PRACTICA

21/11/2024

“Comparativa y Uso de SOAP y REST”

Ahmed Hassan Khamis

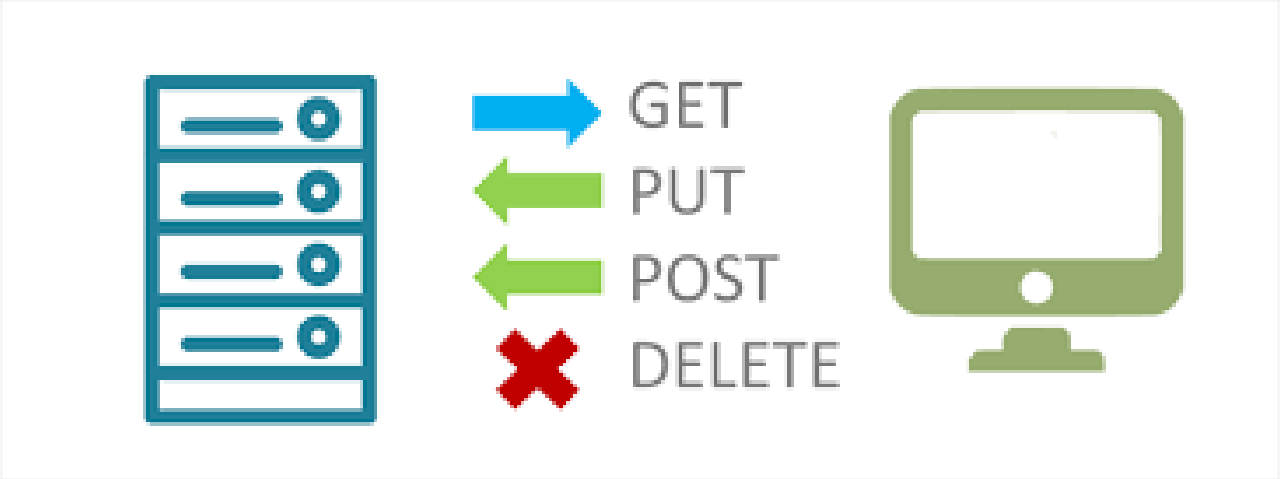
Comparativa entre SOAP y REST Web Services

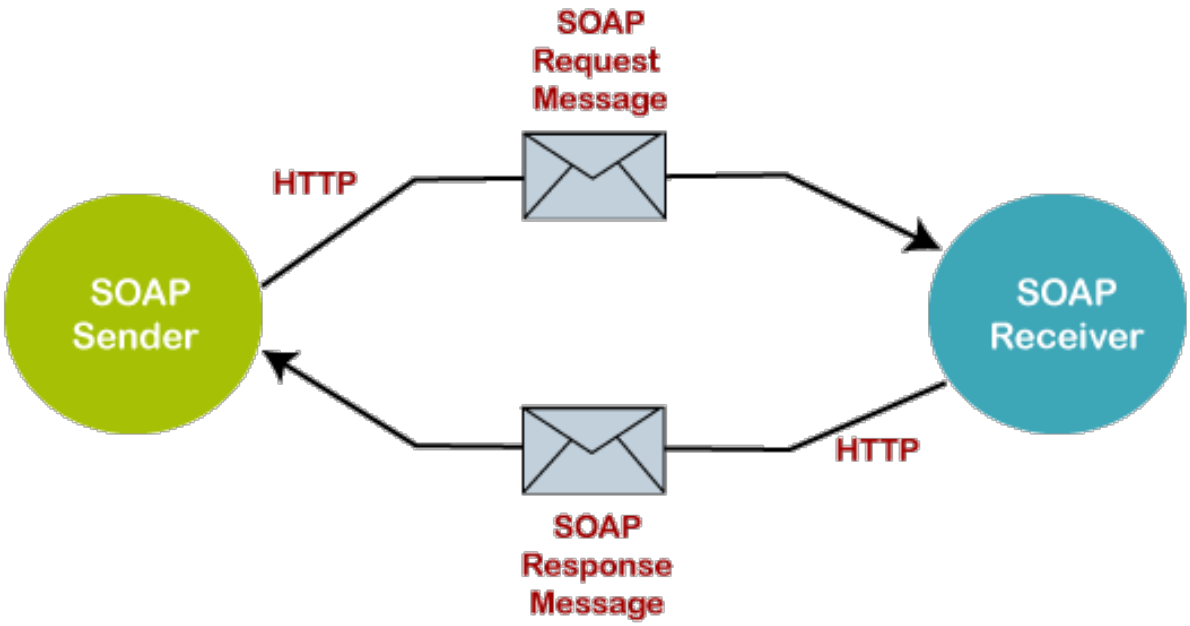
A continuación, se presenta una comparación detallada entre **SOAP** y **REST**, seguida de ejemplos de uso para cada tecnología.

