Paso de mensajes

En ciencias de la computación, el **paso de mensajes** es un paradigma de programación ampliamente usado en el software moderno. Sus aplicaciones cubren un amplio campo, y puede usarse desde para garantizar que los diferentes objetos que constituyen un programa informático puedan trabajar de forma coherente entre ellos hasta para permitir que una tarea pueda ejecutarse de forma sincronizada entre varios ordenadores. Es uno de los conceptos clave en modelos de programación concurrente, programación distribuida y programación orientada a objetos.

De forma abstracta, se llama *mensaje* a una porción de información que un proceso emisor envía a un destinatario (El cual puede ser otro proceso, un actor o un objeto). El modelo de paso de mensajes es el que define los métodos y funciones para poder llevar a cabo el envío de un mensaje de un proceso emisor a un destinatario. Supone un enfoque opuesto al paradigma tradicional en el cual los procesos, funciones y subrutinas sólo podían ser llamados directamente a través de su nombre.

Índice

Introducción

El paso de mensajes es una técnica que se usa para, desde un proceso, invocar de forma abstracta un comportamiento concreto por parte de otro actor (Por ejemplo, ejecutar una función o un programa). Esto supone un gran cambio con respecto al modelo tradicional en el cual se llamaba a los programas y funciones solo por su nombre, el paso de mensajes constituye un mecanismo esencial para distinguir una función de su implementación interna— El emisor envía un mensaje solicitando una acción abstracta al destinatario, y es el destinatario el que, al recibir el mensaje, decidirá la acción que debe ejecutar para cumplir la petición.

Entre las ventajas que proporciona este mecanismo destacan el alto nivel de encapsulamiento y la distribución. En programación, la filosofía de encapsular la información consiste en que los objetos software deben poder invocar servicios a otros objetos, sin tener por qué conocer ni preocuparse ni de cómo ni de qué manera está implementada en su interior la resolución de dicha demanda. Esto deriva en un código más limpio, corto y fácil de comprender y mantener.

Caso práctico en programación orientada a objetos

Un ejemplo clásico de aplicación del modelo de paso de mensajes podría ser en el ámbito de la informática gráfica. Por ejemplo, si el programador está creando un programa que trabaja con diversos tipos de formas geométricas de una clase `Forma`, el programa contendrá varios objetos instancias de dicha clase, como por ejemplo `cuadrado`, `rombo` o `triangulo`.

Imaginemos el caso en que el programador desea conocer el área de los objetos de la clase `Forma` declarados. Como el cálculo del área de dichas formas no es el mismo para todas y cada una tendrá un algoritmo matemático distinto para calcularla, en un modelo tradicional el programador se vería obligado a crear una estructura condicional IF-ELSE que consultase una a una el tipo de forma de cada objeto para ejecutar la operación matemática correspondiente.

En lugar de eso, en el modelo de paso de mensajes el programador simplemente debe enviar un mensaje a cada objeto consultando su área, y sería la implementación interna de la clase `Forma` la que se encargaría de escoger y calcular de forma correcta dicho dato. El objeto respondería a la petición enviando de vuelta otro mensaje al destinatario con el área resultante del cálculo. Como es natural, para que esto sea posible, la clase destinataria debe tener implementado previamente un método por el cual pueda interpretar los mensajes de forma normalizada, así como las funciones necesarias para llevar a cabo el cálculo del área.[1]

Caso práctico en programación concurrente y distribuida

Otra de las aplicaciones donde este modelo supone una clara ventaja es en la programación concurrente y distribuida. En los casos en los que tengamos varios procesos ejecutándose en un mismo o en diferentes ordenadores con características y velocidad de trabajo dispares y estos procesos requieran comunicarse entre ellos para poder cumplir su función, el paso de mensajes entre ellos es una de las formas más ampliamente aceptadas de mantener una sincronización constante y coherente entre ellos. Una de las aplicaciones más conocidas es internet, donde una red de ordenadores dispares se comunican entre ellos para intercambiar datos en diferentes lugares y momentos.

