Protocolo Solicitud-Respuesta

Virginia Padilla

Universidad Nacional Experimental de Guayana

virginiapadillas@gmail.com

19 de noviembre de 2024

Contenido

Protocolo Solicitd Respuesta

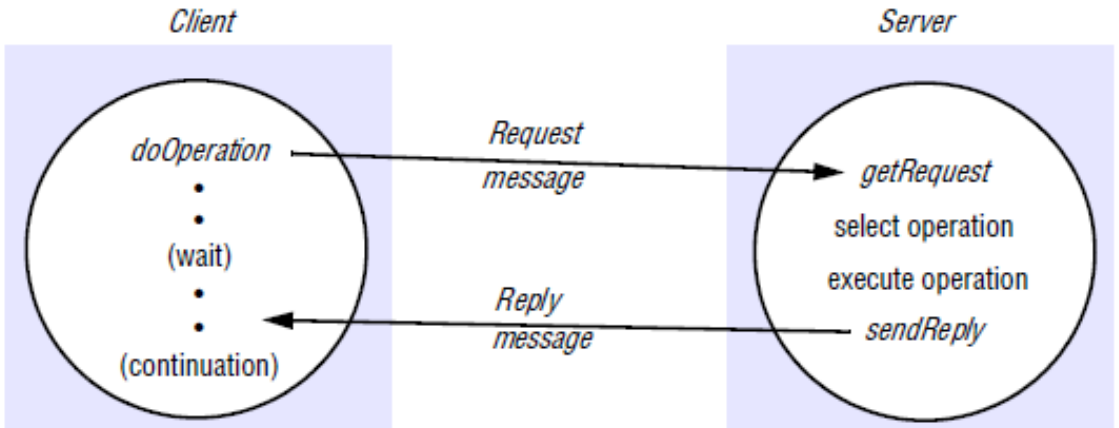
Idea básica

Diseñado para apoyar los roles y los intercambios de mensajes en interacciones cliente-servidor. En la mayoría de los casos, la comunicación solicitud-respuesta es síncrono porque el proceso del cliente se bloquea hasta que llega la respuesta del servidor. La respuesta del servidor es un reconocimiento al cliente.

Protocolo Solicitud Respuesta.

Primitivas de Comunicación

- *doOperation*: clientes invocan operaciones remotas con argumentos que especifican el servidor y la operación a invocar.
- *getRequest* utilizado por un proceso de servidor para adquirir solicitudes de servicio.
- *sendReply* El servidor usa *sendReply* para enviar el mensaje de respuesta al cliente.



Estructura del Mensaje

Información transmite en un mensaje:

- Primer campo indica si el mensaje es una Solicitud o un mensaje de Respuesta
- Segundo campo, requestId, es un identificador de mensaje. Una operación en el cliente genera un requestId para cada mensaje de solicitud y el servidor copia estos ID en los mensajes de respuesta correspondientes.
- Tercer campo es una referencia remota.
- Cuarto campo es un identificador para la operación a invocar.

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*// array of bytes*

Modelo de Fallas

Modelo de Fallas

Si las tres primitivas *doOperation getRequest* y *sendReply* se implementan sobre datagramas*UDP*:

- Sufren fallas por omisión.
- No se garantiza que los mensajes se entreguen en el orden del remitente.

Nota

Para permitir las ocasiones en las que un servidor falla o que se descarta una solicitud o un mensaje de respuesta, *doOperation* usa un *timeout* cuando está esperando obtener mensaje de respuesta del servidor.

Tiempos de Espera

Opciones en operaciones *doOperation* después de un *timeout*

- Lo más simple es indicar al cliente que *doOperation* ha fallado.
- Este no es el enfoque habitual ya que el tiempo de espera puede haber ocurrido debido a la pérdida del mensaje de solicitud o de respuesta. Si se pierde el mensaje de respuesta, se habría realizado la operación en el servidor.

Para compensar la pérdida de mensajes:

- *doOperation* envía el mensaje de solicitud repetidamente hasta que recibe una respuesta o es razonablemente seguro de que el retraso se debe a la falta de respuesta del servidor en lugar de mensajes perdidos.
- Eventualmente, *doOperation* regresa con una excepción, que le indica al cliente que no se recibió resultado alguno.

Mensajes Duplicados

Retransmisión de Mensajes

- En los casos en que el mensaje de solicitud sea retransmitido, el servidor puede recibirlo más de una vez.
- Por ejemplo, el servidor puede recibir el primer mensaje de solicitud, pero tomar más tiempo que el tiempo de espera del cliente para ejecutar el comando y devolver la respuesta.

Retransmisión de Mensajes

- Esto puede llevar a que el servidor ejecute 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, para filtrar los duplicados.