---

1. Protocolo vs Estilo Arquitectónico

- **SOAP:**
  Es un **protocolo** estándar definido por el W3C, diseñado para servicios empresariales con estrictas reglas de comunicación y formato (basado en XML).
- **REST:**
  Es un **estilo arquitectónico** que aprovecha el protocolo HTTP, haciendo uso de sus métodos estándar (GET, POST, PUT, DELETE).





2. Formato de Mensaje

- **SOAP:**
  Los mensajes están en formato XML y requieren envolturas específicas, como `<SOAP-Envelope>`. Esto introduce más sobrecarga.
  Ejemplo de un mensaje SOAP:

```
<SOAP-Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-Body>
        <getWeather>
            <city>Madrid</city>
        </getWeather>
    </SOAP-Body>
</SOAP-Envelope>
```

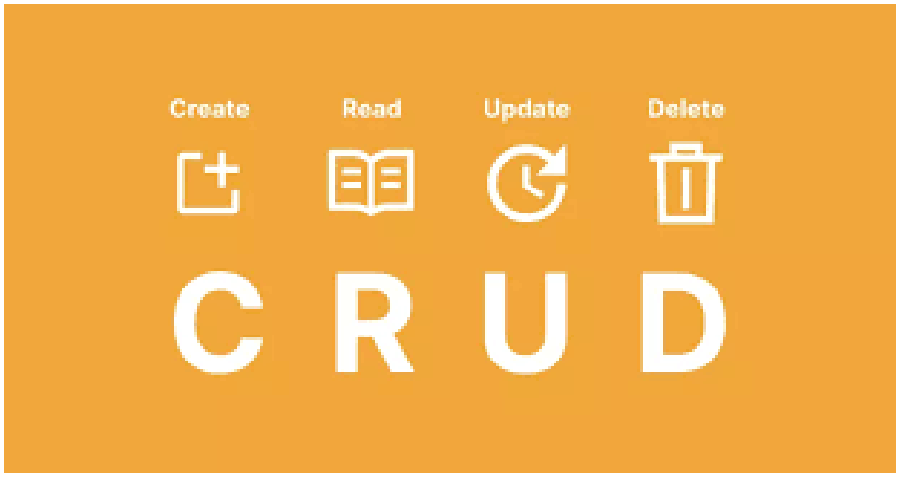
- **REST:**
  Soporta múltiples formatos como JSON, XML, o texto plano. JSON es el más utilizado debido a su simplicidad y menor tamaño.
  Ejemplo de una solicitud REST en JSON:

```
{
    "city": "Madrid"
}
```

---

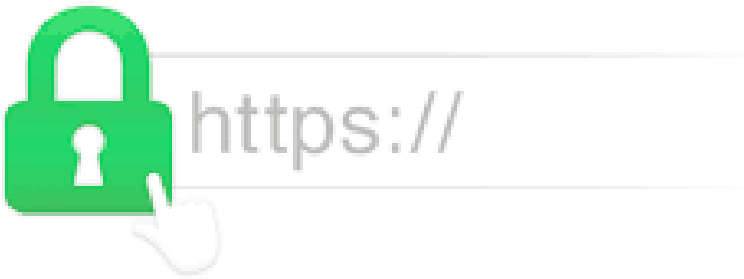
3. Modo de Operación

- **SOAP:**
  Más adecuado para operaciones complejas que requieren confiabilidad, estado y transacciones (por ejemplo, procesamiento bancario o sistemas de pago).
- **REST:**
  Ideal para operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en aplicaciones web y móviles, donde la rapidez y la escalabilidad son esenciales.



4. Seguridad

- **SOAP:**
  Ofrece **seguridad avanzada** a nivel de mensaje a través de WS-Security, que permite cifrar partes específicas de los datos. Esto es crucial para aplicaciones críticas.
- **REST:**
  Depende de **HTTPS** para garantizar la seguridad en el transporte, lo que es suficiente para la mayoría de las aplicaciones modernas.



5. Compatibilidad

- **SOAP:**
  Funciona con diversos protocolos de transporte, incluidos HTTP, SMTP y más.
- **REST:**
  Está estrechamente vinculado al protocolo HTTP, lo que lo hace simple, pero limitado en términos de transporte.

6. Escalabilidad y Rendimiento

- **SOAP:**
  Consume más ancho de banda debido al uso de XML y es menos eficiente para aplicaciones móviles o de bajo consumo.
- **REST:**
  Más rápido y ligero, especialmente con JSON, lo que lo hace adecuado para aplicaciones modernas y escalables.

7. Facilidad de Implementación

- **SOAP:**
  Requiere herramientas y bibliotecas específicas (por ejemplo, WSDL para describir servicios y SOAP UI para pruebas). Es más complejo.
- **REST:**
  Puede ser consumido por cualquier cliente HTTP estándar (navegador, curl, Postman, etc.), haciéndolo más accesible.

