

D. setResponse

¿Cuál es el resultado del metodo doOperation?

A. Una matriz de bytes que contiene la información de la respuesta

B. Un objeto remoto

C. La referencia a un objeto remoto

D. La respuesta

¿Qué sucede cuando el valor requestId alcanza el valor de (2^32)-1?

A. Toma el valor de 0

B. Toma el valor de -1

C. Toma el valor de 2^32

D. Manda un error