El protocolo más extendido para acceder a páginas web — HTTP — también funciona bajo este modelo: en su caso, un ordenador al que se conoce como "servidor" contiene un dato (por ejemplo, una página web), y varios ordenadores independientes (llamados "clientes") desean acceder a dicho dato. Para llevar a cabo dicho propósito se produce un intercambio de mensajes normalizados cliente-servidor: En primer lugar el cliente debe enviar un mensaje al servidor informando de que desea acceder a un dato concreto de los que él tiene. El servidor recibe y ejecuta por su lado el código necesario para llevar a cabo la petición. A continuación, el servidor responde al cliente mediante el paso de otro mensaje, otorgándole el dato (en este caso, la página web) que ha solicitado.

Uno de los estándares de paso de mensajes más ampliamente utilizados en programación concurrente y distribuida es MPI (*Message Passing Interface*).

Paso de mensajes síncrono y asíncrono

Una de las cuestiones más importantes en el paso un mensaje es diferenciar si el envío y recepción se produce de forma síncrona o asíncrona. Para comprender esto, primero es necesario resaltar que el envío y recepción de un mensaje constituyen dos pasos completamente diferentes: por un lado, el proceso emisor ejecuta la función que envía el mensaje al proceso destinatario, y este ejecutará por su parte otra función que le permitirá recibirlo.

Se dice que el paso de mensajes es síncrono cuando el envío y la recepción del mensaje se producen en un mismo instante de tiempo para ambos procesos. En el caso de que el envío por parte del emisor del mensaje y la recepción de este por parte del destinatario se produzcan en instantes de tiempo diferentes, decimos que el paso de mensajes es asíncrono.

Seguridad

Un concepto fundamental relacionado con la diferencia entre el paso síncrono y asíncrono es la seguridad del paso del mensaje. El paso de un mensaje es seguro cuando podemos garantizar que el mensaje que recibe el destinatario es el mismo que envía el emisor, y que no se ha producido ninguna alteración del contenido del mensaje durante el lapso de tiempo que transcurre entre el envío y la recepción. El envío síncrono siempre es seguro, más el envío asíncrono no es seguro en sí mismo, por lo que en ese caso es el programador el responsable de garantizar la integridad del mensaje.

Paso de mensajes síncrono

En el paso de mensajes síncrono, el proceso emisor en primer lugar ejecutará la función de envío síncrono (Por ejemplo, en MPI: `s_send()`). Tras ese momento dicho proceso detendrá su ejecución en ese punto y se quedará bloqueado a la espera de que el proceso receptor reciba el mensaje. Cuando el proceso receptor llega a la instrucción en la cual se le ordena que reciba el mensaje (`receive()`), entonces comienza la transmisión del mensaje por parte de ambos. Ambos procesos permanecerán bloqueados hasta que la transmisión del mensaje haya finalizado correctamente. Cuando suceda, ambos serán desbloqueados y podrán continuar cada uno ejecutando su código. En el caso de que el programa receptor llegue a la instrucción de recibir antes de que el emisor haya llegado a su instrucción de enviar el mensaje, el receptor se quedará bloqueado a la espera del emisor.

El paso de mensajes síncrono siempre es seguro, ya que el bloquearse ambos procesos hasta que la recepción finaliza, podemos garantizar que no hay posibilidad de que el mensaje haya podido ser alterado. Sin embargo, el sistema, a pesar de ser sencillo de programar tiene el inconveniente de ser poco flexible: Ambos procesos deben "citarse", es decir, deben esperar hasta que ambos estén en el mismo punto de su código para poder realizar el mensaje. Como no podemos garantizar que el tiempo que transcurrirá entre que el emisor llegue a su instrucción s_send() y el receptor llegue a su instrucción receive() sea razonable, este método en algunos casos podría ver mermado el rendimiento del conjunto de ambos.

Por otro lado, una mala programación podría llevar a que un proceso ejecutara una función de envío que no tiene su correspondiente función de recibir o viceversa, con lo cual los procesos podrían llegar a quedar bloqueados indefinidamente esperando el uno al otro (interbloqueo).