Casos de Uso

SOAP - Ideal para:

1. Aplicaciones empresariales complejas.
2. Servicios que requieren **transacciones seguras** (por ejemplo, WS-AtomicTransaction).
3. Entornos donde se necesita interoperabilidad estricta entre sistemas (por ejemplo, comunicaciones entre gobiernos o bancos).

REST - Ideal para:

1. Aplicaciones web y móviles rápidas y ligeras.
2. Servicios que manejan datos dinámicos y simples (por ejemplo, APIs públicas como las de Google Maps o Twitter).
3. Microservicios y sistemas distribuidos.

Ejemplo Práctico

SOAP: Consultar el saldo de una cuenta bancaria

- **Servidor:** Provee un WSDL para que los clientes entiendan cómo comunicarse.
- **Cliente:** Envia una solicitud SOAP para obtener el saldo.

```
<SOAP-Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-Body>
        <getAccountBalance>
            <accountNumber>12345</accountNumber>
        </getAccountBalance>
    </SOAP-Body>
</SOAP-Envelope>
```

REST: Consultar el clima en una ciudad

- **Cliente:** Envía una solicitud HTTP GET con el parámetro `city` para consultar el clima.
- **Solicitud HTTP:**

  ```
  GET /weather?city=Madrid HTTP/1.1
  Host: api.weather.com
  ```

- **Respuesta JSON:**

```
{
    "city": "Madrid",
    "temperature": "20°C",
    "condition": "Sunny"
}
```

Conclusión

Característica	SOAP	REST
Estandarización	Protocolos rígidos (WSDL, WS-Security)	Flexible, usa estándares HTTP
Formato	Solo XML	JSON, XML, Texto Plano
Complejidad	Alta	Baja
Velocidad	Menor, más pesado	Rápido y ligero
Seguridad avanzada	Sí (WS-Security)	Limitada a HTTPS

La elección depende del contexto: SOAP para servicios críticos y REST para aplicaciones modernas, rápidas y ligeras.

EJEMPLO EN JAVA DE REST

NOTA: el siguiente ejemplo esta traído a partir de un pequeño esquema en java, luego pasado por una ia para darle lo necesario y restante.

Aquí tienes un ejemplo muy sencillo de un cliente REST en Java usando la biblioteca **HttpURLConnection**, que es parte del JDK estándar, para evitar dependencias adicionales. Este cliente consume un servicio REST que devuelve información sobre el clima.

Ejemplo de Cliente REST en Java

Código del Cliente

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class SimpleRestClient {
    public static void main(String[] args) {
        try {
            // URL del servicio REST (reemplázala con la URL de tu servicio)
            String apiUrl = "https://api.weatherapi.com/v1/current.json?
key=YOUR_API_KEY&q=Madrid";

            // Crear la conexión
            URL url = new URL(apiUrl);
```

```
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();

            // Configurar la solicitud
            connection.setRequestMethod("GET");
            connection.setRequestProperty("Accept", "application/json");

            // Verificar el código de respuesta
            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) { // Código 200
                // Leer la respuesta
                BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
                String inputLine;
                StringBuilder response = new StringBuilder();

                while ((inputLine = in.readLine()) != null) {
                    response.append(inputLine);
                }
                in.close();

                // Imprimir la respuesta
                System.out.println("Respuesta del servidor:");
                System.out.println(response.toString());
            } else {
                System.out.println("Error en la conexión: Código " +
responseCode);
            }

            // Cerrar la conexión
            connection.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Explicación del Código

1. **Configuración de la URL:**
    - La URL apunta al servicio REST. Puedes usar un servicio real como WeatherAPI o tu propio servicio local.
    - Reemplaza `YOUR_API_KEY` con una clave de API válida si usas un servicio externo.
2. **Establecer la conexión:**
    - Se utiliza `HttpURLConnection` para abrir la conexión y configurar el método de solicitud (`GET`).
3. **Leer la respuesta:**
    - Si el servidor responde con código **200 (HTTP_OK)**, se lee el cuerpo de la respuesta línea por línea utilizando un `BufferedReader`.

4. **Mostrar el resultado:**
    - El cliente imprime la respuesta JSON que devuelve el servicio.

Salida Esperada

Si el servicio devuelve información sobre el clima de Madrid, podrías obtener algo como esto:

```
{
    "location": {
        "name": "Madrid",
        "region": "Madrid",
        "country": "Spain"
    },
    "current": {
        "temp_c": 20.0,
        "condition": {
            "text": "Sunny"
        }
    }
}
```

La consola mostrará:

```
Respuesta del servidor:
{
    "location": {
        "name": "Madrid",
        "region": "Madrid",
        "country": "Spain"
    },
    "current": {
        "temp_c": 20.0,
        "condition": {
            "text": "Sunny"
        }
    }
}
```

Nota:

Para servicios más complejos o si deseas manejar automáticamente JSON, puedes usar bibliotecas como **Gson** o **Jackson** para convertir las respuestas JSON a objetos Java. Pero este ejemplo mantiene la simplicidad y utiliza solo clases del JDK estándar.