Patrones de Integración

Las aplicaciones actuales interaccionan unas con otras y le hacen frente a los siguientes problemas:

- Las redes son lentas y no fiables
- Las aplicaciones son diferentes: Framework, sistema, codificación y codificación de los datos.
- El cambio es inevitable, las aplicaciones evolucionan constantemente.

Los desarrolladores suelen adoptar las siguientes soluciones:

- Transferencia de ficheros.
- Base de datos compartida.
- Invocación de procesos remotos.
- Envío de mensajes.

Elementos

Mensaje

es una estructura de datos, puede ser un string, byte array, registro o un objeto Se puede interpretar como:

- Dato
- Una descripción de orden a ejecutar en el receptor
- Descripción de un evento producido en el emisor

Tiene 2 partes:

- Body
- Header

Canal

Son vías lógicas que conectan los programas y el listado de mensajes.

Se comportan como una colección de mensajes.

Se comparten entre ordenadores.

Múltiples aplicaciones pueden usarlos simultáneamente.

Sender y Receiver

Lo que serían el emisor y el receptor

Sistema de mensajería

El envío de mensajes se realiza a través de un sistema de mensajería. Hay que diferenciar entre sistema de mensajería y una base de datos:

- Un sistema de mensajería gestiona mensajes
- Un gestor de base de datos gestiona persistencia

Pasos para transmitir un mensaje

1. Create: El emeisor crea un mensaje y guarda los datos
2. Send: el emisor añade el mensaje al canal
3. Deliver: El sistema de mensajería lo entrega al receptor
4. Receive: El receptor lee el mensaje del canal
5. Process: el receptor extrae los datos del mensaje

Conceptos importantes

- Send and Forget: el emisor envía y olvida
- Store and forward:
    - En el paso 2 el emisor envía y el sistema de mensajería guarda el mensaje en memoria o en disco
    - En el paso 3 el sistema de mensajería entrega el mensaje

Por que usar mensajería

- Es más inmediato que la transferencia de ficheros
- Mejor encapsulado que compartir una base de datos
- Más seguro que la invocación remota de procedimientos (RPC)
- Permite que dos aplicaciones se comuniquen y transfieran datos sin serializar objetos
- Permite la comunicación entre aplicaciones con distinto lenguaje, tecnología y plataformas
- Comunicaciones asíncronas → envío y me olvido
- Variable timing, emisor y receptor trabajan a su ritmo
- Mejor rendimiento del receptor que trabaja a su ritmo
- Comunicación garantizada.
- Permite operaciones desconectadas en sistemas cuya conexión a la red es limitada
- Un sistema de mensajería puede actuar como mediador entre diversas aplicaciones
- Se puede crear un sistema de bloqueo más eficiente cuando se espera un callback.

Retos del envío asíncrono

- Modelo de programación más complejo
- No tenemos una secuencia de mensajes: no se puede asegurar cual va el primero y cual va el último

- Escenarios asíncronos: no se puede usar en escenarios síncronos
- Pérdida de rendimiento al enviar muchos datos.
- Protocolos propios de cada fabricante.

El problema de esto es que cambia el modelo de programación ya que estamos hablando de comunicaciones asíncronas. Para solucionar esto se mete un sistema de mensajería con una URL de retorno o leemos de otro sistema de mensajería para que nos confirme la recepción y procesamiento del mensaje. La URL invocaría un servicio para indicar que hemos terminado y continuar.

Implicaciones

- No existe un único hilo de ejecución, varios subprocesos tienen subprocedimientos
- El emisor tiene que procesar los resultados incluso cuando está con otra tarea
- El emisor debe detectar que subproceso generó el resultado recibido y combinarlo con los otros resultados
    - No se conoce el orden de llegada

Motivación

- Un patrón es una solución general correcta para un problema repetido y común.
- No es la solución en sí, es un esquema o una guía.

From:
https://knoppia.net/ - **Knoppia**

Permanent link:
https://knoppia.net/doku.php?id=dad:patrones_de_integracion&rev=1702297806

Last update: **2023/12/11 12:30**