Paso de mensajes asíncrono

El envío asíncrono es aquel en que los procesos no necesitan ser "citados" para que se produzca el envío del mensaje. Es decir, cuando el emisor llega a la instrucción en la que se produce el envío, no se bloquea a la espera de que el programa destinatario llegue a la instrucción en que lo recibe, sino que sigue ejecutando su código con normalidad. Por su parte, el programa receptor podrá recibir el mensaje en cualquier momento posterior al envío sin frenar por ello al emisor hasta que esto se produzca. Una vez más, si el receptor ha llegado a una orden de recepción de mensaje pero el emisor aún no ha enviado nada, el receptor sí se bloqueará a la espera de que el emisor lleve a cabo el envío.

El que el emisor no quede bloqueado tras enviar el mensaje es una ventaja con respecto a la eficiencia general del sistema, pero puede conllevar un riesgo de seguridad en el paso del mensaje. Por ejemplo, pongamos el caso de que el proceso emisor tiene una variable local `n = 2` y desea enviar el contenido de esta variable a un proceso receptor. El proceso ejecutará su instrucción de envío de la variable y continuará su código con normalidad. Al no poder conocer cuanto tiempo transcurrirá hasta que el receptor reciba la variable, podría suceder que desde que el emisor ejecutó la orden de envío y hasta que se produjo la recepción, el código del emisor ha alterado el contenido de la variable (incluso podría suceder que ya la hubiese eliminado) `n`, por lo que no podemos garantizar la seguridad del envío al no poder afirmar que el emisor recibirá el valor que realmente se quería enviar, el `2`.

Envío asíncrono seguro

Habitualmente, los estándares incluyen un mecanismo de paso de mensajes que, siendo asíncrono, garantiza la seguridad del envío. Para ello, el emisor ejecuta la función correspondiente (`send()`), copia los datos en un lugar seguro para poder evitar que estos se puedan alterar, y a continuación continúa su ejecución con normalidad. En el momento en que el proceso receptor ejecute su orden para recibir el mensaje (`receive()`), recibirá el dato que el emisor copió en su momento en un lugar seguro, por lo que se garantiza que este dato no ha sido alterado durante el tiempo transcurrido.

Envío asíncrono inseguro

Además de las funciones de envío asíncrono seguro, las interfaces de paso de mensajes también suelen incorporar funciones que permitan pasar mensajes de forma insegura. En su caso, el proceso emisor ejecutará la orden de envío inseguro (`i_send()`), se dará la orden de envío e instantáneamente se devolverá el control al programa principal, sin esperar a que los bytes hayan terminado de ser leídos del emisor ni escritos en el receptor. Por su parte, el receptor ejecutará la orden de recepción insegura (`i_send`), por lo que se dará la orden de recepción y a continuación se devolverá el control al programa, sin esperar a que se complete el envío ni a que terminen de llegar los datos.

En este caso, el responsable de garantizar manualmente la integridad del mensaje es el programador. Las interfaces de paso de mensajes incorporan una serie de funciones que permiten al programador sondear el estado del paso del mensaje y actuar en consecuencia. Concretamente, se dispone de funciones de sondeo bloqueante (`wait_send()` y `wait_recv()`), las cuales bloquean al emisor y receptor respectivamente si el envío o recepción del mensaje no ha finalizado aún. El programador también dispone de funciones de sondeo no bloqueante (`test_send()` y `test_recv()`) que permiten conocer este mismo hecho sin bloquear a ninguno de los procesos, simplemente devolviendo un valor booleano para que sea el programador el que decida la acción a tomar en cada caso.

Véase también

- Programación distribuida
- Programación orientada a objetos
- Interfaz de paso de mensajes

Referencias

- Universidad de Granada, ed. (2012). *Sistemas concurrentes y distribuidos*.

1. Goldberg, Adele; David Robson (1989). *Smalltalk-80 The Language*. Addison Wesley. pp. 5-16. ISBN 0-201-13688-0.
