UNIVERSIDAD POLTÉCNICA DE PACHUCA

Programación Concurrente

Profesor: Marco Waldo Angeles TenaGrupo: SFTW_07_01

Cuatrimestre: 7

Grupo:01

Los patrones de programación asincrona futuro (future y promesa (Promise)

Son paradigmas usados para manejar tareas asincronas en sistemas que requieren concurrencia. Son especialmente útiles cuando se deja realizar operaciones que tardan en completarse (como operaciones de red, consultas en bd o procesamiento intensivo), sin bloquear el flujo principal del programa.

Conceptos Clave

Promesa

- Es un objeto que actúa como un compromiso de que una operación asíncrona se completará en el futuro. Cuando una tarea o cálculo asincrona comienza se crea una promesa que indica que en algún momento se resolverá.
- Tiene dos estados:
  - Resuelta (fulfilled): La operación asincrona ha terminado con éxito, y el resultado está disponible.
  - Rechazada (rejected): La operación asincrona ha fallado, y hay un error.

Future

- Es un objeto que representa el resultado de una operación asincrona que aún no ha terminado. Es como un marcador de posición: Cuando se inicia una operación que eventualmente producirá un resultado, se devuelve un futuro que puede ser usado para obtener ese resultado más adelante.
- El futuro permite:
  - Consultar si la operación ha finalizado.
  - Esperar a que la operación termine para obtener el resultado
  - Recuperar el resultado o manejar errores cuando estén disponibles.

Relación entre Futuro y Promesa:

- La promesa es quien realiza la tarea o acción asincrona, y la future es lo que el cliente (usuario) recibe.
- La promesa llena el valor de que el resultado llegará más adelante
- El programador puede seguir trabajando mientras espera el resultado, y luego puede usar el futuro para obtener el valor.

FIRST CLASS

Ejemplo en Pseudocódigo

1.- Promesa (Lado Servidor):

```
function consultaBaseDatos() -> Promise:
Promesa = nuevo Promesa()

  hiloSecundario.lanzar(() -> {
    // Simular un procesamiento largo, por ejemplo, 3 segundos
    esperar(3 segundos)
    // Resolver la promesa con el resultado de la BD
    promesa.resolver(resultado_de_la_base_de_datos)
  })

  retornar promesa
```

2.- Futuro (Lado del cliente)

```
futuro = consultaBaseDatos()
  // Consulta haciendo otras cosas mientras la consulta esta en proceso
  hacerOtrasCosas()

// En algun punto necesitamos el resultados de la consulta
resultado = futuro.obtener()
imprimir(resultados)
```

En este ejemplo, la función consultaBaseDatos devuelve la promesa que será resultado en el futuro. El cliente recibe futuro y puede seguir haciendo otras cosas mientras la base de datos realiza la consulta en segundo plano.

Ventajas del Patrón Futuro/Promesa

1.- No bloqueo: Permite que el flujo principal del programa continúe sin bloquearse mientras se esperan resultados.

2.- Mejora legibilidad: Evita el uso excesivo de callbacks o estructuras complejas para manejar operaciones asíncronas, proporcionando una interfaz más clara

3.- Manejo de errores: Permite un manejo centralizado de errores asincronos cuando algo falla, lo que simplifica el código.

4.- Escalabilidad: Facilita la gestión de múltiples tareas concurrentes (como en servidores web que manejan muchas conexiones simultaneas.

Ejemplo

```
Import concurrent.futures
import time

def operacion_lenta():
    print("Comenzando operación lenta...")
    time.sleep(3)
    return "Resultado listo"

with concurrent.futures.ThreadPoolExecutor() as executor:
    futuro = executor.submit(operacion_lenta)

    print("Haciendo otras cosas mientras esperamos el resultado")

    resultado = futuro.result()
    print(f"El resultado de la operación es: {resultado}")
```

Salida Esperada

```
- Comenzando operación lenta...
- Haciendo otras cosas mientras espero
  el resultado.
- El resultado de la operacion es:
  Resultado listo
```

Explicación Código

```python
# Importamos los modulos necesarios:
# 'concurrent.futures' para manejar la concurrencia (hilos o procesos)
# 'time' para poder simular pausas (operaciones que tardan)
import concurrent.futures
import time

# --- Definicion de la funcion ---
# Esta funcion simula una tarea que tarda un tiempo en completarse.
def operacion_lenta():
    # Imprime un mensaje para saber cuando empieza la tarea
    print("Comenzando operacion lenta...")

    # time.sleep(3) pausa la ejecucion de este hilo durante 3 segundos.
    # Esto simula un trabajo pesado, como una consulta a base de datos,
    # una llamada a una API web o un calculo complejo.
    time.sleep(3)

    # Una vez que la pausa termina, la funcion retorna un valor.
    return "¡Resultado listo!"

# --- Ejecucion principal del script ---

# Usamos 'with concurrent.futures.ThreadPoolExecutor() as executor:'
# Esto crea un "pool" (un grupo) de hilos (workers).
# El 'with' asegura que todos los hilos se limpien y cierren correctamente
# al salir de este bloque, incluso si ocurren errores.
with concurrent.futures.ThreadPoolExecutor() as executor:
```

```python
    # 'executor.submit(operacion_lenta)' le dice al pool de hilos:
    # "Toma la funcion 'operacion_lenta' y ejecutala en uno de tus hilos disponibles"
    # Esto NO bloquea el hilo principal. La ejecucion de 'operacion_lenta'
    # comienza inmediatamente en segundo plano (en otro hilo).
    #
    # La variable 'futuro' (un objeto Future) es una "promesa".
    # Representa el resultado que eventualmente tendra la operacion lenta.
    futuro = executor.submit(operacion_lenta)

    # Dado que la linea anterior no bloqueo, el hilo principal
    # puede seguir ejecutando otras cosas inmediatamente.
    print("Haciendo otras cosas mientras esperamos el resultado...")
    print("Por ejemplo, calcular 1 + 1 =", 1 + 1)

    # --- Esperando el resultado ---

    # 'futuro.result()' es la linea que SI bloquea el hilo principal.
    # Le dice al script: "Espera aqui hasta que la tarea en segundo plano
    # (representada por 'futuro') haya terminado y me entregue su valor".
    #
    # Si la tarea ya termino (porque tardamos mucho en "Haciendo otras cosas"),
    # devuelve el resultado al instante.
    # Si la tarea (los 3 segundos) aun no termina, se detiene aqui hasta que lo haga.
    resultado = futuro.result()

    # Una vez que 'futuro.result()' nos da el valor, lo imprimimos.
    # Esto solo se imprimira despues de que los 3 segundos hayan pasado.
    print(f"El resultado de la operacion es: {resultado}")

print("El programa ha terminado.")
```