Mensajes Perdidos

Mensajes Perdidos

- Si el servidor ya ha enviado la respuesta cuando recibe una solicitud duplicada, ejecuta la operación nuevamente para obtener el resultado, a menos que haya almacenado el resultado de la ejecución original.
- Algunos servidores pueden ejecutar su operaciones más de una vez y obtener los mismos resultados cada vez.

Operaciones `idempotente`

- Es una operación que se puede realizar repetidamente con el mismo efecto que si se hubiera realizado exactamente una vez.
- Ejemplo, una operación para agregar un elemento para configurar es una operación de `idempotente`
- Una operación para agregar un elemento a una secuencia no es una operación `idempotente`.

Historial

Servidores con `historial`

Servidores que requieren una retransmisión de respuestas sin repetir la ejecución de las operaciones

- '`historial`' se usa para referirse a una estructura que contiene registros de los mensajes (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 los mensajes de respuesta cuando los procesos del cliente los soliciten.

Historial

Servidores con `historial`

- El `historial` se volverá grande a menos que el servidor pueda decir que ya no se necesita los mensajes para la retransmisión.
- A medida que los clientes pueden hacer una sola solicitud a la vez, el servidor puede interpretar cada solicitud como un acuse de recibo de su respuesta anterior.
- El `historial` necesita contener solo el último mensaje de respuesta enviado a cada cliente.

Problemas con el `historial`

- El volumen de mensajes de respuesta en el `historial` puede ser un problema cuando tiene una gran cantidad de clientes.
- Se agrava cuando termina un proceso del cliente y no se reconoce la última respuesta que ha recibido, por lo que, los mensajes se descartan después de un período de tiempo

Estilos del protocolo de invocación remota

Estilos del protocolo de invocación remota

- El protocolo Solicitud (R). El cliente envía un mensaje de solicitud único al servidor.
- El protocolo de solicitud-respuesta (RR). Es útil para la mayoría de los intercambios de clientes-servidores.
- El protocolo Solicitud-Respuesta (RRA). Se basa en el intercambio de tres mensajes: Solicitud, respuesta y reconocimiento.

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

Estilos del protocolo de invocación remota

Estilos del protocolo de invocación remota con UDP

- El protocolo Solicitud (R). El cliente puede procesar inmediato después de que se envíe el mensaje de solicitud, ya que no es necesario esperar un mensaje de respuesta.
- El protocolo de solicitud-respuesta (RR). No se requieren mensajes de acuse de recibo especial, ya que el mensaje de respuesta de un servidor se considera un acuse de recibo del mensaje de solicitud del cliente.
- El protocolo Solicitud-Respuesta (RRA). El mensaje de reconocimiento contiene el *requestId* del mensaje de respuesta.

Estilos del protocolo de invocación remota

Estilos del protocolo de invocación remota con UDP: Fallas

- El protocolo Solicitud (R) puede utilizar cuando no hay valor que deba ser devuelto desde la operación remota y el cliente no requiere confirmación de que la operación haya sido ejecutada.
- El protocolo de solicitud-respuesta (RR). Las fallas de comunicación por perdidas de datagramas UDP pueden ser enmascarados por la retransmisión de las solicitudes con un filtrado duplicado y se ahorran las respuestas en el historial para la retransmisión.
- El protocolo Solicitud-Respuesta (RRA). La llegada de un *requestId* en un mensaje de acuse de recibo se interpretará como un acuse de recibo de todos los mensajes de respuesta con requestIds más bajos.

TCP para implementar el protocolo de solicitud-respuesta

TCP para implementar el protocolo de solicitud-respuesta

- Los mensajes de solicitud y respuesta se entregan de manera confiable: no es necesario que el protocolo de solicitud-respuesta se ocupe de la retransmisión de mensajes ni el filtrado de duplicados o con historiales.
- Mecanismo de control de flujo permite pasar grandes argumentos y resultados sin medidas especiales.

¿ Por qué se elige?

- Simplifica la implementación del protocolo solicitud-respuesta.
- Si las solicitudes y respuestas sucesivas entre el mismo par cliente-servidor se envían en el mismo flujo, no es necesario la sobrecarga de conexión en invocaciones remotas.
- La sobrecarga debida a los acuse de recibo se reduce con los mensaje de respuesta después de un mensaje de solicitud.

References


Van Steen M , Tanenbaum, A (2017)
Distributed Systems
Pearson Education, Inc.


Coulouris G, Dollimore J, Kindberg T, Blair G (2011)
Distributed Systems: Concepts and Design
Addison-Wesley Publishing Company.


Veríssimo, P. and Rodrigues, L. (2012)
Distributed Systems for System Architects
Springer.


Limoncelli T, Strata R, Hogan C. (2014)
The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems
Addison Wesley.

Fin